Partial Information Network Queries[☆]

Ron Y. Pinter, Hadas Shachnai, Meirav Zehavi*

Department of Computer Science, Technion - Israel Institute of Technology, Haifa 32000, Israel

Abstract

We study the *Partial Information Network Query (PINQ)* problem, which generalizes two problems that often arise in bioinformatics: the *Alignment Network Query (ANQ)* problem and the *Topology-Free Network Query (TFNQ)* problem. In both ANQ and TFNQ we have a pattern $\mathcal{P}$ and a graph H, and we seek a subgraph of H that *resembles* $\mathcal{P}$. ANQ requires knowing the topology of $\mathcal{P}$, while TFNQ ignores it. PINQ fits the scenario where partial information is available on the topology of $\mathcal{P}$. Our main result is a parameterized algorithm that handles inputs for PINQ in which $\mathcal{P}$ is a set of trees. This algorithm significantly improves the best known O^* running time in solving TFNQ. We also improve the best known O^* running times in solving two special cases of ANQ.

Keywords: parameterized algorithm, pattern matching, partial information network query, alignment network query, topology-free network query

1. Introduction

Algorithms for the *Alignment Network Query (ANQ)* and *Topology-Free Network Query (TFNQ)* problems provide means to study the function and evolution of biological networks. Given a pattern $\mathcal{P}$ and a host graph H, these queries seek a subgraph of H that *resembles* $\mathcal{P}$. Today, with the increasing amount of information we have on biological networks, ANQ and TFNQ are becoming widely spread (see, e.g., [2] and [3]). We note that similar queries for sequences have been studied and used extensively in the past four decades [4].

TFNQ, also known as *Graph Motif*, requires only the connectivity of the solution, while ANQ requires resemblance between the *topology* of $\mathcal{P}$ and the solution. A user having partial information on the topology of $\mathcal{P}$ can either

[☆]A preliminary version of this paper appeared in the proceedings of the 24$^{\text{th}}$ *International Workshop on Combinatorial Algorithms (IWOCA'13)* [1].

Abbreviations: Partial Information Network Query (PINQ), Alignment Network Query (ANQ), Topology-Free Network Query (TFNQ).

*Corresponding author.

Email addresses: pinter@cs.technion.ac.il (Ron Y. Pinter), hadas@cs.technion.ac.il (Hadas Shachnai), meizeh@cs.technion.ac.il (Meirav Zehavi)

run an alignment network query for each possible topology for $\mathcal{P}$, given this partial information, or run a topology-free network query. The first method is inefficient, while the second may output undesirable results that contradict the partial information on $\mathcal{P}$. We present a generalization of ANQ and TFNQ, that we call the *Partial Information Network Query (PINQ)* problem, which fits the scenario where only partial information is available on $\mathcal{P}$.

Parameterized algorithms are an approach to solve NP-hard problems by confining the combinatorial explosion to a parameter k. More precisely, a problem is *fixed-parameter tractable (FPT)* with respect to a parameter k if an instance of size n can be solved in time $O^*(f(k))$ for some function f [5].[1]

In this paper we present parameterized algorithms for NP-hard special cases of PINQ. In particular, we significantly improve the best known O^* running time in solving TFNQ.

Notation: Given a graph G, let $V(G)$ and $E(G)$ denote its node set and edge set, respectively. Given $U \subseteq V(G)$, let $G[U]$ denote the subgraph of G induced by U. Given a node v, let $l(v)$ and $N(v)$ denote its label and neighbor set, respectively. Given a set of tuples A, $i \in \mathbb{N}$ and an element e, let $A[(i, e)]$ denote the set of tuples in A such that e appears in their i^{th} position.

1.1. Problem Statement

Roughly speaking, given a graph H and a set of graphs $\mathcal{P}$, PINQ asks if H has a connected subgraph that can be partitioned into subgraphs, such that each resembles a different graph in $\mathcal{P}$.

Formally, the input for PINQ consists of

- L - A set of labels.
- $\Delta : L \times L \to \{-\infty\} \cup \mathbb{R}$ - A label-to-label similarity score table.
- $\mathcal{P}$ - A set of labeled graphs $P_1, P_2, \ldots, P_t$.
- $H, w : E(H) \to \mathbb{R}$ - An edge-weighted labeled graph.
- $W \in \mathbb{R}$ - A minimum score (for a solution).

A solution consists of a connected subgraph S of H, a partition of $V(S)$ into the subsets $\{V_S^1, \ldots, V_S^t\}$, and an isomorphism m_i between $S[V_S^i]$ and P_i, for all $1 \leq i \leq t$, such that

- $\sum_{1 \leq i \leq t} \sum_{v \in V_S^i} \Delta(l(v), l(m_i(v))) + \sum_{e \in E(S)} w(e) \geq W$.
- Any cycle in S is completely contained in $S[V_S^i]$, for some $1 \leq i \leq t$.[2]

Let $V(\mathcal{P}) = \bigcup_{1 \leq i \leq t} V(P_i)$, and $k = |V(\mathcal{P})|$.

Special Cases of PINQ: We note that ANQ is the special case where $t = 1$, and all the edge-weights are 0. Moreover, TFNQ is the special case where $t = k$, and $\Delta(l, l') \in \{-\infty, 0\}$ for all $l, l' \in L$.

[1] O^* hides factors polynomial in the input size.

[2] We thus avoid solving a generalization of the Clique problem, which is W[1]-hard [6].

Reference	Running Time	Method
Guillemot et al. [20]	$O^*(4^k W^2)$	multilinear detection [21]
Pinter et al. [11]	$O^*(2^k W)$	narrow sieves [22]
Bruckner et al. [19]	$O^*(k!3^k)$	color coding [12]
This paper	$\mathbf{O^*(20.25^{k+O(\log^2 k)})}$	**randomized divide-and-conquer [23]**

Table 1: Parameterized algorithms for (weighted) TFNQ. The first two algorithms require W and the edge-weights to be nonnegative integers, and their running times depend on the numeric value of W.

1.2. Prior Work

The Alignment Network Query Problem: ANQ is NP-hard even if the single graph in $\mathcal{P}$ is a path, since this case generalizes the Hamiltonian path problem [7].

Pinter et al. [8] gave a polynomial time algorithm that handles inputs for ANQ in which both P_1 and H are trees. This algorithm was used to perform inter-species and intra-species alignments of metabolic pathways [9], and a pathway evolution study [10]. Moreover, it was recently extended to handle a certain family of DAGs [11].

Another approach, based on color coding [12], enables H to be a general graph, and provides parameterized algorithms with parameter k. This approach is used by QPath [13] to perform simple path queries in time $O^*(5.437^k)$. QNet [14] extends QPath by allowing P_1 to be a graph whose treewidth tw is bounded. Its running time is $O^*(8.155^k|V(H)|^{tw+1})$. PADA1 [15] is an alternative to QNet that bounds the size fvs of the feedback vertex set of P_1 instead of its treewidth. Its running time is $O^*(8.155^k|V(H)|^{fvs})$. Hüffner et al. [16] reduced the running time of QPath to $O^*(4.314^k)$. All of these algorithms are randomized.

We note that there are several problems related to ANQ that have applications in bioinformatics, and refer the reader to the surveys [2, 17] for the precise details.

The Topology-Free Network Query Problem: Unweighted TFNQ (i.e., TFNQ restricted to inputs in which all the edge-weights are 0) was introduced by Lacroix et al. [18], and TFNQ was introduced by Bruckner et al. [19]. Lacroix et al. [18] proved that unweighted TFNQ is NP-hard even if H is a tree. On the positive side, TFNQ, when parameterized by k, is in FPT [19].

In recent years, many papers investigated the parameterized complexity of unweighted TFNQ and other closely related problems (see [24, 25, 26, 27, 28, 19, 29, 30, 31, 20, 32, 11]); yet, prior to this paper, the algorithm of best O^* running time for TFNQ that does not depend on the numeric value of W remained that of Bruckner et al. [19], running in time $O^*(k!3^k)$. Table 1 presents known parameterized algorithms for (weighted) TFNQ. All of the algorithms are randomized.

1.3. Our Contribution

Our first algorithm, ANQ-Alg, handles inputs for PINQ in which H is a general graph and $\mathcal{P}$ is a set consisting of a single tree. ANQ-Alg runs in time $O^*(6.75^k)$, which improves the O^* running times of QNet and PADA1 for inputs where P_1 is a tree. For the special case in which P_1 is a path, ANQ-Alg runs in time $O^*(4^k)$, which further improves the O^* running time of QPath. ANQ-Alg is based on the randomized divide-and-conquer method [23].

Our main result is the second algorithm, PINQ-Alg, which builds on ANQ-Alg. PINQ-Alg handles inputs for PINQ in which H is a general graph and $\mathcal{P}$ is a set of trees. It runs in time $O^*(6.75^{k+O(\log^2 k)}3^t)$. In particular, it solves TFNQ in time $O^*(20.25^{k+O(\log^2 k)})$ (since then $t = k$), which significantly improves the previous best $O^*(k!3^k)$ running time of Bruckner et al. [19].

2. An Algorithm for ANQ

We start by presenting an algorithm, that we call ANQ-Alg, for the special case of PINQ where $\mathcal{P}$ is a set consisting of a single tree (i.e., $t = 1$ and P_1 is a tree). In Section 2.1, we give an overview of ANQ-Alg. Section 2.2 includes some definitions required for the pseudocode of ANQ-Alg (given in Section 2.3). Finally, in Section 2.4, we prove the correctness and analyze the running time of ANQ-Alg.

2.1. Overview

We use the randomized divide-and-conquer method of [23]: Our algorithm ANQ-Alg randomly divides the problem into two smaller subproblems that it recursively solves, and then combines the answers. Next, by using Fig. 1, we describe and illustrate a recursive stage in more detail.

Each recursive stage concerns a rooted subtree R of P_1, a set $U \subseteq V(H)$ and a set *Solved* of rooted subtrees of R, such that the subgraph R' of R induced by the nodes in R that do not belong to any tree in *Solved* and the roots of the trees in *Solved* is a tree. For example, part A of Fig. 1 illustrates an input for algorithm ANQ-Alg and a recursive stage, where *Solved* contains the squares, triangles and hexagons trees, and R' is the subtree of R induced by the bold nodes. Each tree in *Solved* has several pairs, where each such pair consists of a node $h \in V(H)$ and a score s, and it concerns an isomorphism of score s between the tree (to which the pair belongs) and a subtree of H that maps the root of the tree to h. For example, part A of Fig. 1 illustrates the pairs $(c, 2)$ and $(e, 2)$ that belong to the squars tree.

Using such pairs, algorithm ANQ-Alg computes scores of isomorphisms between R and subtrees of H by computing scores of isomorphisms between the smaller tree R' and subtrees of H which map the nodes in R' (excluding its root) to nodes in U. In the base cases of the recursion, R' contains at most two nodes, and thus it can be easily mapped. In the step of the recursion, algorithm ANQ-Alg divides R' into two subtrees that have a common node, and randomly divides U into two subsets. Algorithm ANQ-Alg uses the first subset to map the

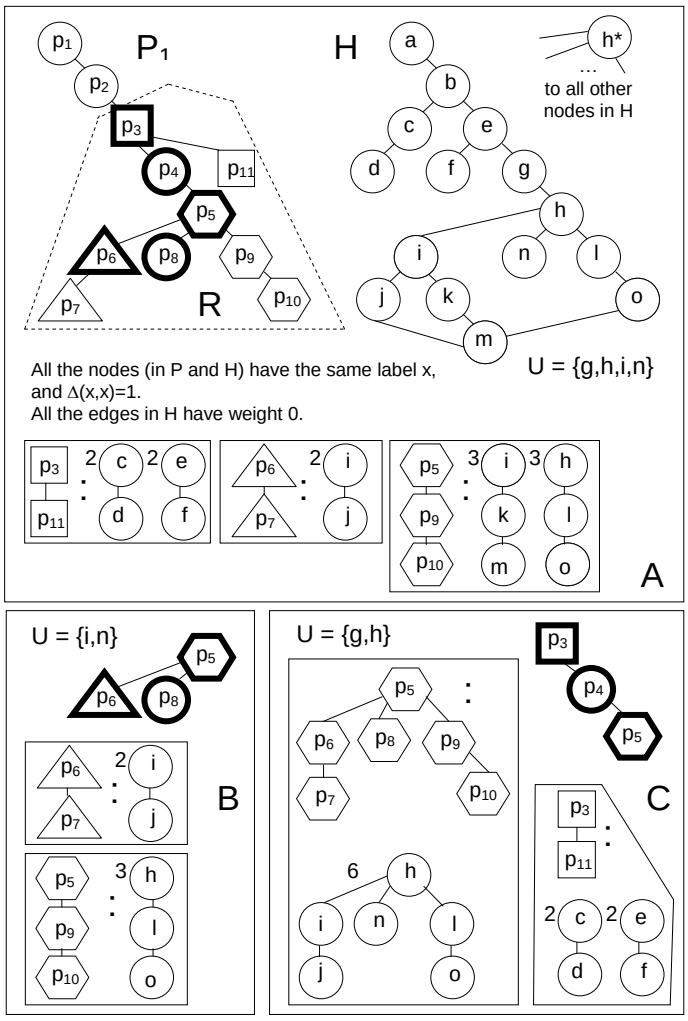

Figure 1: An illustration of a randomized divide-and-conquer step in ANQ-Alg.

first subtree; then it uses the corresponding results and the second subset to map the second subtree. For example, in order to solve the problem illustrated in part A of Fig. 1, algorithm ANQ-Alg divides it into the subproblems illustrated in parts B and C. Algorithm ANQ-Alg solves the subproblem illustrated in part B and uses its answer (i.e., the isomorphism of the hexagons tree in part C) to solve the subproblem illustrated in part C.

2.2. Some Definitions

Each definition given in below is preceded by an explanation of its relevance and is illustrated in Fig. 1.

First choose some node $p_3 \in V(P_1)$, and add $p_1, p_2, \{p_1, p_2\}$ and $\{p_2, p_3\}$ (these are new nodes and edges) to P_1. Also add a new node h^* to H and connect it to all the nodes in $V(H)$ by edges of weight 0. We thus avoid a special treatment of the first call to the recursive procedure ANQ-Rec, that is the main part of our algorithm ANQ-Alg.

Root P_1 at p_1, and use a preorder to denote its nodes by $p_1, p_2, \ldots, p_{|V(P_1)|}$. Given nodes p and $n_p \in N(p)$, let $T(p, n_p)$ denote the subtree induced by p, its children whose indexes are greater than that of n_p, and the descendants of these children. For example, in part A of Fig. 1, we have that $T(p_3, p_2) = R$.

Each stage of ANQ-Rec concerns a subtree of P_1 of the form $T(r, n_r)$, for nodes r and $n_r \in N(r)$, a set $U \subseteq V(H)$, and a set *Solved* of disjoint subtrees of $T(r, n_r)$. The trees in *Solved* are of the form $T(p, n_p)$, and can thus be represented by pairs (p, n_p). Definition 1 concerns this set of trees.

Definition 1. *Given Solved* $\subseteq \{(p,n_p) : p \in V(P_1), n_p \in N(p)\}$, $r \in V(P_1)$ *and* $n_r \in N(r)$, *we say that Solved is an* (r,n_r)-subtree set *if its trees are disjoint subtrees of* $T(r,n_r)$ *and one of them is rooted at* r *(i.e.,* $Solved[(1,r)] \neq \emptyset$*).*

For example, in part A of Fig. 1, we have that $Solved = \{(p_3,p_4), (p_6,p_5), (p_5, p_8)\}$ is a (p_3, p_2)-subtree set.

Each tree $T(p, n_p)$ in *Solved* has several scores. Each score corresponds to its mapping to a subtree T of H. We only know the root of T, and it belongs to U iff $p \neq r$. Moreover, no tree has different scores for isomorphisms that map its root to the same node in $V(H)$. We use a tuple (p, n_p, h, s) to represent an isomorphism of score s between $T(p, n_p)$ and a subtree of H that maps p to h. Definition 2 concerns these tuples. We note that PS stands for Partial Solutions.

Definition 2. *Let* $PS \subseteq \{(p,n_p,h,s) : p \in V(P_1), n_p \in N(p), h \in V(H), s \in \mathbb{R}\}$, $r \in V(P_1), n_r \in N(r)$ *and* $U \subseteq V(H)$. *We say that PS is an* (r,n_r,U)-set *if*

1. $\{(p, n_p) : PS[(1, p), (2, n_p)] \neq \emptyset\}$ *is an* (r, n_r)*-subtree set.*
2. $\forall (p, n_p, h, s) \in PS$*:* $\forall s'[(p, n_p, h, s') \in PS \rightarrow s = s']$ *and* $(p \neq r \leftrightarrow h \in U)$.

For example, in part A of Fig. 1, we have that $PS = \{(p_3, p_4, c, 2), (p_3, p_4, e, 2), (p_6, p_5, i, 2), (p_5, p_8, i, 3), (p_5, p_8, h, 3)\}$ is a (p_3, p_2, U)-set.

Suppose we have an (r, n_r, U)-set PS. We find the best options (corresponding to different mappings of r) to map the roots of the subtrees of $T(r, n_r)$ in PS and the nodes in $T(r, n_r)$ which do not belong to these subtrees to subtrees whose nodes (excluding the mappings of r) are in U. Thus we map all $T(r, n_r)$ and use only nodes in U and nodes that we have already used for computing PS. Definition 3 concerns this set of nodes in $T(r, n_r)$ which we want to map.

Definition 3. *Given an (r, n_r, U)-set PS, $T(PS)$ is the subtree of P_1 induced by $\{v \in V(T(r, n_r)) : \nexists (p, n_p, h, s) \in PS$ s.t. $v \in V(T(p, n_p)) \setminus \{p\}\}$.*

For example, in part A of Fig. 1, we have that $T(PS)$ is the subtree of R induced by its bold nodes.

We divide our problem into two smaller subproblems. We achieve this by finding a node $m \in V(T(PS))$ and a neighbor $n_m \in N(m)$ that divide $T(PS)$ into two smaller subtrees: $P_1[V(T(PS)) \cap V(T(m, n_m))]$ and $P_1[V(T(PS)) \setminus V(T(m, n_m))] \cup \{m\}$. Definition 4 concerns our division options.

Definition 4. *Given an (r, n_r, U)-set PS, we define:*
$mid(PS) = \{(m, n_m, size_L, size_R) : m \in V(T(PS)), n_m \in N(m), size_L = |V(T(PS)) \cap V(T(m, n_m))| - 1, size_R = |V(T(PS))| - size_L - 1\}$.

For example, in part A of Fig. 1, we have that $mid(PS) = \{(p_3, p_2, 4, 0), (p_3, p_4, 0, 4), (p_3, p_{11}, 0, 4), (p_4, p_3, 3, 1), (p_4, p_5, 0, 4), (p_5, p_4, 2, 2), (p_5, p_6, 1, 3), (p_5, p_8, 0, 4), (p_5, p_9, 0, 4, (p_6, p_5, 0, 4), (p_6, p_7, 0, 4), (p_8, p_5, 0, 4)\}$.

We seek a tuple $(m, n_m, size_L, size_R) \in mid(PS)$ that minimizes $\max\{size_L, size_R\}$. Then, as the following lemma implies, our new subproblems are small.

Lemma 1. *Given a rooted tree T such that $v_1, v_2, \ldots, v_n$ is a preorder of $V(T)$ and $n \geq 3$, there are $v_i \in V(T)$ and $v_j \in N(v_i)$ such that $\max\{2, \lceil \frac{n}{3} \rceil\} \leq |V(T(v_i, v_j))| \leq \lfloor \frac{2n}{3} \rfloor$. If T is a path, then there are $v_i \in V(T)$ and $v_j \in N(v_i)$ such that $|V(T(v_i, v_j))| = \lceil \frac{n}{2} \rceil$.*

Proof. Given $v_x, v_y \in V(T)$, denote $V_{x,y} = V(T(v_x, v_y))$. Also denote $U_1 = \{(v_x, v_y) : v_x \in V(T), v_y \in N(v), |V_{x,y}| \leq \lfloor \frac{2n}{3} \rfloor\}$, and $U_2 = \{(v_x, v_y) : v_x \in V(T), v_y \in N(v), \max\{2, \lceil \frac{n}{3} \rceil\} \leq |V_{x,y}|\}$.

Since $n \geq 3$, we have that if $|V_{1,2}| < \max\{2, \lceil \frac{n}{3} \rceil\}$, then $|V_{2,1}| = n - |V_{1,2}| \geq \max\{2, \lceil \frac{n}{3} \rceil\}$; therefore $U_2 \neq \emptyset$. Let (v_i, v_j) be a pair in U_2 that minimizes $|V_{i,j}|$.

Suppose, by way of contradiction, that $(v_i, v_j) \notin U_1$. Since $|V_{i,j}| \geq 2$, we can denote by v_l the child of v_i with the smallest index that is greater than j. Since $|V_{l,i}| < |V_{i,j}|$, our choice of (v_i, v_j) implies that $(v_l, v_i) \notin U_2$. Therefore $|V_{i,l}| = |V_{i,j}| - |V_{l,i}| \geq (\lfloor \frac{2n}{3} \rfloor + 1) - (\max\{2, \lceil \frac{n}{3} \rceil\} - 1) = \lfloor \frac{2n}{3} \rfloor - \max\{2, \lceil \frac{n}{3} \rceil\} + 2 \geq \max\{2, \lceil \frac{n}{3} \rceil\}$. We get that $(v_i, v_l) \in U_2$, but since $|V_{i,l}| < |V_{i,j}|$, this is a contradiction. Thus $(v_i, v_j) \in U_1$, which proves the first part of the lemma.

Now suppose that T is a path. Denote by l the index of the leaf in T that is not v_n. If $l < \lfloor \frac{n}{2} \rfloor + 1$, then $i = \lfloor \frac{n}{2} \rfloor + 1$ and j denotes the index of the father of v_i (which exists since $i > 1$), and thus $|V_{i,j}| = |\{v_i, v_{i+1}, ..., v_n\}| = n - (i - 1) = \lceil \frac{n}{2} \rceil$. Else if $l = \lfloor \frac{n}{2} \rfloor + 1$, then $i = 1$ and $j = 2$, and thus $|V_{i,j}| =$

$|\{v_1\} \cup \{v_{l+1}, v_{l+2}, ..., v_n\}| = 1+n-(\lfloor \frac{n}{2} \rfloor +1) = \lceil \frac{n}{2} \rceil$. Otherwise $i = l - \lceil \frac{n}{2} \rceil + 1$ and $j = i-1$ (note that $j \geq 1$), and thus $|V_{i,j}| = |\{v_i, v_{i+1}, ..., v_l\}| = l-(i-1) = \lceil \frac{n}{2} \rceil$. □

2.3. The Algorithm

We are now ready to present our algorithm ANQ-Alg, whose main component is a recursive procedure called ANQ-Rec. We first note that an input for ANQ-Rec is of the form (r, n_r, U, PS), where $r \in V(P_1)$, $n_r \in N(r)$, $U \subseteq V(H)$, and PS is an empty set or an (r, n_r, U)-set. The output SOL of ANQ-Rec is an empty set or an (r, n_r, U)-set such that $SOL[(1, r), (2, n_r)] = SOL$ (i.e., the tuples in SOL represent mappings of $T(r, n_r)$).

ANQ-Alg($\mathcal{P}, H, \Delta, W$):

1. Add elements to the input as described in Section 2.2.
2. $SOL \Leftarrow$ ANQ-Rec$(p_2, p_1, V(H) \setminus \{h^*\}, \{(p_2, p_3, h^*, 0)\})$.
3. Accept iff $(SOL \neq \emptyset \wedge \max_{(p,n_p,h,s) \in SOL}\{s\} \geq W)$.

Next, we present the pseudocode of ANQ-Rec.

ANQ-Rec(r, n_r, U, PS):

1. If $PS = \emptyset \vee |V(T(PS))| = 1$: Return PS.

We handle two base cases. $PS = \emptyset$ implies that we could not map some subtree of $T(r, n_r)$ in previous computations, and thus we return $\emptyset$.

2. If $|V(T(PS))| = 2$:
   (a) Denote by v the node in $V(T(PS))$ which is not r.
   (b) If $PS[(1, v)] = \emptyset$: Return

$$\bigcup_{\substack{h \in V(H) \\ \text{s.t. } U \cap N(h) \neq \emptyset, \\ s \in \mathbb{R} \\ \text{s.t. } (r,v,h,s) \in PS}} \{(r, n_r, h, \max_{h' \in U \cap N(h)}\{s + w(\{h, h'\}) + \Delta(l(v), l(h'))\})\}.$$

2. (c) Return

$$\bigcup_{\substack{h \in V(H) \\ \text{s.t.} \\ \bigcup_{h' \in N(h)}\{PS[(1,v),(3,h')]\} \neq \emptyset, \\ s \in \mathbb{R} \\ \text{s.t. } (r,v,h,s) \in PS}} \{(r,n_r,h, \max_{\substack{h' \in N(h), \\ s' \in \mathbb{R} \\ \text{s.t.} \\ (v,r,h',s') \in PS}} \{s+s'+w(\{h,h'\})\})\}.$$

We handle the two remaining base cases. They correspond to whether or not v is a root of a tree in PS. In both, for each mapping of r, we find the best legal mapping of v to a node h' in U.

3. $SOL \Leftarrow \emptyset$.

The set SOL will hold tuples that represent the best mappings we find for $T(r, n_r)$.

4. Choose $(m,n_m,size_L,size_R) \in mid(PS)$ that minimizes $\max\{size_L,size_R\}$.

We find the best nodes m and n_m to divide our problem of mapping $T(PS)$ into the two smaller subproblems of mapping $V(T(PS)) \cap V(T(m, n_m))$ and $(V(T(PS)) \setminus V(T(m, n_m))) \cup \{m\}$.

5. $prob_L \Leftarrow \dfrac{size_L}{|V(T(PS))| - 1}$ and $prob_R \Leftarrow 1 - prob_L$.
6. Repeat $\dfrac{1}{(1-1/e)^2 {prob_L}^{size_L} {prob_R}^{size_R}}$ times:
   (a) $U_L \Leftarrow \emptyset$ and $U_R \Leftarrow U$.
   (b) For each $h \in U$: With probability $prob_L$, move h from U_R to U_L.

We randomly partition U into two sets, U_L and U_R, that we use in the first and second subproblems, respectively, as follows. The nodes in $(V(T(PS)) \cap V(T(m, n_m))) \setminus \{m\}$ are mapped to nodes in U_L, and then the other nodes in $V(T(PS)) \setminus \{r\}$ are mapped to nodes in U_R. The probability $prob_L$ of a node to be in U_L and the number of executions of Step 6 guarantee that with good probability the solutions to our subproblems allow solving our problem of mapping $T(PS)$.

6. (c) $SOL_L \Leftarrow \emptyset$ and $SOL_R \Leftarrow \emptyset$.

The sets SOL_L and SOL_R will hold the solutions we find to our subproblems.

6. (d) $PS_L \Leftarrow \{(p, n_p, h, s) \in PS : p \in V(T(m, n_m)), p \neq m \leftrightarrow h \in U_L\}$.
   (e) If $PS[(1, m)] = \emptyset$:
      i. If $U_R = \emptyset$: Go to the next iteration.
      ii. Add $\bigcup_{n_p \in N(m) \text{ s.t. } V(T(m,n_p))=\{m\}, h \in U_R} \{(m,n_p,h,\Delta(l(m),l(h)))\}$ to PS_L.
   (f) If $\exists p \in V(T(m, n_m))[PS[(1, p)] \neq \emptyset \wedge PS_L[(1, p)] = \emptyset]$: Go to the next iteration.

The set PS_L is initialized to hold the tuples in PS that are relevant to mapping $T(m, n_m)$. If it does not include a tree rooted at m, then we add all the options of mapping the tree that contains only m to a node in U_R (if $U_R = \emptyset$, then we skip the rest of the iteration). If we lost the tuples representing all the mappings in PS of a tree that is relevant to PS_L, then we skip the rest of the iteration.

6. (g) $SOL_L \Leftarrow \mathsf{ANQ\text{-}Rec}(m, n_m, U_L, PS_L)$.

We solve our first subproblem.

6. (h) $PS_R \Leftarrow SOL_L \cup \{(p, n_p, h, s) \in PS : p \notin V(T(m, n_m)), h \notin U_L\}$.

(i) If $SOL_L = \emptyset \ \vee \ \exists p \notin V(T(m, n_m))[PS[(1,p)] \neq \emptyset \wedge PS_R[(1,p)] = \emptyset]$: Go to the next iteration.

The set PS_R is initialized to hold SOL_L and the tuples in PS that are relevant to our second subproblem. If $SOL_L = \emptyset$ or we lost the tuples representing all the mappings in PS of a tree that is relevant to our second subproblem, then we skip the rest of the iteration.

6. (j) $SOL_R \Leftarrow \mathsf{ANQ\text{-}Rec}(r, n_r, U_R, PS_R)$.
   (k) For each h, s s.t. $(r,n_r,h,s) \in SOL_R \wedge \nexists s'[(r,n_r,h,s') \in SOL \wedge s \leq s']$: $SOL \Leftarrow (SOL \cup \{(r, n_r, h, s)\}) \setminus SOL[(1, r), (2, n_r), (3, h)]$.
7. Return SOL.

We solve our second subproblem. Then we update SOL to hold the tuples representing the best mappings of $T(r, n_r)$ we have found so far.

2.4. Correctness and Running Time

In this section we prove the following theorem.

Theorem 1. *Algorithm* ANQ-Alg *solves inputs for PINQ in which $\mathcal{P}$ is a set of one tree in time $O(6.75^{k+O(\log k)}|E(H)|)$ and space $O(|V(H)| \log^2 k)$. If P_1 is a path, then it runs in time $O(4^{k+O(\log k)}|E(H)|)$.*

First we prove the correctness of algorithm ANQ-Alg.

Let $r \in V(P_1), n_r \in N(r), U \subseteq V(H)$ and PS be an (r, n_r, U)-set. Given $h \in V(H)$, let $ISO(r, n_r, U, PS)_h$ denote the set of every isomorphism M between $T(PS)$ and a subtree of H, such that $M(r) = h$ and for each $p \in V(T(PS))$ the following condition holds.

1. If $PS[(1,p)] = \emptyset$: $M(p) \in U$. Denote $s(M,p) = \Delta(l(p), l(M(p)))$.
2. Else: $PS[(1,p),(3,M(p))] \neq \emptyset$.
   Denote by $s(M,p)$ the score s for which $PS[(1,p),(3,M(p)),(4,s)] \neq \emptyset$.

Given $M \in ISO(r, n_r, U, PS)_h$, let $s(ISO(r, n_r, U, PS)_h, M)$ denote the score $\sum_{p \in V(T(PS))} s(M,p) + \sum_{\{p,p'\} \in E(T(PS))} w(\{M(p), M(p')\})$. If $ISO(r,n_r,U, PS)_h \neq \emptyset$, then let $s(ISO(r,n_r,U,PS)_h)$ denote the score $\max_{M \in ISO(r,n_r,U,PS)_h}\{ s(ISO(r, n_r, U, PS)_h, M)\}$.

Given an input I, let $(\mathcal{P}, H, \Delta, W)$ denote the instance that we get after adding $p_1, p_2, \{p_1, p_2\}, \{p_2, p_3\}, h^*$ and its edges to I (in Step 1 of ANQ-Alg). Note that I has a solution iff there is an isomorphism M between $T(p_2, p_1)$ and a subtree S of H s.t. $M(p_2) = h^*$ and $\sum_{p \in V(T(p_2,p_1)) \setminus \{p_2\}} \Delta(l(p), l(M(p))) + \sum_{e \in E(S)} w(e) \geq W$. Thus, the following lemma implies the correctness of ANQ-Alg.

Lemma 2. *Given $r \in V(P_1), n_r \in N(r), U \subseteq V(H)$ and $\emptyset$ or an (r, n_r, U)-set PS, the $\emptyset$ or (r, n_r, U)-set SOL returned by* $\mathsf{ANQ\text{-}Rec}(r, n_r, U, PS)$ *satisfies*

1. *If $PS = \emptyset$, then $SOL = \emptyset$.*

2. *Else* $\forall h^* \in V(H)$*:*
   (a) *If* $ISO(r, n_r, U, PS)_{h^*} = \emptyset$*, then* $SOL[(3, h^*)] = \emptyset$.
   (b) *Else:*
      i. *For each* s *s.t.* $(r, n_r, h^*, s) \in SOL$, $s \leq s(ISO(r, n_r, U, PS)_{h^*})$.
      ii. *With probability at least* $1-1/e$, $(r, n_r, h^*, s(ISO(r,n_r, U,PS)_{h^*}))$ $\in SOL$.

PROOF. We prove the lemma by using induction on $l = |V(T(PS))|$ (where $|V(T(\emptyset))| = 0$). If $0 \leq l < 3$, then by Steps 1 and 2 of the pseudocode, the lemma holds.

Now suppose that $l \geq 3$, and assume that the lemma holds $\forall r' \in V(P_1), n'_r \in N(r'), U' \subseteq V(H)$ and $\emptyset$ or an (r', n'_r, U')-set PS' s.t. $|V(T(PS'))| < l$. Let h^* be a node in $V(H)$.

Let (U_L, U_R) be a partition chosen in an iteration of Step 6. If we do not skip the rest of the iteration in Steps 6(e)i or 6(f), then by the induction hypothesis, the set SOL_L computed in Step 6(g) satisfies $[\forall h, s$ s.t. $(m, n_m, h, s) \in SOL_L : (PS[(1, m)] = \emptyset \rightarrow h \in U_R) \wedge (PS[(1, m)] \neq \emptyset \rightarrow PS[(1, m), (3, h)] \neq \emptyset) \wedge ISO(m, n_m, U_L, PS_L)_h \neq \emptyset \wedge s \leq s(ISO(m, n_m, U_L, PS_L)_h)]$. If we do not skip the rest of the iteration in Step 6(i), then $PS[(1, p), (2, n_p)] \neq \emptyset \rightarrow (PS_L \cup PS_R)[(1, p), (2, n_p)] \neq \emptyset$, and $PS_L \cup (PS_R \setminus SOL_L) \subseteq PS$ to which we add $(\bigcup_{n_p \in N(m) \text{ s.t. } V(T(m,n_p))=\{m\}, h \in U_R} \{(m, n_p, h, \Delta(l(m), l(h)))\})$ iff $PS[(1, m)] = \emptyset$. By the induction hypothesis, the set SOL_R computed in Step 6(j) satisfies $[\forall h, s$ s.t. $(r, n_r, h, s) \in SOL_R : PS[(1, r), (3, h)] \neq \emptyset \wedge ISO(r, n_r, U_R, PS_R)_h \neq \emptyset \wedge s \leq s(ISO(r, n_r, U_R, PS_R)_h)]$. Thus, by the pseudocode and the definitions of ISO and $s(ISO(\ldots))$, we get that if $ISO(r, n_r, U, PS)_{h^*} = \emptyset$ and we do not skip the rest of the iteration before Step 6(j), then $SOL_R[(3, h^*)] = \emptyset$, and otherwise $[\forall s$ s.t. $(r, n_r, h^*, s) \in SOL_R$: $s \leq s(ISO(r, n_r, U, PS)_{h^*})]$.

Now suppose that $ISO(r, n_r, U, PS)_{h^*} \neq \emptyset$. Then, there is a mapping $M \in ISO(r, n_r, U, PS)_{h^*}$ s.t. $s(ISO(r, n_r, U, PS)_{h^*}, M) = s(ISO(r, n_r, U, PS)_{h^*})$.

Denote $S_L = \{h \in U : \exists p \in V(T(PS)) \cap V(T(m, n_m))$ s.t. $M(p) = h\} \setminus \{M(m)\}$, and $S_R = \{h \in U : \exists p \in V(T(PS)) \setminus (V(T(m, n_m)) \setminus \{m\})$ s.t. $M(p) = h\} \setminus \{M(r)\}$.

The probability that a partition (U_L, U_R) s.t. $S_L \subseteq U_L$ and $S_R \subseteq U_R$ is chosen in a given iteration of Step 6 is ${prob_L}^{size_L} {prob_R}^{size_R}$. Now consider an iteration in which such a partition is chosen. We do not skip before Step 6(g). By the induction hypothesis, with probability at least $1 - 1/e$, the set SOL_L computed in Step 6(g) includes $(m, n_m, M(m), s(ISO(m, n_m, U_L, PS_L)_{M(m)}))$. Then, we do not skip the rest of the iteration in Step 6(i); and by the induction hypothesis, with probability at least $1 - 1/e$, the set SOL_R computed in Step 6(j) includes $(r, n_r, h^*, s(ISO(r, n_r, U_R, PS_R)_{h^*}))$. Note that $PS[(1, p), (2, n_p)] \neq \emptyset \rightarrow (PS_L \cup PS_R)[(1, p), (2, n_p)] \neq \emptyset$, and $PS_L \cup (PS_R \setminus SOL_L) \subseteq PS$ to which we add $(\bigcup_{n_p \in N(m) \text{ s.t. } V(T(m,n_p))=\{m\}, h \in U_R} \{(m, n_p, h, \Delta(l(m), l(h)))\})$ iff $PS[(1, m)] = \emptyset$. We get that $s(ISO(r, n_r, U_R, PS_R)_{h^*}) = s(ISO(r, n_r, U, PS)_{h^*})$.

Thus, the probability that there is an iteration in which we reach Step 6(j) and the computed set SOL_R includes $(r, n_r, h^*, s(ISO(r, n_r, U, PS)_{h^*}))$ is at least

$$1-(1-(1-1/e)^2 {prob_L}^{size_L} {prob_R}^{size_R})^{1/((1-1/e)^2 {prob_L}^{size_L} {prob_R}^{size_R})} \geq 1-\frac{1}{e}$$

□

Now we analyze the running time of algorithm ANQ-Alg. Assume WLOG that $|V(H)| \leq |E(H)|$. Given an input (r, n_r, U, PS) to ANQ-Rec such that $l = |V(T(PS))|$, let $T(l)$ be the running time of ANQ-Rec(r, n_r, U, PS). The pseudocode and Lemma 1 imply the following recurrence relation for some constants a and b (note that if $l \geq 4$, then $2 \leq \lceil \frac{l}{3} \rceil$):

- If $0 \leq l < 4 : T(l) \leq b|E(H)|$.
- Else, if P_1 is a path

$$\begin{aligned} T(l) &\leq a \cdot (\frac{l-1}{\lceil \frac{l}{2} \rceil - 1})^{\lceil \frac{l}{2} \rceil - 1} (\frac{l-1}{\lfloor \frac{l}{2} \rfloor})^{\lfloor \frac{l}{2} \rfloor} [l|V(H)| + T(\lceil \frac{l}{2} \rceil) + T(\lfloor \frac{l}{2} \rfloor + 1)] \\ &\leq b \cdot 2^l [l|E(H)| + T(\lfloor \frac{l}{2} \rfloor + 1)]. \end{aligned}$$

- Else,

$$\begin{aligned} T(l) &\leq a \cdot \max_{\lceil \frac{l}{3} \rceil \leq l' \leq \lfloor \frac{2l}{3} \rfloor} \{(\frac{l-1}{l'-1})^{l'-1} (\frac{l-1}{l-l'})^{l-l'} [l|V(H)| + T(l') + T(l-l'+1)]\} \\ &\leq b \cdot (\frac{3}{4^{\frac{1}{3}}})^l [l|E(H)| + T(\lfloor \frac{2l}{3} \rfloor + 1)]. \end{aligned}$$

Lemma 3. *If P_1 is a path, then* ANQ-Alg *runs in time $O(4^k k^{O(1)}|E(H)|)$.*

Proof. Algorithm ANQ-Alg runs in time $O(T(k+1))$. We prove that for all $0 \leq l$, $T(l) \leq c4^l(\max\{l, 1\})^c|E(H)|$, where $c \geq b$ is a constant s.t. $8c(\frac{3}{4})^c \leq 1$. We use induction on l. If $0 \leq l < 4$, then the claim clearly holds.

Now suppose that $l \geq 4$, and assume that the claim holds for all $l' < l$. Using the induction hypothesis, we get that

$$\begin{aligned} T(l) &\leq b \cdot 2^l [l|E(H)| + c \cdot 4^{(\lfloor \frac{l}{2} \rfloor + 1)} (\lfloor \frac{l}{2} \rfloor + 1)^c |E(H)|] \\ &\leq c \cdot 2^l l|E(H)| + 4c^2 \cdot 4^l (\frac{l}{2} + 1)^c |E(H)| \\ &\leq 8c^2 \cdot 4^l (\frac{3l}{4})^c |E(H)| \leq c4^l l^c |E(H)|. \end{aligned}$$

□

Lemma 4. *Algorithm* ANQ-Alg *runs in time $O(6.75^k k^{O(1)}|E(H)|)$.*

PROOF. Algorithm ANQ-Alg runs in time $O(T(k+1))$. We prove that for all $0 \leq l$, $T(l) \leq c6.75^l(\max\{l,1\})^c|E(H)|$, where $c \geq b$ is a constant s.t. $13.5c(\frac{11}{12})^c \leq 1$. We use induction on l. If $0 \leq l < 4$, then the claim clearly holds.

Now suppose that $l \geq 4$, and assume that the claim holds for all $l' < l$. Using the induction hypothesis, we get that

$$\begin{aligned} T(l) &\leq b \cdot (\frac{3}{4^{\frac{1}{3}}})^l[l|E(H)| + c \cdot 6.75^{(\lfloor \frac{2l}{3} \rfloor + 1)}(\lfloor \frac{2l}{3} \rfloor + 1)^c|E(H)|] \\ &\leq c \cdot (\frac{3}{4^{\frac{1}{3}}})^l l|E(H)| + 6.75c^2 \cdot 6.75^l(\frac{2l}{3}+1)^c|E(H)| \\ &\leq 13.5c^2 \cdot 6.75^l(\frac{11l}{12})^c|E(H)| \\ &\leq c6.75^l l^c|E(H)|. \end{aligned}$$

□

In order to analyze the space complexity of algorithm ANQ-Alg, we first prove the following lemma.

Lemma 5. *For any call* ANQ-Rec(r, n_r, U, PS) *performed during the execution of* ANQ-Alg*, we have that* $|\{p \in V(P_1) : PS[(1,p)] \neq \emptyset\}| = O(\log k)$.

PROOF. Algorithm ANQ-Alg calls ANQ-Rec with a set PS s.t. $|\{p \in V(P_1) : PS[(1,p)] \neq \emptyset\}| = 1$. Lemma 1 and the pseudocode imply that the recursive depth of ANQ-Rec is bounded by $O(\log k)$. Moreover, by the pseudocode, each call ANQ-Rec(r', n_r', U', PS') executed by a call ANQ-Rec(r, n_r, U, PS) satisfies $|\{p \in V(P_1) : PS'[(1,p)] \neq \emptyset\}| \leq |\{p \in V(P_1) : PS[(1,p)] \neq \emptyset\}| + 1$, and we thus conclude the lemma. □

By the pseudocode and Lemma 5, we get that each recursive call to ANQ-Rec uses $O(|V(H)| \log k)$ space, and the recursive depth of ANQ-Rec is bounded by $O(\log k)$. Thus, we conclude that the space complexity of ANQ-Alg is bounded by $O(|V(H)| \log^2 k)$.

3. An Algorithm for PINQ

We now present an algorithm, that we call PINQ-Alg, which handles inputs for PINQ in which $\mathcal{P}$ is a set of trees (i.e., P_i is a tree for all $1 \leq i \leq t$). In Section 3.1, we give an overview of PINQ-Alg. Section 3.2 includes some definitions required for the pseudocode of PINQ-Alg (given in Section 3.3). Finally, in Section 3.4, we prove the correctness and analyze the running time of PINQ-Alg.

3.1. Overview

We use the randomized divide-and-conquer method [23]: Our algorithm PINQ-Alg randomly divides the problem into two smaller subproblems that it recursively solves, and then combines the answers. Next, by using Fig. 2, we describe and illustrate a recursive stage in more detail.

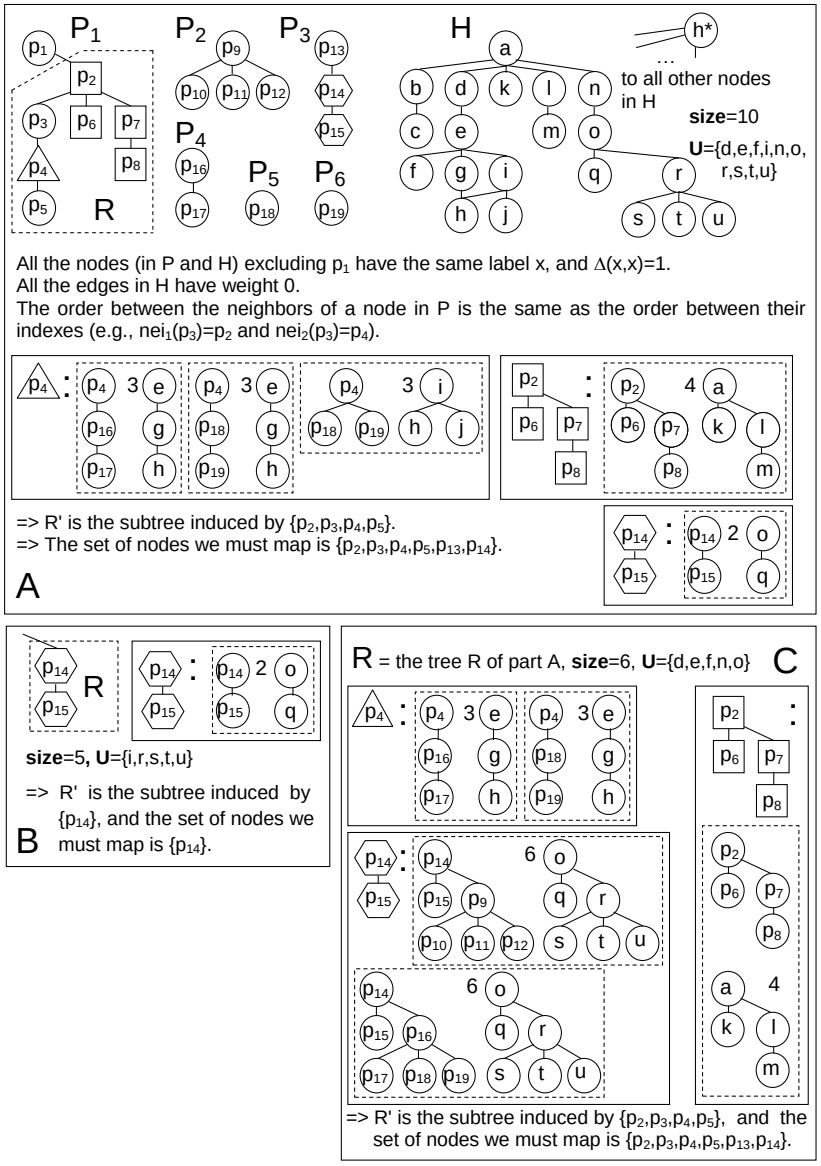

Figure 2: An illustration of a randomized divide-and-conquer step in PINQ-Alg.

Each recursive stage concerns a rooted subtree R of a tree in $\mathcal{P}$, $U \subseteq V(H)$, a set *Solved* of rooted trees and a positive integer *size*. For example, part A of Fig. 2 illustrates an input for algorithm PINQ-Alg and a recursive stage, where *Solved* contains the squares, triangles and hexagons trees. Each tree in *Solved* has several triples (as opposed to the pairs used by ANQ-Alg), where each such triple consists of a set of trees $\hat{\mathcal{P}} \subseteq \mathcal{P}$, a node $h \in V(H)$ and a score s. Such a triple $(\hat{\mathcal{P}}, h, s)$ concerns a subtree S of H, a partition of $V(S)$ into the subsets $\{V_S^1, \ldots, V_S^{|\hat{\mathcal{P}}|+1}\}$, an isomorphism m_1 between the tree in *Solved* (to which the triple belongs) and $S[V_S^1]$ that maps the root of the tree to h, and an isomorphism m_i between $S[V_S^i]$ and a different tree in $\hat{\mathcal{P}}$, for all $2 \leq i \leq |\hat{\mathcal{P}}|+1$, such that $\sum_{1 \leq i \leq |\hat{\mathcal{P}}|+1} \sum_{v \in V_S^i} \Delta(l(v), l(m_i(v))) + \sum_{e \in E(S)} w(e) = s$. Note that the triple $(\hat{\mathcal{P}}, h, s)$ can be considered as a "partial solution". For example, part A of Fig. 2 illustrates the triples $(\{P_4\}, e, 3)$, $(\{P_5, P_6\}, e, 3)$ and $(\{P_5, P_6\}, i, 3)$ that belong to the triangles tree.

Using such triples, algorithm PINQ-Alg maps sets of *size* nodes to subtrees of H, such that each node (excluding the root of R) is mapped to a node in U and neighbors are mapped to neighbors. Each such set of *size* nodes contains the nodes of the subtree R' of R induced by the nodes in R that do not belong to any tree in *Solved* and the roots of the subtrees of R in *Solved* (see Fig. 2 for an example of such a tree R'). Moreover, each such set of *size* nodes must help us "complete" mapping the trees in $\mathcal{P}$ that have subtrees in *Solved*, excluding the tree containing R; thus it also contains their nodes, excluding those that belong to trees in *Solved* and are not their roots. The number of nodes we have just mentioned to be contained in such set a of *size* nodes may be less than *size*; therefore algorithm PINQ-Alg examines several choices of adding nodes of trees in $\mathcal{P}$ that do not have subtrees in *Solved* and thus getting sets of *size* nodes (to be mapped to subtrees of H).

In the base cases of the recursion, $size \leq 2$, and then the problem can be easily solved. In the step of the recursion, algorithm PINQ-Alg divides the problem into two subproblems as follows. Any set of *size* nodes that PINQ-Alg attempts to map may contain nodes of different trees in $\mathcal{P}$, and it does not know in advance how to "connect" them.[3] Thus PINQ-Alg examines several choices of dividing the set of nodes it must map (i.e., the nodes that must be contained in any set of *size* nodes that it attempts to map) into two sets to be separately mapped in the first and second subproblems. Such a division may not imply the number of nodes to be mapped in each subproblem (since the number of nodes that PINQ-Alg must map may be less than *size*), and thus PINQ-Alg also examines several choices of these numbers. Algorithm PINQ-Alg randomly divides U into two subsets. It uses the first subset to solve first subproblem; then it uses the corresponding results and the second subset to solve the second subproblem. For example, in order to solve the problem illustrated in part A of

[3] Algorithm PINQ-Alg needs to "connect" these trees to get one tree to be mapped to a subtree of H.

Fig. 2, algorithm PINQ-Alg divides it into the subproblems illustrated in parts B and C. Algorithm PINQ-Alg solves the subproblem illustrated in part B and uses its answer (i.e., the isomorphism of the hexagons tree in part C) to solve the subproblem illustrated in part C.

3.2. Some Definitions

Each definition given below is preceded by an explanation of its relevance and is illustrated in Fig. 2.

First choose some node $p_2 \in V(P_1)$, and add a new node p_1 and an edge $\{p_1, p_2\}$ of weight 0 to P_1. Define $\forall l \in L : \Delta(l(p_1), l) = -\infty$. Also add a new node h^* to H and connect it to all the nodes in $V(H)$ by edges of weight 0.

Since we may not know a topology for a solution (we only know that it is a tree), we cannot define a preorder on its nodes and use the form $T(p, n_p)$, which is used by ANQ-Alg, for the trees considered at each recursive stage. Thus, we define a different form as follows. We order the neighbors of each node $p \in V(\mathcal{P})$ (arbitrarily), and denote them by $nei_1(p), \ldots, nei_{|N(p)|}(p)$. Denote $\widehat{N}(p) = N(p) \cup \{nil\}$. Also denote $N(p, nei_i(p), nei_j(p)) = \{nei_l(p) \in N(p) : i \leq l < j \vee j < i \leq l \vee l < j < i\}$, $N(p, nei_i(p), nil) = \{nei_l(p) \in N(p) : i \leq l\}$ and $N(p, nil, nil) = \{\}$. Let $P(p)$ denote the tree in $\mathcal{P}$ that contains p, and let $T(p, nei_i(p), nei_j(p))$ denote the subtree induced by the nodes reachable from p in $P(p)[V(P(p)) \setminus (N(p) \setminus N(p, nei_i(p), nei_j(p)))]$. Root $T(p, nei_i(p), nei_j(p))$ at p. For example, in part A of Fig. 2, we have that $T(p_2, p_3, p_1) = R$. Algorithm PINQ-Alg uses the form $T(p, nei_i(p), nei_j(p))$ for the trees it considers at each recursive stage. Definition 5 concerns these trees.

Definition 5. *Given Solved* $\subseteq \{(p, n_p^1, n_p^2) : p \in V(\mathcal{P}), n_p^1 \in \widehat{N}(p), n_p^2 \in \widehat{N}(p)$ *s.t.* $n_p^1 = nil \rightarrow n_p^2 = nil\}$, $r \in V(\mathcal{P}), n_r^1 \in \widehat{N}(r)$ *and* $n_r^2 \in \widehat{N}(r)$ *s.t.* $n_r^1 = nil \rightarrow n_r^2 = nil$, *we say that Solved is an* (r, n_r^1, n_r^2)-subtree set *if its trees are disjoint, those that are subtrees of* $P(r)$ *are also subtrees of* $T(r, n_r^1, n_r^2)$, *and* $Solved[(1, r), (3, n_r^2)] \neq \emptyset$.

For example, in part A of Fig. 2, we have that $Solved = \{(p_2, p_6, p_1), (p_4, p_3, p_3), (p_{14}, p_{15}, p_{13})\}$ is a (p_2, p_3, p_1)-subtree set.

We now define the information of each tree in *Solved* similarly to Definition 2; though now each tree in *Solved* has additional information on a set $\hat{\mathcal{P}}$ of trees in $\mathcal{P}$ that are connected to it, and each of its scores corresponds to a mapping of it *and* the trees in $\hat{\mathcal{P}}$.

Definition 6. *Let* $PS \subseteq \{(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s) : p \in V(\mathcal{P}), n_p^1 \in \widehat{N}(p), n_p^2 \in \widehat{N}(p)$ *s.t.* $n_p^1 = nil \rightarrow n_p^2 = nil, h \in V(H), \hat{\mathcal{P}} \subseteq \mathcal{P}, s \in \mathbb{R}\}$, $r \in V(\mathcal{P}), n_r^1 \in \widehat{N}(r), n_r^2 \in \widehat{N}(r)$ *s.t.* $n_r^1 = nil \rightarrow n_r^2 = nil$, *and* $U \subseteq V(H)$. *We say that PS is an* (r, n_r^1, n_r^2, U)-set *if*

1. $\{(p, n_p^1, n_p^2) : PS[(1, p), (2, n_p^1), (3, n_p^2)] \neq \emptyset\}$ *is an* (r, n_r^1, n_r^2)*-subtree set.*
2. $\forall (p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s) \in PS$*:* $\forall s'[(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s') \in PS \rightarrow s = s']$, *(*$p \neq r \leftrightarrow h \in U$*) and* $\forall \hat{\mathcal{P}}'[PS[(4, \hat{\mathcal{P}}')] \neq \emptyset \rightarrow P(p) \notin \hat{\mathcal{P}}']$.

For example, in part A of Fig. 2, we have that $PS = \{(p_2, p_6, p_1, \{\}, a, 4), (p_4, p_3, p_3, \{P_4\}, e, 3), (p_4, p_3, p_3, \{P_5, P_6\}, e, 3), (p_4, p_3, p_3, \{P_5, P_6\}, i, 3), (p_{14}, p_{15}, p_{13}, \{\}, o, 2)\}$ is a (p_2, p_3, p_1, U)-set.

Next we define the set of nodes that a given (r, n_r^1, n_r^2, U)-set PS implies we must map. This is a modification of Definition 3.

Definition 7. *Given an (r, n_r^1, n_r^2, U)-set PS, $V(PS)$ is the set of nodes* $[V(T(r, n_r^1, n_r^2)) \cup \bigcup_{p \text{ s.t. } PS[(1,p)] \neq \emptyset \wedge P(p) \neq P(r)} V(P(p))] \setminus (\bigcup_{(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s) \in PS} V(T(p, n_p^1, n_p^2)) \setminus \{p\})$.

For example, in part A of Fig. 2, we have that $V(PS) = \{p_2, p_3, p_4, p_5, p_{13}, p_{14}\}$.

We do not use a definition similar to Definition 4 since we may not know a topology for $V(PS)$ in a solution, and thus we cannot determine how each node divides it. We consider every node in $V(PS)$ as a possible divisor of our problem and examine several options for the sizes of the resulting smaller subproblems.

The next definition is used for the sake of clarity of the pseudocode. Given (r, n_r^1, n_r^2, U)-sets PS and PS', we define a calculation that uses PS' to update PS to hold the information of both sets that corresponds to the best scores.

Definition 8. *Given (r, n_r^1, n_r^2, U)-sets PS and PS', $PS \stackrel{+}{\Leftarrow} PS'$ is defined as* $\{(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s) \in PS \cup PS' : \forall s'[(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s') \in PS \cup PS' \rightarrow s' \leq s]\}$.

3.3. The Algorithm

We are now ready to present our algorithm PINQ-Alg, whose main component is a recursive procedure called PINQ-Rec. We first note that an input for PINQ-Rec is of the form $(r, n_r^1, n_r^2, U, size, PS)$, where $r \in V(\mathcal{P})$, $n_r^1 \in \widehat{N}(p)$, $n_r^2 \in \widehat{N}(p)$ s.t. $n_r^1 = nil \rightarrow n_r^2 = nil$, $U \subseteq V(H)$, $size \in \mathbb{N}$ and PS is an empty set or an (r, n_r^1, n_r^2, U)-set such that $|V(PS)| \leq size$. The output SOL of PINQ-Rec is an empty set or an $(r, n_r^1, n_r^2, U, size)$-set such that $SOL[(1, r), (2, n_r^1), (3, n_r^2)] = SOL$.

PINQ-Alg$(\mathcal{P}, H, \Delta, W)$**:**

1. Add elements to the input as described in Section 3.2.
2. $SOL \Leftarrow$ PINQ-Rec$(p_1, p_2, nil, V(H) \setminus \{h^*\}, k+1, \{(p_1, nil, nil, h^*, 0)\})$.
3. Accept iff $(SOL \neq \emptyset \wedge \max_{(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s) \in SOL}\{s\} \geq W)$.

Next, we present the pseudocode of PINQ-Rec.

PINQ-Rec$(r, n_r^1, n_r^2, U, size, PS)$**:**

1. If $PS = \emptyset \vee size = 1$: Return PS.
2. $SOL \Leftarrow \emptyset$.

Step 1 handles two base cases. The set SOL will hold tuples that represent the best mappings we find.

3. If $size = 2$:
   (a) If $|V(PS)| = 2$: Denote by v the node in $V(PS)$ which is not r.
   (b) If $|V(PS)| = 1$, then for each $(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s) \in PS, v \in V(\mathcal{P}) \setminus \{r\}$ s.t. $[P(v) \notin \hat{\mathcal{P}} \ \wedge \ N(v) = \emptyset], h' \in U \cap N(h)$:
      - $SOL \stackrel{+}{\Leftarrow} \{(r, n_r^1, n_r^2, \hat{\mathcal{P}} \cup \{P(v)\}, h, s + w(\{h, h'\}) + \Delta(l(v), l(h')))\}$.
   (c) Else if $PS[(1, v)] = \emptyset$, then for each $(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s) \in PS, h' \in U \cap N(h)$:
      - $SOL \stackrel{+}{\Leftarrow} \{(r, n_r^1, n_r^2, \hat{\mathcal{P}}, h, s + w(\{h, h'\}) + \Delta(l(v), l(h')))\}$.
   (d) Else if $v \in V(P(r))$, then for each $(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s) \in PS[(1, r)], \hat{\mathcal{P}}' \subseteq \mathcal{P} \setminus \hat{\mathcal{P}}, h' \in N(h)$, s' s.t. $\exists n_v[(v, n_v, r, \hat{P}', h', s') \in PS]$:
      - $SOL \stackrel{+}{\Leftarrow} \{(r, n_r^1, n_r^2, \hat{\mathcal{P}} \cup \hat{\mathcal{P}}', h, s + w(\{h, h'\}) + s')\}$.
   (e) Else for each $(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s) \in PS[(1, r)], \hat{\mathcal{P}}' \subseteq \mathcal{P} \setminus \hat{\mathcal{P}}, h' \in N(h)$, s' s.t. $\exists n_v[(v, n_v, nil, \hat{\mathcal{P}}', h', s') \in PS]$:
      - $SOL \stackrel{+}{\Leftarrow} \{(r, n_r^1, n_r^2, \hat{\mathcal{P}} \cup \hat{\mathcal{P}}' \cup \{P(v)\}, h, s + w(\{h, h'\}) + s')\}$.
   (f) Return SOL.

If $size = 2$, then we have four base cases. For each of the two base cases corresponding to whether or not $PS[(1, v)] = \emptyset$, we have two base cases that correspond to whether or not v and r belong to the same tree in $\mathcal{P}$ (note that if they do, then they are neighbors).

4. For each $m \in V(\mathcal{P})$,
   $n_m^1 \in \widehat{N}(m), n_m^2 \in \widehat{N}(m)$ s.t. $(n_m^1 = nil \rightarrow n_m^2 = nil)$,
   $size_L \in \{\max\{1, \lceil \frac{size}{3} \rceil - 1\}, \max\{2, \lceil \frac{size}{3} \rceil\}, \ldots, \lfloor \frac{2size}{3} \rfloor - 1\}$,
   partition (P_L, P_R) of $\{p : PS[(1, p)] \neq \emptyset]\}$ s.t. $m \notin P_R$:

We examine choices for m, n_m^1 and n_m^2 that may divide our problem of mapping sets of $size$ nodes (which are supersets of $V(PS)$) into smaller subproblems.

Since we do not know all the nodes we need to map in each of the resulting subproblems, we examine several choices for the sizes of the subproblems. We only examine choices in which both sizes are at most $\lfloor \frac{2size}{3} \rfloor + 1$ (if $size = 3$, then at most 2) to get the desired running time. This still allows us to find a solution (see Lemma 1 for intuition). For the same reason, we examine all the partitions of the set of roots of trees in PS into two sets, P_L and P_R, to be used in the first and second subproblems, respectively. We require that $m \notin P_R$ since in our second subproblem we will only be interested in mappings of m that were found in solutions to our first subproblem.

4. (a) $size_R \Leftarrow size - size_L - 1, prob_L \Leftarrow \frac{size_L}{size - 1}$ and $prob_R \Leftarrow 1 - prob_L$.
   (b) Repeat $\frac{1}{(1 - 1/e)^2 {prob_L}^{size_L} {prob_R}^{size_R}}$ times:
      i. $U_L \Leftarrow \emptyset$ and $U_R \Leftarrow U, SOL_L \Leftarrow \emptyset$ and $SOL_R \Leftarrow \emptyset$.
      ii. For each $h \in U$: With probability $prob_L$ move h from U_R to U_L.

We randomly partition U into two sets, U_L and U_R, that we use in the first and second subproblems, respectively. The sets SOL_L and SOL_R will hold the solutions we find to our first and second subproblems.

4. (b) iii. $PS_L \Leftarrow \{(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s) \in PS : p \in P_L, p \neq m \leftrightarrow h \in U_L\}$.
   iv. If $m \notin P_L$: $PS_L \stackrel{+}{\Leftarrow} \bigcup_{h \in U_R} \{(m, n_m^2, n_m^2, \emptyset, h, \Delta(l(m), l(h)))\}$.
   v. For each $(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s) \in PS_L$ s.t. $P(m) \in \hat{\mathcal{P}}$:
   $PS_L \Leftarrow PS_L \setminus \{(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s)\}$.
   vi. If $\exists p \in P_L$ s.t. $PS_L[(1,p)] = \emptyset \ \vee$ PS_L is not an (m, n_m^1, n_m^2, U_L) -set $\vee\ |V(PS_L)| > size_L + 1$: Go to the next iteration.

The set PS_L is initialized to hold all the tuples in PS that are relevant to our first subproblem, and if it does not include a tree rooted at m, then we add all the options of mapping the tree that contains only m to a node in U_R. We remove tuples that do not correspond to m and yet map its entire tree from PS_L. If P_L contains a node that is not a root of a tree in PS_L, or PS_L is not an (m, n_m^1, n_m^2, U)-set, or PS_L requires mapping too many nodes, then we skip the rest of the iteration.

4. (b) vii. $SOL_L \Leftarrow \mathsf{PINQ\text{-}Rec}(m, n_m^1, n_m^2, U_L, size_L + 1, PS_L)$.

We solve our first subproblem.

4. (b) viii. $PS_R \Leftarrow SOL_L \cup \{(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s) \in PS : p \in P_R, h \notin U_L\}$.
   ix. For each $(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s) \in PS_R$ s.t. $\exists p' \in P_R \cup \{m\}$ for which $P(p') \in \hat{\mathcal{P}}$: $PS_R \Leftarrow PS_R \setminus \{(p, n_p^1, n_p^2, \hat{\mathcal{P}}, h, s)\}$.
   x. If $\exists p \in P_R \cup \{m\}$ s.t. $PS_R[(1,p)] = \emptyset \ \vee$ PS_R is not an (r, n_r^1, n_r^2, U_R)-set $\vee\ |V(PS_R)| > size_R + 1$: Go to the next iteration.

The set PS_R is initialized to hold the solutions to the first subproblem and the tuples in PS that are relevant to our second subproblem. We remove tuples that do not correspond to a node $p' \in P_R \cup \{m\}$ and yet map its entire tree from PS_R. If the resulting PS_R is illegal (the check is similar to that in Step 4(b)vi), then we skip the rest of the iteration.

4. (b) xi. $SOL \stackrel{+}{\Leftarrow} \mathsf{PINQ\text{-}Rec}(r, n_r^1, n_r^2, U_R, size_R + 1, PS_R)$.
5. Return SOL.

We solve our second subproblem and update SOL.

3.4. Correctness and Running Time

In this section we prove the following theorem.

Theorem 2. *Algorithm* PINQ-Alg *solves inputs for PINQ in which $\mathcal{P}$ is a set of trees in time $O(6.75^{k+O(\log^2 k)} 3^t |E(H)|)$ and space $O(2^t |V(H)| \log^2 k)$.*

First we prove the correctness of algorithm PINQ-Alg.

Let $r \in V(\mathcal{P}), n_r^1 \in \widehat{N}(p), n_r^2 \in \widehat{N}(p)$ s.t. $n_r^1 = nil \rightarrow n_r^2 = nil, U \subseteq V(H), size \in \mathbb{N}$, PS be an (r, n_r^1, n_r^2, U)-set s.t. $|V(PS)| \leq size$, $h \in V(H)$ and $\hat{\mathcal{P}} \subseteq \mathcal{P}$.

Given $\mathcal{A} \subseteq \hat{\mathcal{P}}$, denote $V(PS, \mathcal{A}) = V(PS) \cup (\bigcup_{P_i \in \mathcal{A}} V(P_i))$.

If $P(r) \in \hat{\mathcal{P}}$ or $[\exists p \in V(PS)$ s.t. $P(p) \notin \hat{\mathcal{P}} \cup \{P(r)\}]$, then denote $ISO(r, n_r^1, n_r^2, U, size, PS)_{h,\hat{\mathcal{P}}} = \emptyset$.

Else let $ISO(r, n_r^1, n_r^2, U, size, PS)_{h,\hat{\mathcal{P}}}$ denote the set of every tuple $M = (\mathcal{A}, M_V, M_P)$, where $\mathcal{A} \subseteq \hat{\mathcal{P}} \setminus \bigcup_{p \in V(PS)} P(p)$ s.t. $|V(PS, \mathcal{A})| = size$, M_V is a mapping between a tree on $V(PS, \mathcal{A})$ and a subtree S_M of H s.t. $M_V(r) = h$, $M_P : V(PS, \mathcal{A}) \rightarrow 2^{\hat{\mathcal{P}}}$ s.t. $[(\forall p, p' \in V(PS) : M_P(p) \cap M_P(p') = \emptyset) \wedge \bigcup_{p \in V(PS)} M_P(p) = \hat{\mathcal{P}} \setminus \bigcup_{p \in V(PS, \mathcal{A})} P(p)]$, and for each $p \in V(PS, \mathcal{A})$ the following conditions hold.

1. For all $n_p \in N(p) \cap V(PS, \mathcal{A})$: $\{M_V(p), M_V(n_p)\} \in E(S_M)$.
2. If $PS[(1, p)] = \emptyset$: $M_V(p) \in U$ and $M_P(p) = \emptyset$.
   Denote $s(M, p) = \Delta(l(p), l(M_V(p)))$.
3. Else: There is an element n_p^2 s.t. $PS[(1,p), (3,n_p^2), (4,M_P(p)), (5,M_V(p))] \neq \emptyset$ and [if $p \neq r$, then $(f \notin V(P(p)) \wedge n_p^2 = nil) \vee f = n_p^2$, where f is the node for which $M_V(f)$ is the father of $M_V(p)$ in S_M when rooted at h].
   Let $s(M, p)$ denote the score s for which $PS[(1, p), (4, M_P(p)), (5, M_V(p)), (6, s)] \neq \emptyset$.

Given $M \in ISO(r, n_r^1, n_r^2, U, size, PS)_{h,\hat{\mathcal{P}}}$, let $s(ISO(r, n_r^1, n_r^2, U, size, PS)_{h,\hat{\mathcal{P}}}, M)$ denote the score $\sum_{p \in V(PS, \mathcal{A})} s(M, p) + \sum_{e \in E(S_M)} w(e)$.

If $ISO(r, n_r^1, n_r^2, U, size, PS)_{h,\hat{\mathcal{P}}} \neq \emptyset$, then we denote $s(ISO(r, n_r^1, n_r^2, U, size, PS)_{h,\hat{\mathcal{P}}}) = \max_{M \in (r, n_r^1, n_r^2, U, size, PS)_{h,\hat{\mathcal{P}}}} \{s(ISO(r, n_r^1, n_r^2, U, size, PS)_{h,\hat{\mathcal{P}}}, M)\}$.

Given an input I, let $(\mathcal{P}, H, \Delta, W)$ the instance we get after we adding $p_1, \{p_1, p_2\}, h^*$ and its edges to I (in Step 1 of PINQ-Alg). Note that I has a solution iff there is a subtree S of H whose nodes can be partitioned into the subsets $\{V_1^S, \ldots, V_t^S\}$, such that there is an isomorphism M_1 between $T(p_1, p_2, nil)$ and $S[V_1^S]$ for which $M(p_1) = h^*$, there is an isomorphism M_i between P_i and $S[V_i^S]$, for all $2 \leq i \leq t$, and $\sum_{1 \leq i \leq t} \sum_{v \in V(P_i) \setminus \{p_1\}} \Delta(l(v), l(M_i(v))) + \sum_{e \in E(S)} w(e) \geq W$. Thus, the following lemma implies the correctness of PINQ-Alg.

Lemma 6. *Let $r \in V(\mathcal{P}), n_r^1 \in \widehat{N}(p), n_r^2 \in \widehat{N}(p)$ s.t. $n_r^1 = nil \rightarrow n_r^2 = nil, U \subseteq V(H), size \in \mathbb{N}$ and $\emptyset$ or an (r, n_r^1, n_r^2, U)-set PS s.t. $|V(PS)| \leq size$. The empty set or (r, n_r^1, n_r^2, U)-set SOL returned by* PINQ-Rec*$(r, n_r^1, n_r^2, U, size, PS)$ satisfies*

1. *If $PS = \emptyset$, then $SOL = \emptyset$.*
2. *Else $\forall h^* \in V(H), \hat{\mathcal{P}}^* \subseteq \mathcal{P}$:*
   (a) *If $ISO(r, n_r^1, n_r^2, U, size, PS)_{h^*,\hat{\mathcal{P}}^*} = \emptyset$, then $SOL[(4,\hat{\mathcal{P}}^*), (5,h^*)] = \emptyset$.*
   (b) *Else:*
      i. *For each score s s.t. $(r, n_r^1, n_r^2, \hat{\mathcal{P}}^*, h^*, s) \in SOL$:*
         $s \leq s(ISO(r, n_r^1, n_r^2, U, size, PS)_{h^*,\hat{\mathcal{P}}^*})$.

ii. *With probability at least* $1 - 1/e$*:*
$(r, n_r^1, n_r^2, \hat{\mathcal{P}}^*, h^*, s(ISO(r, n_r^1, n_r^2, U, size, PS)_{h^*, \hat{\mathcal{P}}^*})) \in SOL$.

PROOF. By Step 1 of the pseudocode, the lemma holds for $PS = \emptyset$. We next assume that $PS \neq \emptyset$, and prove the lemma by using induction on $size$. If $1 \leq size < 3$, then by Steps 1 and 3 of the pseudocode, the lemma holds.

Now suppose that $size \geq 3$, and assume that the lemma holds $\forall r' \in V(\mathcal{P}), n_r^{1\prime} \in \widehat{N}(p), n_r^{2\prime} \in \widehat{N}(p)$ s.t. $n_r^{1\prime} = nil \rightarrow n_r^{2\prime} = nil, U' \subseteq V(H), size' < size$ and $\emptyset$ or an $(r', n_r^{1\prime}, n_r^{2\prime}, U')$-set PS' s.t. $|V(PS')| \leq size'$. Let h^* be a node in $V(H)$ and $\hat{\mathcal{P}}^*$ be a subset of $\mathcal{P}$.

Consider an iteration of Step 4, and denote the elements to which it corresponds by $m, n_m^1, n_m^2, size_L$ and (P_L, P_R). Let (U_L, U_R) be a partition chosen next in an iteration of Step 4(b). If we do not skip the rest of the iteration in Step 4(b)vi, then by the induction hypothesis, the next computed set SOL_L satisfies $[\forall \hat{\mathcal{P}}, h, s$ s.t. $(m, n_m^1, n_m^2, \hat{\mathcal{P}}, h, s) \in SOL_L : (PS[(1,m)] = \emptyset \rightarrow h \in U_R) \wedge (PS[(1,m)] \neq \emptyset \rightarrow PS[(1,m),(5,h)] \neq \emptyset) \wedge ISO(m, n_m^1, n_m^2, U_L, size_L + 1, PS_L)_{h,\hat{\mathcal{P}}} \neq \emptyset \wedge s \leq s(ISO(m, n_m^1, n_m^2, U_L, size_L + 1, PS_L)_{h,\hat{\mathcal{P}}})]$. If we do not skip the rest of the iteration in Step 4(b)x, then $PS[(1,p),(2,n_p^1),(3,n_p^2)] \neq \emptyset \rightarrow (PS_L \cup PS_R)[(1,p),(2,n_p^1),(3,n_p^2)] \neq \emptyset$, and $PS_L \cup (PS_R \setminus SOL_L) \subseteq PS$ to which we add $(\bigcup_{h \in U_R}\{(m, n_m^2, n_m^2, \emptyset, h, \Delta(l(m), l(h)))\})$ iff $PS[(1,m)] = \emptyset$. By the induction hypothesis, the set returned by PINQ-Rec in Step 4(b)xi, which we next denote by SOL_R, satisfies $[\forall \hat{\mathcal{P}}, h, s$ s.t. $(r, n_r^1, n_r^2, \hat{\mathcal{P}}, h, s) \in SOL_R : PS[(1,r),(5,h)] \neq \emptyset \wedge ISO(r, n_r^1, n_r^2, U_R, size_R + 1, PS_R)_{h,\hat{\mathcal{P}}} \neq \emptyset \wedge s \leq s(ISO(r, n_r^1, n_r^2, U_R, size_R + 1, PS_R)_{h,\hat{\mathcal{P}}})]$. Thus, by the pseudocode and the definitions of ISO and $s(ISO(\ldots))$, we get that if $ISO(r, n_r^1, n_r^2, U, size, PS)_{h^*, \hat{\mathcal{P}}^*} = \emptyset$ and we do not skip the rest of the iteration before Step 4(b)xi, then $SOL_R[(4,\hat{\mathcal{P}}^*),(5,h^*)] = \emptyset$, and otherwise $[\forall s$ s.t. $(r, n_r^1, n_r^2, \hat{\mathcal{P}}^*, h^*, s) \in SOL_R$: $s \leq s(ISO(r, n_r^1, n_r^2, U, size, PS)_{h^*, \hat{\mathcal{P}}^*})]$.

Now suppose that $ISO(r, n_r^1, n_r^2, U, size, PS)_{h^*, \hat{\mathcal{P}}^*} \neq \emptyset$. Then, there is a tuple $M = (\mathcal{A}, M_V, M_P) \in ISO(r, n_r^1, n_r^2, U, size, PS)_{h^*, \hat{\mathcal{P}}^*}$ s.t. $s(ISO(r, n_r^1, n_r^2, U, size, PS)_{h^*, \hat{\mathcal{P}}^*}, M) = s(ISO(r, n_r^1, n_r^2, U, size, PS)_{h^*, \hat{\mathcal{P}}^*})$.

Let T^* denote the tree on $V(PS, \mathcal{A})$ rooted at r whose edge set is $\{\{p, n_p\} : p \in V(PS, \mathcal{A}), n_p \in V(PS, \mathcal{A}), \{M_V(p), M_V(n_p)\} \in E(S_M)\}$. Order the nodes of T^* by using a preorder as $v_1, v_2, \ldots, v_{size}$, where for each $p \in V(T^*)$, we first visit its children which do not belong to $P(p)$ (in an arbitrary order), and then the children which belong to $P(p)$ in the following order:

1. Let $nei_i(p)$ and $nei_j(p)$ be neighbors of p (in $P(p)$) that are its children in T^*, such that $i < j$ (if p has less than two such children, then there is no order that we need to define).
2. If $p = r \wedge n_r^2 \neq nil$, then let l be the index for which $nei_l(r) = n_r^2$ and visit $nei_i(p)$ before $nei_j(p)$ iff $l < i \vee j < l$.
3. Else if $(p = r \wedge n_r^2 = nil)$ or the father of p in T^* is not in $P(p)$, then visit $nei_i(p)$ before $nei_j(p)$.

4. Else let l be the index for which $nei_l(p)$ is the father of p. Visit $nei_i(p)$ before $nei_j(p)$ iff $l < i \vee j < l$.

By Lemma 1, there are neighbors v_i and v_j in T^* such that $\max\{2, \lceil \frac{n}{3} \rceil\} \leq |V(T(v_i, v_j))| \leq \lfloor \frac{2n}{3} \rfloor$. Denote $\hat{\mathcal{P}}_L = \{P' \in \hat{\mathcal{P}}^* : \exists v \in V(T(v_i, v_j)) \text{ s.t. } P' \in M_P(v) \cup \{P(v)\} \setminus \{P(v_i)\}$, $m = v_i$ and

1. If $m = r$, then denote $n_m^2 = n_r^2$.
2. Else if the father of m in T^* belongs to $P(m)$, then denote it by n_m^2.
3. Else denote $n_m^2 = nil$.
4. Let n_m^1 denote the node in $(V(T(v_i, v_j)) \cap N(m)) \cup \{n_m^2\}$ s.t. $N(m, n_m^1, n_m^2) = N(m) \cap V(T(v_i, v_j))$ (the preorder we have used implies that it exists).

Denote $size_L = |V(T(v_i, v_j))| - 1$, $P_L = \{p \in V(T(v_i, v_j)) : PS[(1, p)] \neq \emptyset\}$ and $P_R = \{p \notin V(T(v_i, v_j)) : PS[(1, p)] \neq \emptyset\}$. Note that there is an iteration of Step 4 in which we iterate over these elements. Next consider this iteration.

Denote $S_L = \{h \in U : \exists p \in V(T(v_i, v_j)) \text{ s.t. } M_V(p) = h\} \setminus \{M_V(m)\}$ and $S_R = \{h \in U : \exists p \in V(PS) \setminus (V(T(v_i, v_j)) \setminus \{m\}) \text{ s.t. } M_V(p) = h\} \setminus \{M_V(r)\}$.

The probability of choosing a partition (U_L, U_R) s.t. $S_L \subseteq U_L$ and $S_R \subseteq U_R$ in a given iteration of Step 4(b) is ${prob_L}^{size_L} {prob_R}^{size_R}$. Consider an iteration in which such a partition is chosen. We do not skip the rest of the iteration in Step 4(b)vi. By the induction hypothesis, with probability at least $1 - 1/e$, the set SOL_L computed in Step 4(b)vii includes $(m, n_m^1, n_m^2, \hat{\mathcal{P}}_L, M_V(m), s(ISO(r, n_r^1, n_r^2, U_L, size_L + 1, PS_L)_{M_V(m), \hat{\mathcal{P}}_L}))$. Then we do not skip the rest of the iteration in Step 4(b)x, and by the induction hypothesis, with probability at least $1 - 1/e$, the set returned by PINQ-Rec in Step 4(b)xi includes $(r, n_r^1, n_r^2, \hat{\mathcal{P}}^*, h^*, s(ISO(r, n_r^1, n_r^2, U_R, size_R + 1, PS_R)_{h^*, \hat{\mathcal{P}}^*}))$. We have that $PS[(1, p), (2, n_p^1), (3, n_p^2)] \neq \emptyset \rightarrow (PS_L \cup PS_R)[(1, p), (2, n_p^1), (3, n_p^2)] \neq \emptyset$, and $PS_L \cup (PS_R \setminus SOL_L) \subseteq PS$ to which we add the set $(\bigcup_{h \in U_R} \{(m, n_m^2, n_m^2, \emptyset, h, \Delta(l(m), l(h)))\})$ iff $PS[(1, m)] = \emptyset$. These arguments imply that $s(ISO(r, n_r^1, n_r^2, U_R, size_R + 1, PS_R)_{h^*, \hat{\mathcal{P}}^*}) = s(ISO(r, n_r^1, n_r^2, U, size, PS)_{h^*, \hat{\mathcal{P}}^*})$.

We get that the probability that there is an iteration in which we reach Step 4(b)xi and then the set returned by PINQ-Rec includes $(r, n_r^1, n_r^2, \hat{\mathcal{P}}^*, h^*, s(ISO(r, n_r^1, n_r^2, U, size, PS)_{h^*, \hat{\mathcal{P}}^*}))$ is at least

$$1 - (1 - (1 - 1/e)^2 {prob_L}^{size_L} {prob_R}^{size_R})^{1/((1-1/e)^2 {prob_L}^{size_L} {prob_R}^{size_R})} \geq 1 - \frac{1}{e}$$

□

Now we analyze the running time of algorithm PINQ-Alg. Assume WLOG that $|V(H)| \leq |E(H)|$. We start by proving the following lemma.

Lemma 7. *For any call* PINQ-Rec$(r, n_r^1, n_r^2, U, size, PS)$ *performed during the execution of* PINQ-Alg*, we have that* $|\{p \in V(\mathcal{P}) : PS[(1, p)] \neq \emptyset\}| = O(\log k)$.

Proof. Algorithm PINQ-Alg calls PINQ-Rec with a set PS s.t. $|\{p \in V(\mathcal{P}) : PS[(1, p)] \neq \emptyset\}| = 1$. Lemma 1 and the pseudocode imply that the recursive

depth of PINQ-Rec is bounded by $O(\log k)$. Moreover, by the pseudocode, each call PINQ-Rec$(r', n_r^{1'}, n_r^{2'}, U', size', PS')$ executed by a call PINQ-Rec$(r, n_r^1, n_r^2, U, size, PS)$ satisfies $|\{p \in V(\mathcal{P}) : PS'[(1,p)] \neq \emptyset\}| \leq |\{p \in V(\mathcal{P}) : PS[(1,p)] \neq \emptyset\}| + 1$, and we thus conclude the lemma. □

Given an input $(r, n_r^1, n_r^2, U, size, PS)$ to PINQ-Rec such that if $PS = \emptyset$ then $l = 0$ and else $l = size$, let $T(l)$ be the running time of PINQ-Rec$(r, n_r^1, n_r^2, U, size, PS)$. The pseudocode and Lemma 7 imply the following recurrence relation for some constants a and b (note that if $l \geq 4$, then $2 \leq \lceil \frac{l}{3} \rceil$):

- If $0 \leq l < 4 : T(l) \leq b \cdot 3^t |E(H)|$.
- Else,

$$\begin{aligned} T(l) &\leq a \sum_{m \in V(P)} \sum_{n_m^1 \in \widehat{N}(m)} \sum_{n_m^2 \in \widehat{N}(m)} \sum_{\lceil \frac{l}{3} \rceil \leq l' \leq \lfloor \frac{2l}{3} \rfloor} [\\ &\quad (\frac{l-1}{l'-1})^{l'-1} (\frac{l-1}{l-l'})^{l-l'} 2^{a \log k} (l 2^t |V(H)| + T(l') + T(l - l' + 1))] \\ &\leq b \cdot k^b (\frac{3}{4^{\frac{1}{3}}})^l [l 3^t |E(H)| + T(\lfloor \frac{2l}{3} \rfloor + 1)]. \end{aligned}$$

Lemma 8. *Algorithm* PINQ-Alg *runs in time* $O(6.75^k k^{O(\log k)} 3^t |E(H)|)$.

Proof. Algorithm PINQ-Alg runs in time $O(T(k+1))$. We prove that for all $0 \leq l$, $T(l) \leq c 6.75^l k^{c \log l} 3^t |E(H)|$, where c is a constant s.t. $1 + b \leq c \log(\frac{23}{22}) \wedge 13.5 c k^{c \log(\frac{23l}{24})} \leq k^{c \log l}$. We use induction on l. If $0 \leq l < 4$, then the claim clearly holds.

Now suppose that $4 \leq l \leq k+1$, and assume that the claim holds for all $l' < l$. Using the induction hypothesis, we get that

$$\begin{aligned} T(l) &\leq b \cdot k^b (\frac{3}{4^{\frac{1}{3}}})^l [l 3^t |E(H)| + c \cdot 6.75^{\lfloor \frac{2l}{3} \rfloor + 1} k^{c \log(\lfloor \frac{2l}{3} \rfloor + 1)} 3^t |E(H|)] \\ &\leq c \cdot k^{b+1} (\frac{3}{4^{\frac{1}{3}}})^l 3^t |E(H)| + 6.75 c^2 \cdot 6.75^l k^{b + c \log(\frac{2l}{3} + 1)} 3^t |E(H)| \\ &\leq c \cdot k^c (\frac{3}{4^{\frac{1}{3}}})^l 3^t |E(H)| + 6.75 c^2 \cdot 6.75^l k^{c \log(\frac{23}{22}) + c \log(\frac{11l}{12})} 3^t |E(H)| \\ &\leq 13.5 c^2 \cdot 6.75^l k^{c \log(\frac{23l}{24})} 3^t |E(H)| \leq c 6.75^l k^{c \log l} 3^t |E(H)|. \end{aligned}$$

□

In terms of space complexity, by the pseudocode and Lemma 7, we get that each recursive call to PINQ-Rec uses $O(2^t |V(H)| \log k)$ space, and the recursive depth of PINQ-Rec is bounded by $O(\log k)$. Thus, we conclude that the space complexity of PINQ-Alg is bounded by $O(2^t |V(H)| \log^2 k)$.

References

[1] R. Y. Pinter, M. Zehavi, Partial information network queries, in: Proc. IWOCA, 2013, pp. 362–375.

[2] V. Fionda, L. Palopoli, Biological network querying techniques: Analysis and comparison, J. Comput. Biol. 18 (2011) 595–625.

[3] F. Sikora, An (almost complete) state of the art around the graph motif problem, Université Paris-Est Technical reports (2012).

[4] D. W. Mount, Bioinformatics: Sequence and Genome Analysis, Cold Spring Harbor Laboratory Press: Cold Spring Harbor, New York, 2004.

[5] R. Niedermeier, Invitation to fixed-parameter algorithms, Oxford University Press, 2006.

[6] R. G. Downey, M. R. Fellows, Fixed-parameter tractability and completeness II: on completeness for W[1], Theor. Comput. Sci. 141 (1995) 109–131.

[7] M. R. Garey, D. S. Johnson, Computers and intractability: a guide to the theory of NP-completeness, W.H. Freeman, New York, 1979.

[8] R. Y. Pinter, O. Rokhlenko, D. Tsur, M. Ziv-Ukelson, Approximate labelled subtree homeomorphism, J. Discrete Algorithms 6 (2008) 480–496.

[9] R. Y. Pinter, O. Rokhlenko, E. Yeger-Lotem, M. Ziv-Ukelson, Alignment of metabolic pathways, Bioinformatics 21 (2005) 3401–3408.

[10] A. Mano, T. Tuller, O. Béjà, R. Y. Pinter, Comparative classification of species and the study of pathway evolution based on the alignment of metabolic pathways, BMC Bioinform. 11 (2010) S38.

[11] R. Y. Pinter, M. Zehavi, Algorithms for topology-free and alignment queries, Technion Technical Reports CS-2012-12 (2012).

[12] N. Alon, R. Yuster, U. Zwick, Color coding, J. Assoc. Comput. Mach. 42 (1995) 844–856.

[13] T. Shlomi, D. Segal, E. Ruppin, R. Sharan, QPath: a method for querying pathways in a protein-protein interaction networks, BMC Bioinform. 7 (2006) 199.

[14] B. Dost, T. Shlomi, N. Gupta, E. Ruppin, V. Bafna, R. Sharan, QNet: a tool for querying protein interaction networks, J. Comput. Biol. 15 (2008) 913–925.

[15] G. Blin, F. Sikora, S. Vialette, Querying graphs in protein-protein interactions networks using feedback vertex set, IEEE/ACM Trans. Comput. Biol. Bioinform. 7 (2010) 628–635.

[16] F. Hüffner, S. Wernicke, T. Zichner, Algorithm engineering for color-coding with applications to signaling pathway detection, Algorithmica 52 (2008) 114–132.

[17] H. Wang, T. Xiang, X. Hu, Research on pattern matching with wildcards and length constraints: methods and completeness, www.intechopen.com/books/ bioinformatics (2012).

[18] V. Lacroix, C. G. Fernandes, M. F. Sagot, Motif search in graphs: Application to metabolic networks, IEEE/ACM Trans. Comput. Biol. Bioinform. 3 (2006) 360–368.

[19] S. Bruckner, F. Hüffner, R. M. Karp, R. Shamir, R. Sharan, Topology-free querying of protein interaction networks, in: Proc. RECOMB, 2009, pp. 74–89.

[20] S. Guillemot, F. Sikora, Finding and counting vertex-colored subtrees, in: Proc. MFCS, 2010, pp. 405–416.

[21] I. Koutis, Faster algebraic algorithms for path and packing problems, in: Proc. ICALP, 2008, pp. 575–586.

[22] A. Björklund, T. Husfeldt, P. Kaski, M. Koivisto, Narrow sieves for parameterized paths and packings, CoRR abs/1007.1161 (2010).

[23] J. Chen, J. Kneis, S. Lu, D. Molle, S. Richter, P. Rossmanith, S. Sze, F. Zhang, Randomized divide-and-conquer: Improved path, matching, and packing algorithms, SIAM J. on Computing 38 (2009) 2526–2547.

[24] A. M. Ambalath, R. Balasundaram, R. H. Chintan, K. Venkata, M. Neeldhara, P. Geevarghese, M. S. Ramanujan, On the kernelization complexity of colorful motifs, in: Proc. IPEC, 2010, pp. 14–25.

[25] N. Betzler, R. Bevern, M. R. Fellows, C. Komusiewicz, R. Niedermeier, Parameterized algorithmics for finding connected motifs in biological networks, IEEE/ACM Trans. Comput. Biol. Bioinform. 8 (2011) 1296–1308.

[26] A. Björklund, P. Kaski, L. Kowalik, Probably optimal graph motifs, in: Proc. STACS, 2013, pp. 20–31.

[27] R. Rizzi, F. Sikora, Some results on more flexible versions of graph motif, in: Proc. CSR, 2012, pp. 278–289.

[28] M. Zehavi, Parameterized algorithms for module motif, in: Proc. MFCS, 2013, pp. 825–836.

[29] R. Dondi, G. Fertin, S. Vialette, Weak pattern matching in colored graphs: minimizing the number of connected components, in: Proc. ICTCS, 2007, pp. 27–38.

[30] R. Dondi, G. Fertin, S. Vialette, Finding approximate and constrained motifs in graphs, in: Proc. CPM, 2009, pp. 221–235.

[31] M. R. Fellows, G. Fertin, D. Hermelin, S. Vialette, Upper and lower bounds for finding connected motifs in vertex-colored graphs, J. Comput. Syst. Sci. 77 (2011) 799–811.

[32] I. Koutis, Constrained multilinear detection for faster functional motif discovery, Inf. Process. Lett. 112 (2012) 889–892.

Figure Captions

Figure 1

An illustration of a randomized divide-and-conquer step in ANQ-Alg.

Figure 2

An illustration of a randomized divide-and-conquer step in PINQ-Alg.