The complexity of edge set partitioning problems

Full Version

Jean-François Couturier* David Eppstein† Henning Fernau‡ Joachim Kneis§
Dieter Kratsch* Mathieu Liedloff¶ Daniel Meister‡ Peter Rossmanith§
Somnath Sikdar§

April 21, 2011

Abstract

We study the complexity of three problems (abbreviated as TTP, PTP, and PPP, respectively) that ask to partition the edge set of a given undirected graph G into two sets A, B such that $\{G[A], G[B]\}$ are (a) two trees, (2) one path and one tree, and (3) two paths, respectively. After stating NP-completeness of these problems, we derive exact algorithms that are based, somewhat surprisingly, on quite different algorithmic techniques.

1 Introduction

We are interested in a class of edge set partition problems. Let P and Q be graph properties.

Problem (P, Q) Edge Set Partition
Given: an undirected graph G
Question: Is there a partition (A, B) of $E(G)$ such that
$G[A]$ has property P and $G[B]$ has property Q?

We investigate the complexity of the problem for properties P and Q that describe classes of trees. Hence, both A and B form feedback edge sets, i.e., edge sets whose removal destroys all cycles. More specifically, we consider the following three problems:
— Tree-Tree Edge Set Partition, TTP for short, asks to partition $E(G)$ into two sets A and B forming trees $G[A]$ and $G[B]$;

*Laboratoire d'Informatique Théorique et Appliquée, Université Paul Verlaine, Metz, France. Emails: couturier@univ-metz.fr, kratsch@univ-metz.fr

†Computer Science, University of California, Irvine, USA. Email: eppstein@uci.edu

‡Abteilung Informatik / Wirtschaftsinformatik, Universität Trier, Germany. Emails: fernau@uni-trier.de, daniel.meister@uni-trier.de

§Theoretical Computer Science, RWTH Aachen University, Germany. Emails: kneis@cs.rwth-aachen.de, rossmani@cs.rwth-aachen.de, sikdar@cs.rwth-aachen.de

¶Laboratoire d'Informatique Fondamentale d'Orléans, Université d'Orléans, France. Email: mathieu.liedloff@univ-orleans.fr

— Path-Tree Edge Set Partition, PTP for short, asks to partition $E(G)$ into two sets A and B forming a path $G[A]$ and a tree $G[B]$; and
— Path-Path Edge Set Partition, PPP for short, asks to partition $E(G)$ into two sets A and B forming two paths $G[A]$ and $G[B]$.

The problem can be motivated from different directions. Application-wise, the general problem of partitioning the edge set of a graph into k edge-disjoint trees relates to problems in network reliability and graph drawing [2]. Theory-wise, the problem is closely related to the graph parameter *arboricity*, that asks for the smallest size of a partition of the edges set into forests. Gabow and Westermann showed that a partition of the edge set of a given graph into at most k forests can be computed in polynomial time [6]. The difference between arboricity and our problem is the lack of connectivity. Biedl and Brandenburg [2] showed that the problem of partitioning the edge set of a graph into at most k trees is NP-hard for all $k \geq 2$. An improvement of this NP-hardness result is due to Pálvölgyi [9], who showed that the partition of the edge set of a graph into two trees is NP-hard on graphs of maximum degree 4. This hardness result naturally extends to the partition problem into at most k trees. So much about intractability results. On the positive side, the question of whether the edge set of a given graph can be partitioned into two spanning trees is polynomial-time solvable due to Edmonds [4].

We continue the initiated research line by refining both NP-hardness as well as feasibility results. We study the edge set partition problem into two trees, where we also add degree-bound restrictions. More specifically, we consider the problem of partitioning the edge set of a graph into two trees where one or both trees need to be paths, i.e., one or both edge-induced partition graphs must have maximum vertex degree 2. As a first group of results, we will show that despite this strong restriction, the two edge set partition problems remain NP-hard. As a second group of results, we will give exact algorithms for the three considered edge set partition problems.

2 Definitions and notation

We consider only undirected finite graphs, that will mostly be simple. Let $G = (V, E)$ be a graph, where $V = V(G)$ is the *vertex set* of G and $E = E(G)$ is the *edge set* of G. Edges are denoted as uv, where u and v are vertices of G. Formally, E is a set of two-element sets of vertices from V. A *multi-graph* is a construct $G = (V, E)$, where E is a multi-set of sets of at most two vertices from V. We will encounter multi-graphs only in intermediate steps in the construction of the proof of Theorem 3.6 and in the algorithm presented in Sec. 4. If uv is an edge of G then u and v are *adjacent* in G, and uv is *incident* to u and v; if uv is no edge of G then u and v are called *non-adjacent*. For a vertex pair u, v of G, if u and v are adjacent, then v is a *neighbor* of u. The *degree* of a vertex is the number of neighbors. Equivalently, for a (simple) graph, the degree of a vertex is the number of incident edges. A set X of vertices of G is an *independent set* of G if the vertices in X are pairwise non-adjacent in G. We use n to denote the number of vertices and m to denote the number of edges of a graph unless otherwise stated.

Let G be a graph, and let $A \subseteq E(G)$. The subgraph of G *induced* by A, denoted as $G[A]$, is the graph (V', A) where V' is the set of vertices of G that are incident to some edge in A. For a vertex pair u, v of G, a *u, v-path* in G is a sequence $(x_0, \ldots, x_r)$ of pairwise different vertices of G such that $x_0 = u$ and $x_r = v$ and $x_i x_{i+1} \in E(G)$ for every $0 \leq i \leq r-1$. If x_0 and x_r are also adjacent, then $(x_0, \ldots, x_r)$ is a cycle. A graph G is *connected* if there is a u,v-path in G for every vertex pair u, v of G. A connected graph that contains no cycle is called a *tree*. A tree whose vertices have degree at most 2 is called a *path*.

3 NP-hardness results for edge set partition problems

In this paper, we consider the complexity of the following problem: given a graph G, decide whether $E(G)$ can be partitioned into two sets (A, B) such that A and B induce acyclic connected subgraphs of G. Next to this general problem, we consider the two variants, where we add degree bounds to one or both edge-induced subgraphs. In this section, we consider lower bounds on the complexity of the three problems. It is known that the general tree-tree edge set partition problem is NP-complete [2, 9]. We show in this section that the path-tree edge set partition problem and the path-path edge set partition problem are NP-complete. The former result is achieved by a reduction from a Hamilton path problem, and the latter by a reduction from a satisfiability problem.

3.1 The path-tree edge set partition problem PTP

Recall that the path-tree edge set partition problem is, given a graph G, to decide whether $E(G)$ can be partitioned into two sets A and B such that $G[A]$ is a tree and $G[B]$ is a path. We show that this problem is NP-complete through a reduction from the Hamilton path problem on cubic bipartite graphs.

A graph G is called *bipartite* if $V(G)$ can be partitioned into set X and Y such that X and Y are independent in G. A graph G is called *cubic* if each vertex of G has degree 3 in G. A *cubic bipartite* graph is a graph that is both cubic and bipartite. It is an easy observation that cubic graphs have an even number of vertices. A *perfect matching* is a set A of edges of G such that $|A| = \frac{1}{2}|V(G)|$ and $G[A]$ contains all vertices of G. A set B of edges of G induces a *Hamilton path* if $G[B]$ is a path and contains all vertices of G. We say that G has a *Hamilton cycle* if there is $B \subseteq E(G)$ such that $G[B]$ contains all the vertices of G, is connected and each vertex of $G[B]$ has degree 2.

Lemma 3.1. *Let G be a cubic bipartite graph, and let (A, B) be a partition of $E(G)$. Then, B induces a Hamilton cycle in G if and only if A is a perfect matching of G and $G[B]$ is connected.*

Proof. Let B induce a Hamilton cycle in G; let $C = (x_1, \ldots, x_n)$ be the cycle. Since B induces a cycle, $G[B]$ is trivially connected. We show that A is a perfect matching of G. Note for every $1 \leq i \leq n$, A contains exactly one edge that is incident to x_i. Thus, A contains no pair of adjacent edges, and A is a matching for A. By construction, B contains exactly $2n$ edges, so that A contains exactly n edges, hence, matching A is perfect. For the converse, let A be a perfect matching of G and let $G[B]$ be connected. Note that $G[B] = G \setminus A$ and that every vertex of G has degree 2 in $G[B]$. A connected 2-regular graph is an induced cycle, and since every vertex of G appears in $G[B]$, B induces a Hamilton cycle in G. ■

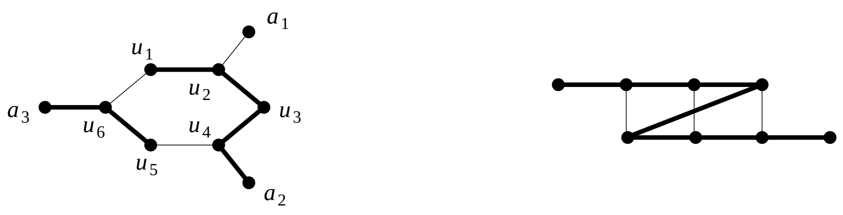


Figure 1: Two graphs that are used in the construction of the proof of Corollary 3.3.

The problem Hamilton Cycle is, given a graph G, to decide whether G has a Hamilton cycle. Analogously, the problem Hamilton Path is, given a graph G, to decide whether G has a Hamilton path.

Theorem 3.2 ([1, 7])**.** Hamilton Cycle *is* NP-*complete on cubic bipartite graphs.*

Corollary 3.3. Hamilton Path *is* NP-*complete on cubic bipartite graphs.*

Proof. We reduce from the Hamilton cycle problem on cubic bipartite graphs, that is NP-hard due toTheorem 3.2. Our reduction is in two steps. Let G be a cubic bipartite graph, and let a be an arbitrary vertex of G. We obtain G' from G by replacing vertex a by a chordless cycle $(u_1, \ldots, u_6)$ on six vertices and making the edges of G incident to a incident to u_2, u_4, u_6 in G. A schematic drawing of this local replacement is given on the left-hand side of Figure 1, where a_1, a_2, a_3 are the neighbors of a in G. It is important to note that G' is bipartite and contains only vertices of degree 2 or 3, and the vertices of degree 2 are exactly u_1, u_3, u_5. Let G_1, G_2, G_3 be three vertex-disjoint copies of G'. If x is a vertex in G', then x^i refers to the copy of x in G_i. We obtain the graph G^* as follows:

– $V(G^*) =_{\text{def}} V(G_1) \cup V(G_2) \cup V(G_3)$, and

– $E(G^*) =_{\text{def}} E(G_1) \cup E(G_2) \cup E(G_3) \cup \{u_5^1 u_1^2, u_5^2 u_1^3\}$.

We claim that G contains a Hamilton cycle if and only if G^* contains a Hamilton path. Namely, assume (w.l.o.g., due to symmetry) that $(a, a_2, \ldots, a_1, \ldots, a_3)$ describes the Hamilton cycle in G. Then, $(u_1^1, u_2^1, u_3^1, u_4^1, a_2^1, \ldots, a_1^1, \ldots, a_3^1, u_6^1, u_5^1, u_1^2, \ldots, u_4^2, a_2^2, \ldots, a_1^2, \ldots, a_3^2, u_6^2, u_5^2, u_1^3, \ldots, u_4^3, a_2^3, \ldots, a_1^3, \ldots, a_3^3, u_6^3, u_5^3)$ describes a Hamilton path in G^*. This situation is described using thick edges on the left-hand side of Figure 1. Conversely, observe that there is no Hamilton path in G^* with a contiguous sub-path that contains all vertices from $\{u_1^i, \ldots, u_6^i\}$ for any $i = 1, 2, 3$, because the edges $u_5^1 u_1^2$, $u_5^2 u_1^3$ must be used in any Hamilton path as bridges between the subgraphs G_1, G_2 and G_3. Therefore and because of the special role of degree-2 vertices, the Hamilton path has one end in $\{u_1^1, u_3^1\}$ and the other end in $\{u_3^3, u_5^3\}$. Observing the initial path from $u \in \{u_1^1, u_3^1\}$ to u_5^1 naturally describes a Hamilton cycle in G.

For completing the proof, we have to deal with the vertices of degree 2 of G^*. Note that these are exactly the following vertices: $u_1^1, u_3^1, u_3^2, u_3^3, u_5^3$. We replace each vertex and its two incident edges by the graph that is depicted on the right-hand side of Figure 1. It remains a small observation that the resulting graph has a Hamilton path if and only if G^* has a Hamilton path, which proves the claimed hardness result. ∎

In the Path Feedback Edge Set problem, PFES for short, we are given an undirected graph $G = (V, E)$ and the question is whether there exists a path P in G whose edges $E(P)$ form a feedback edge set, i.e., $G[E - E(P)]$ is acyclic.

Theorem 3.4. Path Feedback Edge Set *and* Path-Tree Edge Set Partition *are* NP-*complete on bipartite graphs of maximum degree at most 4.*

Proof. Since testing whether a graph is a tree or a path is easy, the two problems are clearly in NP.

For the NP-hardness of the problem, we reduce from Hamilton Path on cubic bipartite graphs, that is NP-hard due to Corollary 3.3. Let $G = (X, Y, E)$ be a cubic bipartite graph. Each vertex of G has exactly three neighbors, that we denote as $\nu_u(1), \nu_u(2), \nu_u(3)$. The index of each neighbor is chosen completely arbitrarily. Based on G, we construct a new graph G^*. Vertices of G are represented by vertex gadgets, that can be of two types. The two types are given in Figure 2. Vertices from X are represented by the vertex gadget on sixteen vertices, vertices from Y are represented by the vertex gadget on nine vertices. For a vertex u of G, X_u denotes the set of vertices of the vertex gadget for u. Each of the vertex gadgets has three connecting vertices, that are drawn as larger cycles in Figure 2; they provide connection between different vertex gadgets. For a vertex u of G, the three connecting vertices of its gadget are named as u_1, u_2, u_3. The assignment between name and vertex can be chosen arbitrarily. The large vertex gadgets have an explicitly named vertex u_0 for $u \in X$. These vertices are connected by a tree structure, as shown in Figure 2. The set of these tree vertices is denoted by K. It remains to define edges that connect the vertex gadgets. Let uv be an edge of G, and let i, j be such that $u = \nu_v(j)$ and $v = \nu_u(i)$. Then, G^* contains edge $u_i v_j$. These edges are called *terminal edges* in the course of the proof. This completes the definition of G^*. It is not difficult to see that G^* is a bipartite graph whose vertices have degree at most 4.

We can show that G has a Hamilton path if and only if $E(G^*)$ admits a partition into sets A and B such that $G^*[A]$ is a tree and $G^*[B]$ is a path. This proves the NP-hardness of PTP. NP-hardness of the second problem is shown by simply employing the graph $G^* \setminus K$. For the first direction, assume that G has a Hamilton path. Since G is bipartite and has an even number of vertices, it is a simple observation that X and Y contain the same number of vertices. Let $n =_{\text{def}} |X| = |Y|$, and let $P = (x^1, \ldots, x^{2n})$ be a Hamilton path in G. We construct a partition (A, B) of $E(G^*)$. We begin with the terminal edges. For $1 \leq i \leq 2n - 1$, where $u = x^i$ and $v = x^{i+1}$ and $u = \nu_v(q)$ and $v = \nu_u(p)$, $\{u_p v_q\} \subseteq B$; all other terminal edges are in A. Intuitively, B is the set of edges on P, and A contains the edges of G that are not used by P. Observe that for every vertex but two, namely x^1 and x^{2n}, exactly two terminal edges are selected for B and the third terminal edge is contained in A. For each of x^1 and x^{2n}, one terminal edge is selected for B and the two other terminal edges are contained in A. Using the configurations for the vertex gadgets as depicted in Figure 2, it is not difficult to see that the definition of (A, B) can be completed to satisfy the claimed properties, that $G^*[A]$ is a tree and $G^*[B]$ is a path. Note here that A contains all edges of G^* that are incident to vertices in K. For PFES, it suffices to observe that $(G^*[B]) \setminus K = G^*[B]$ and that $G^*[A] \setminus K$ is a forest, thus acyclic.

For the converse, let (A, B) be a partition of $E(G^*)$ such that $G^*[A]$ is a tree and $G^*[B]$ is an induced path. Let $P = (x^1, \ldots, x^r)$ be the path in G^* that is induced by B. We show two

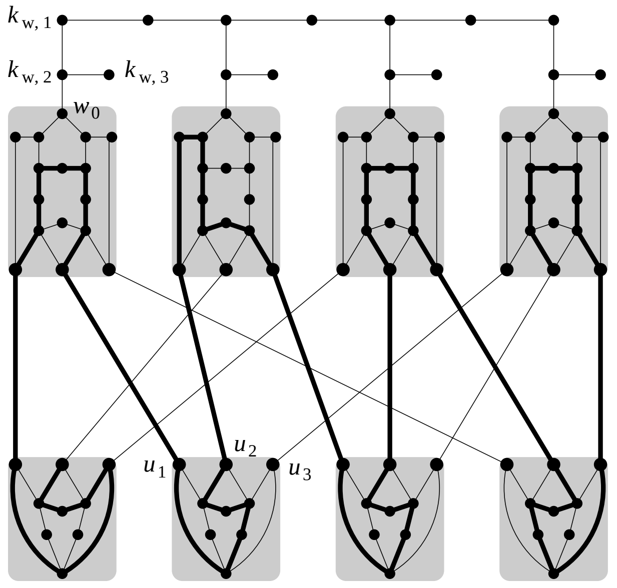


Figure 2: The figure shows to result of the reduction in the proof of Theorem 3.4 for a cubic bipartite graph on four vertices in each color class. The grey areas represent the two types of vertex gadgets.

auxiliary results about the structure of P. Let u, v be a vertex pair of G and let $i < i+1 < j$ be such that $x^i x^{i+1} \in E(G^*[X_u])$ and $x^j x^{j+1} \in E(G^*[X_v])$. Suppose for a contradiction that $\{x^i, \ldots, x^j\} \cap K \neq \emptyset$. Since $G^*[K]$ is a tree, it is not difficult to see that this situation requires $(w_0, k_{w,2}, k_{w,1})$ for some $w \in X$ to be a subgraph of $G^*[B]$, thus isolating the edge $k_{w,2}k_{w,3}$. So, $\{x^i, \ldots, x^j\} \subseteq \bigcup_{u \in V(G)} X_u$. Assume that there is $i < t < j$ such that $\{x^i, \ldots, x^t\} \subseteq X_u$ and $\{x^{t+1}, \ldots, x^{j+1}\} \subseteq X_v$. Since $G^*[B]$ is a path, $x^t x^{t+1} \in E(G^*)$. This means that $x^t x^{t+1}$ is a terminal edge, and therefore, u and v are adjacent in G. We show now that P can be used to construct a Hamilton path in G.

Since B contains an edge from every cycle of G^*, the following is an easy observation: for every $u \in V(G)$, $B \cap E(G^*[X_u]) \neq \emptyset$. So, for every vertex u of G, there is a smallest index i such that $x^i x^{i+1} \in E(G^*[X_u])$; denote this number by $a(u)$. Let $\langle z^1, \ldots, z^{2n} \rangle$ be the vertex ordering for G that satisfies $a(z^1) < a(z^2) < \cdots < a(z^{2n})$. We show that $Q = (z^1, \ldots, z^{2n})$ corresponds to a Hamilton path for G. Due to construction, each vertex of G appears exactly once on Q, so it remains to consider pairs of consecutive vertices. We essentially show that $z^i z^{i+1} \in E(G)$ for every $1 \leq i \leq 2n-1$, with at most one exception. We show the claim by induction. Let $1 \leq i \leq 2n-1$. Let $p' =_{\text{def}} a(z^i)$ and $q =_{\text{def}} a(z^{i+1})$. Additionally to the claim, we assume that for every vertex $z^2, \ldots, z^{i-1}$, two of its terminal edges appear on the path $(y^1, \ldots, y^{p'})$. Let p be largest such that $\{y^{p'}, \ldots, y^p\} \subseteq X_{z^i}$. We consider the y^p, y^q-path $(y^p, \ldots, y^q)$ in $G^*[B]$. If $q = p+1$, then $y^p y^q$ is a terminal edge and $z^i z^{i+1} \in E(G)$ due to the above auxiliary result. In the other case, assume that $q \geq p+2$. We consider the y^p, y^q-path $(y^p, \ldots, y^q)$ in $G^*[B]$. Remember that $\{y^p, \ldots, y^q\} \subseteq \bigcup_{u \in V(G)} X_u$. Let $u \in V(G)$ be such that $y^{p+1} \in X_u$. By the choice of p, it holds that $u \neq z^i$. The construction of G^* requires that y^{p+2} is a vertex from X_u, too. Thus,

$y^{p+1}y^{p+2} \in E(G^*[X_u])$. Due to the definition of $a(u)$ and Q, it follows that $u \notin \{z^{i+1}, \ldots, z^{2n}\}$, so that $u \in \{z^1, \ldots, z^{i-1}\}$. Let $p + 2 \leq t \leq q$ be largest such that $\{y^{p+1}, \ldots, y^t\} \subseteq X_u$. Clearly, $y^{t+1} \notin X_u$. Now, observe that $y^p y^{p+1}$ and $y^t y^{t+1}$ are terminal edges for u. Note that these two terminal edges could not have been used before, and thus, $u = z^1$. Let $v \in V(G)$ be such that $y^{t+1} \in X_v$. If $v \neq z^{i+1}$ then, with the same arguments, v is a vertex from $\{z^1, \ldots, z^i\}$. However, $(y^1, \ldots, y^t)$ contains at least two of the three terminal edges of each of the vertices $z^1, \ldots, z^i$, which yields a contradiction. Thus, if $q \geq p+2$ then $z^i z^1$ and $z^1 z^{i+1}$ are edges of G. We conclude that Q is a path of G or that there is $2 \leq j \leq 2n - 1$ such that $(z^2, \ldots, z^j, z^1, z^{j+1}, \ldots, z^{2n})$ is a path of G. Thus, G contains a Hamilton path.

For PFES, observe that the connectedness property for $G^*[A]$ was used only in proving the first auxiliary result, when being applied to edges incident to vertices from K. Thus, starting from an appropriate partition of $E(G^* \setminus K)$ yields the desired result. This completes the proof of the theorem. ■

3.2 The path-path edge set partition problem PPP

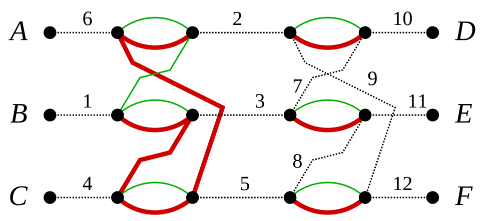
The path-path edge set partition problem is, given a graph G, to decide whether $E(G)$ can be partitioned into sets A and B such that $G[A]$ and $G[B]$ are trees and each vertex in $G[A]$ and in $G[B]$ has at most two neighbors. We show that this problem is NP-complete through a reduction from a variant of the satisfiability problem.

A *3-CNF formula* is a formula of propositional logic in conjunctive normal form where each clause contains exactly three literals. An assignment β is also considered as mapping formulas to truth values, hence identifying it with the induced interpretation function. The problem Not-all-equal-3-Sat, Nae-3Sat for short, is the problem, given a 3-CNF formula φ, to decide whether φ has a truth-value assignment β so that each clause of φ contains a literal that is true under assignment β and a literal that is false under assignment β. Such an assignment β satisfies φ if and only if the complementary assignment of β satisfies φ.

Theorem 3.5 ([10]). Nae-3Sat *is* NP-*complete.*

Theorem 3.6. Path-Path Edge Set Partition *is* NP-*complete.*

Proof. We show the result by reducing from Nae-3Sat [10]. Given a 3-CNF formula, our construction will mainly use special graphs as clause gadgets and connect the gadgets. For the beginning, we study properties of the designated clause gadget graph. Consider the top graph in Figure 3, that we denote as K. Note that K is a multi-graph, since there are multiple edges between vertex pairs. We will later derive a simple graph by subdividing these edges. Structurally, it is important to observe that K contains six cycles of length 2 and two cycles of length 6. We consider an arbitrary graph G that contains K as an induced subgraph, and we assume that G admits a partition of $E(G)$ into two set, A, B, such that $G[A]$ and $G[B]$ are paths. Since $G[A]$ and $G[B]$ are acyclic, every cycle of K must have an edge in A and an edge in B. Due to the symmetry of K, we can assume, without loss of generality, that A contains the edges of K that are drawn as thick red in Figure 3 and B contains the edges that are thin green. The dotted edges are yet to be decided. We can show that the situation for the remaining edges is completely determined by this beginning, through a sequence of implications:



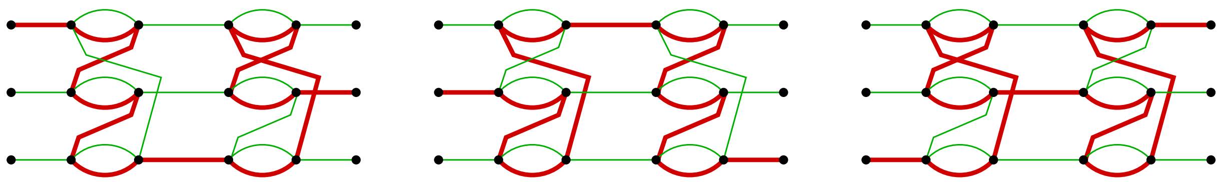


Figure 3: The above multi-graph defines the clause gadget used in the proof of Theorem 3.6. The colors of the edges, alternatively represented as thick, thin and dotted segments, stand for a subsolution of a partition into two subgraphs of degree at most 2. Depending on the actual choice of the partition, the three colored graphs below show the unique full partitions.

1) edges 1 and 2 are adjacent to two green edges, so they must be red; otherwise, $G[B]$ would contain a vertex with more than two neighbors

2) edges 3, 4, 5, 6 are adjacent to two red edges, so they must be green; otherwise, $G[A]$ would contain a vertex with more than two neighbors

3) edges 7 and 8 are adjacent to two green edges, and edge 9 is adjacent to two red edges, so 7 and 8 must be red and 9 must be green

4) analogously, 10 and 11 must be green and 12 must be red.

The final result is depicted in the middle graph on the bottom line of Figure 3. The important property that we conclude is the following:

Fact: *If edge 1 is red and edges 4 and 6 are green, then edge 12 is red and edges 10 and 11 are green.*

Due to the symmetry of K, the following is a direct consequence of the above Fact.

Result: *Assume that exactly one of the three edges 6, 1, 4 is red and the other two are green. Then, exactly one of the edges 10, 11, 12 is red, and the following holds: if 6 is red then 11 is red, if 1 is red then 12 is red, and if 4 is red then 10 is red.*

The corresponding edge set partitions are shown on the bottom line of Figure 3.

We draw the following important corollary, that is the basis for our reduction: if K represents a clause of a 3-CNF formula, then the variable assignment information can pass through gadget K. This can be also seen in the example described in Figure 4.

We are ready for giving the reduction more formally. Let φ be a 3-CNF formula. Let $x_1, \ldots, x_n$ be the variables that appear in φ and let $L_1, \ldots, L_m$ be the clauses of φ. By definition, every clause contains exactly three literals, and every clause contains at least two different

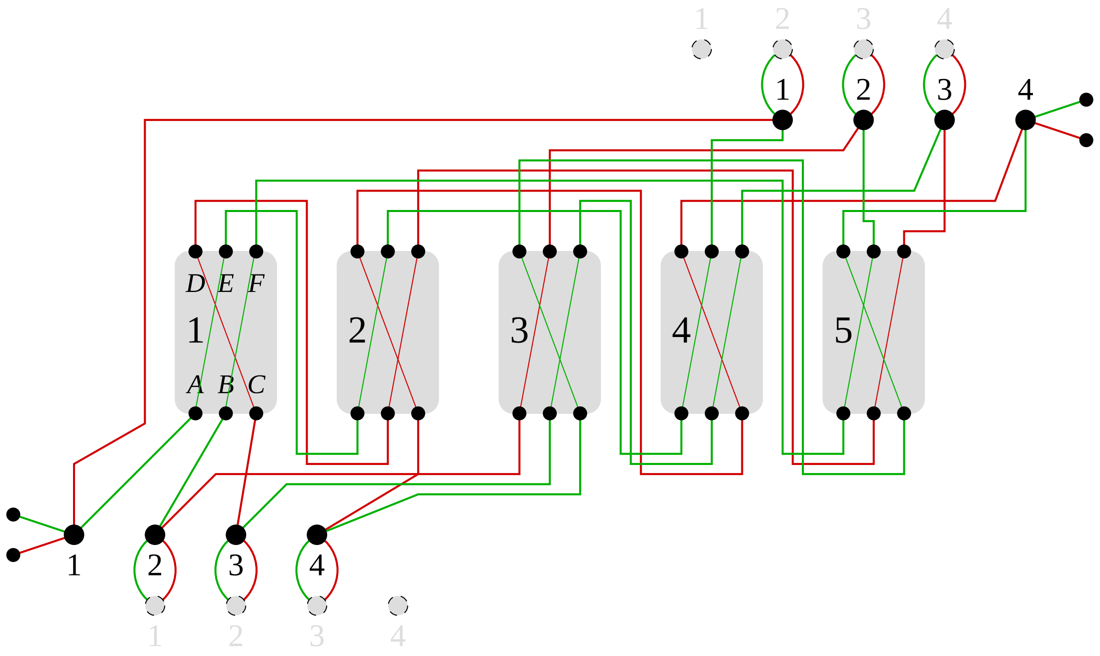


Figure 4: The figure shows the result of the construction of the proof of Theorem 3.6, applied to the formula $\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee \neg x_3 \vee \neg x_4)$. In addition, the colors of the edges define a partition of the edge set into two paths. One color connects all true literals, and the other color connects all false literals for the given assignment β, where $\beta(x_1) = \beta(x_2) = \beta(x_3) = 1$ and $\beta(x_4) = 0$. The clause gadgets, a copy for each of the five clauses, are schematically drawn as grey areas.

variables. If a variable appears as positive and as negative literal in the same clause then every assignment satisfies this clause, and the clause can safely be deleted. For every $1 \leq i \leq j$, denote the literals of L_i as a_i, b_i, c_i, where the actual assignment is arbitrary.

From φ, we construct a multi-graph H_φ. Let $K_1, \ldots, K_m$ be m copies of K, where K_i shall correspond to L_i. Denote the distinguished vertices A, B, C, D, E, F of K in K_i as $A_i, \ldots, F_i$. Let $\psi : \{x_1, \neg x_1, x_2, \ldots, \neg x_n\} \times \{1, \ldots, m\} \to \{1, \ldots, m\}$ be the function that assigns to every literal occurrence the "closest" clause with another occurrence of the same literal. Formally, for $x \in \{x_1, \ldots, \neg x_n\}$ and $1 \leq i \leq m$ where $x \in \{a_i, b_i, c_i\}$, if literal x occurs in some of the clauses $L_{i+1}, \ldots, L_m$ then let $\psi(x, i) =_{\text{def}} j$ for j smallest with $i < j \leq m$ and $x \in \{a_j, b_j, c_j\}$, and if $x \notin \{a_j, b_j, c_j : i < j \leq m\}$ then let $\psi(x, i) =_{\text{def}} i$. We incrementally define the target graph H_φ. We begin with the disjoint union of the graphs $K_1, \ldots, K_m$; we add the following vertices and edges:

- For every $1 \leq i \leq n$, add the new vertices α_i and ω_i.

- Let $1 \leq l \leq n$ and $x \in \{x_l, \neg x_l\}$ and $1 \leq i \leq m$, where $x \in \{a_i, b_i, c_i\}$; let $i \leq j \leq m$ be with $\psi(x, i) = j$:
  if $x = a_i$, then let $u =_{\text{def}} E_i$; if $x = b_i$, then let $u =_{\text{def}} F_i$; if $x = c_i$, then let $u =_{\text{def}} D_i$;
  if $x = a_j$, then let $v =_{\text{def}} A_j$; if $x = b_j$, then let $v =_{\text{def}} B_j$; if $x = c_j$, then let $v =_{\text{def}} C_j$;

if $i < j$, then add edge uv; and
if $i = j$, then add edge $u\omega_l$.

– Let $1 \leq l \leq n$.
If x_l occurs in φ, then let j be smallest such that $x_l \in \{a_j, b_j, c_j\}$;
if $x_l = a_j$, then add edge $\alpha_l A_j$; if $x_l = b_j$, then add edge $\alpha_l B_j$; if $x_l = c_j$, then add edge $\alpha_l C_j$.

– Let $1 \leq l \leq n$.
If $\neg x_l$ occurs in φ, then add $\alpha_l A_j$ or $\alpha_l B_j$ or $\alpha_l C_j$ analogous to the preceding case.

– Let $1 \leq l \leq n$.
If x_l or $\neg x_l$ does not occur in φ, then add edge $\alpha_l \omega_l$.

– For every $1 \leq i < n$, add two copies of edge $\omega_i \alpha_{i+1}$.

– Add four new vertices and make two of them adjacent to α_1 and the others adjacent to ω_n.

This completes the construction of H_φ. An example for a formula with five clauses is given in Figure 4. For later considerations, the following observations are important. For $1 \leq i \leq n$, there is an α_i, ω_i-path in H_φ that connects all occurrences of literal x_i, and there is an α_i, ω_i-path in H_φ that connects all occurrences of literal $\neg x_i$. We will refer to the two paths as the *x_i-path* and the *$\neg x_i$-path*. The two paths are not unique, but the edges incident to α_i and ω_i are unique for each path.

We show that φ has a satisfying assignment with the correct property if and only if H_φ admits a partition of $E(H_\varphi)$ into two paths. Let $H =_{\text{def}} H_\varphi$. First, let β be a satisfying assignment for φ such that each clause of φ has a true and a false literal. It is as simple as important that $I_\beta(x_1)$ is true if and only if $I_\beta(\neg x_i)$ is false. We describe a partition of $E(H)$ into A and B. Intuitively, we partition the edges along the two paths connecting α_i and ω_i such that one path is completely in A and the other is completely in B. We first partition the edges of the clause gadgets. For $1 \leq i \leq m$, if the literal associated with A_i is true with respect to I_β then the incident edge in K_i is in A, otherwise it is in B; analogously for B_i and C_i. Due to the properties of β, it follows that exactly two of the three edges are either in A or in B. Following the cases in Figure 3, this pre-partition uniquely extends to all edges of K_i. We can extend the partition of the edges of the clause gadgets to complete x_i- and $\neg x_i$-paths. We obtain a partition into two paths. An example partition is given in Figure 4.

For the converse, assume that $E(H)$ admits a partition (A, B) such that $H[A]$ and $H[B]$ are paths. Note that each of $H[A]$ and $H[B]$ contains exactly one of the two pending edges that are incident to α_1 and ω_n. Due to the previously established results, (A, B) on each of the clause gadgets corresponds to one of the three situations in Figure 3. We color the vertices $A_1, B_1, C_1, \ldots, A_m, B_m, C_m$ as "red" and "green" as follows: for every $1 \leq j \leq m$, if the edge of K_j incident to A_j is in A then A_j is colored "red", otherwise, the edge is in B, A_j is colored "green"; analogously for B_j and C_j. We show that this coloring is consistent, which means that colored vertices that represent the same literal have the same color, and vertices that represent complementary literals, i.e., x_i and $\neg x_i$, have different colors. For an illustration of the arguments, we also refer to the example in Figure 4.

First, we consider $\alpha_1, \ldots, \alpha_n$. Observe that α_1 is adjacent to two pending vertices, and the two incident edges must belong to different partition classes (otherwise, one of the two partition classes would consist of exactly two edges, which contradicts the properties for the clause gadgets). Thus, the two other edges that are incident to α_1 must also belong to two different color classes. Now, consider α_i for $2 \leq i \leq n$. The two edges connecting ω_{i-1} and α_i form a cycle and therefore belong to different partition classes, and thus, the two other edges incident to α_i belong to two different partition classes.

Next, consider a clause gadget K_j and a vertex X for $X \in \{A_j, B_j, C_j, D_j, E_j, F_j\}$. Note that X has degree 2 and cannot be a vertex of degree 1 in $H[A]$ or $H[B]$. Thus, the two incident edges must belong to the same partition class. Let $1 \leq i \leq n$ be such that both literals of x_i – x_i and $\neg x_i$ – appear in φ. Let $1 \leq j, j' \leq m$ be smallest such that x_i appears in L_j and $\neg x_i$ appears in $L_{j'}$. Let $X \in \{A_j, B_j, C_j\}$ and $X' \in \{A_{j'}, B_{j'}, C_{j'}\}$ be such that X and X' correspond to respectively x_i and $\neg x_i$. Due to the above definition and shown properties, X is colored "red" if and only if X' is colored "green". Now, let $2 \leq j \leq m$ be arbitrary and let $Y \in \{A_j, B_j, C_j\}$. As remarked above, Y is incident to exactly two edges. Assume that Y is not adjacent to a vertex α_i. Then, there is $1 \leq j' < j$ and $Z \in \{D_{j'}, E_{j'}, F_{j'}\}$ such that Y and Z are adjacent. Note that the three edges that are incident to Y and Z belong to the same partition class. By the shown auxiliary result, it directly follows

– if $Z = E_{j'}$ then Y and $A_{j'}$ represent the same literal and they have the same color

– if $Z = F_{j'}$ then Y and $B_{j'}$ represent the same literal and they have the same color

– if $Z = D_{j'}$ then Y and $C_{j'}$ represent the same literal and they have the same color.

By induction, the claimed results holds, and the coloring of the vertices $A_1, B_1, \ldots, C_m$ is indeed consistent.

We define truth-value assignment β as follows. Let $1 \leq i \leq n$, and assume that literal x_i appears in φ. Let $1 \leq j \leq m$ be smallest such that literal x_i appears in L_j. Let $\beta(x_i) =_{\text{def}} 1$ if the vertex of K_j corresponding to x_i, so, one of A_j, B_j, C_j, is colored "red"; otherwise, let $\beta(x_i) =_{\text{def}} 0$. If literal x_i does not occur in φ then literal $\neg x_i$ occurs in φ, and, analogous to the previous case, we let $\beta(x_i) =_{\text{def}} 1$ if and only if the corresponding vertex in the first clause gadget is colored "green". Remember that the vertex coloring is consistent. Due to the definition of β and the consistency, it follows for every $1 \leq j \leq m$ that $I_\beta(a_j) = 1$ if and only if A_j is colored "red"; analogously for b_j and c_j. Due to the auxiliary result, for every $1 \leq j \leq m$, one of A_j, B_j, C_j is colored "red" and one of them is colored "green". We conclude that β is a satisfying assignment for φ of the desired form.

To complete the proof, we have to transform H into a simple graph by resolving the multiple edges. We obtain the final graph H^* by subdividing multiple edges. It is not difficult to see that a subdivision operation introduces a vertex of degree 2, that can be either the beginning of the two paths or the two incident edges must belong to the same path. Since H^* has exactly four vertices of degree 1, that must form the four endpoints of the two paths, the two edges incident to a subdivision vertex must belong to the same partition set. Thus, the above shown equivalence between satisfiability of φ in the desired sense and the existence of a partition of $E(H)$ into two paths directly translates to H^*.■

Biedl and Brandenburg considered the complexity of deciding whether a given graph G admits an edge set partition into k trees for k some fixed integer. They showed that this problem is NP-complete for every $k \geq 2$ [2]. A similar result holds for the case of partitioning the edge set of a given graph into k paths, where $k \geq 2$; we apply the reduction of the proof of Theorem 3.6 and add $k-2$ isolated edges. Each of these edges must form a single path in a possible partition, and thus, the construction of Theorem 3.6 directly extends to arbitrary k.

4 Partition into two paths

In this section, we present a Monte Carlo algorithm for the Path-Path Edge Set Partition problem with a running time of $O(1.784^n)$. Then, we show that this algorithm can be adapted to design a deterministic $O(1.8906^n)$-time algorithm for the problem. We start with a simple property of graphs whose edge set admits such a partition.

Lemma 4.1. *Let G be a graph that is not a path. If $E(G)$ can be partitioned into sets A and B such that $G[A]$ and $G[B]$ are paths, then G has maximum vertex degree 4 and G has at most four vertices of odd degree. The number of odd-degree vertices is always even. If it is zero, then both paths have the same endpoints. If it is two, then both paths share exactly one endpoint.*

Proof. Let (A, B) be a partition of $E(G)$ such that $G[A]$ and $G[B]$ are paths. Since G is not a path, A and B must be non-empty. Each vertex of $G[A]$ has degree 1 or 2 and each vertex of $G[B]$ has degree 1 or 2. Thus, each vertex of G has degree at most 4. Furthermore, since exactly two vertices of $G[A]$ and of $G[B]$ have degree 1, at most four vertices of G can have odd degree. If the two paths do not share any endpoints, then exactly four vertices of G have odd degree. If the two paths share exactly one endpoint, then that shared (endpoint) vertex has degree two, while the other two endpoints are at vertices of odd degree. If the two paths have the same endpoints, then these shared (endpoint) vertices have degree two, so there are no odd degree vertices. ■

Given a graph $G = (V, E)$, an instance to the problem, we first regularize it in Phase 0 that consists of two steps:
1. If G has two vertices of odd degree, then create from G at most n many instances by selecting a vertex v of degree two and attaching a new pendant vertex (v') to v. If G has zero vertices of odd degree, then create from G at most $n^2 - n$ many instances by selecting two vertices u, v of degree two and attaching a new pendant vertex (u' resp. v') to u resp. v. Continue with Step 2. of Phase 0 (and the following phases) for each of the created instances (which we refer to as G in the following for simplicity).
2. Apply the following reduction rules as long as possible.

Rule 1. If u is a vertex of degree two with neighbors x and y, delete u and add an edge between x and y.

Rule 2. If u is a vertex of degree three, add a new vertex u' and connect it to u.

Rule 3. If there are at least three edges between two vertices or if there is a loop, i.e., an edge connecting a vertex x with itself, the given instance is a no-instance.

The rules create an equivalent loopless multi-graph with (possible) double edges between vertices that has exactly four vertices of odd degree, and this degree is one. So, it is known where the paths start.

Let G_1 be the (multi-)graph obtained by repeatedly applying Rule 1 until no longer possible. Clearly, $E(G_1)$ can be partitioned into two paths iff $E(G)$ can be partitioned into two paths. By Lemma 4.1, if $E(G_1)$ indeed admits a partition into two paths, the endpoints of these paths are either of degree one or three. Reduction Rule 2 simply ensures that the endpoints are always of degree one.

In broad brush-strokes the algorithm now works as follows. It colors the edges alternately red and green and after it has finished coloring all edges, checks whether the edges of each color induce a path. If yes, the algorithm outputs the paths; else, it starts afresh (i.e., at the begining of Phase 1), coloring the edges alternately all over again. Call a 2-coloring of the edges of a graph a *good coloring* if the edges of both colors induce a path. We show that, given a yes-instance, the algorithm runs in time $O(1.784^n)$ and finds a good 2-coloring with high probability.

The algorithm colors the graph in three phases, which we now describe. At the end of each phase, the following invariant is preserved:

> **Invariant.** Every vertex of degree four either has all its incident edges colored or exactly two of its incident edges colored with an equal number of edges colored red and green.

In Phase 1, the algorithm starts at an arbitrary vertex u of degree one and colors the edge uv incident to it red. It then moves to vertex v and considers the edges incident to v. At any given point in Phase 1, when the algorithm reaches a vertex w via an edge xw, there are three possibilities. Vertex w is of degree four with three uncolored edges incident to it, in which case the algorithm picks an uncolored edge wz with probability $1/3$ and colors it red or green depending on whether xw is green or red, and moves to vertex z. Vertex w is of degree four and two of its edges are colored. In this case, one of these edges is colored red and the other green. Suppose the algorithm reached w via edge xw and that the edge wy is as yet uncolored. If the algorithm colored xw red, then it colors wy green and red, otherwise. Finally, vertex w could be of degree one, in which case the algorithm proceeds to Phase 2.

At the end of Phase 1, exactly two vertices u, v of degree one have been visited by the algorithm and the edges e_u, e_v incident to both of these have been assigned colors. Phase 2 starts at a vertex x of degree one which has not yet been visited in the previous phase. The edge e_x incident to this vertex receives a color that depends on the colors assigned to e_u, e_v. If both e_u, e_v are colored red, edge e_x is colored green; if both e_u, e_v are colored green, edge e_x is assigned red; otherwise e_x is assigned red or green with equal probability. The algorithm now proceeds as in Phase 1. If it reaches a vertex y via edge e, then there are three possibilities. Either y is of degree four and all its incident edges are uncolored, or it has two incident edges that are colored, or y is a vertex of degree one. In the first case, the algorithm picks an uncolored edge with probability $1/3$ and assigns it a color different from that of e. In the second, the algorithm assigns the remaining uncolored edge a color such that vertex y has two red and two green edges. Finally if y has degree one, Phase 2 is complete. Note that the invariant continues to hold at the end of Phase 2.

At the end of Phase 2, edges incident to vertices of degree one have been assigned colors. At the beginning of Phase 3, the algorithm does a simple sanity check: there must be two red edges and two green edges incident to the vertices of degree one. If this is not the case, the algorithm abandons this coloring and starts afresh. Otherwise if all edges have been assigned colors, the algorithm checks whether the edges of each color induce a path. If they do, the algorithm outputs these two paths and halts; else, it starts coloring the edges afresh. If, however, all edges have not been assigned colors, the algorithm does the following. It first assigns colors red and green to the edges of 2-cycles. Recall that the graph that we are dealing with is possibly a multi-graph and 2-cycles can occur in the graph. This assignment of colors is valid because both edges of a 2-cycle cannot belong to the same path. Again the invariant continues to hold after all 2-cycles are colored.

There may still be edges that have not yet received a color. The algorithm now works in a sequence of subphases. In each subphase, it picks an uncolored edge xy arbitrarily and colors it red or green with equal probability. It then moves to vertex y (say) and sees if there are any edges left to color. In this case, we call vertex x the *initial vertex* of the subphase. If all edges incident to y are uncolored, the algorithm picks one of the uncolored edges with equal probability (say, yz) and colors it with a color different from the one assigned to xy and moves to z. If y has two of its incident edges colored then these must be differently colored by the invariant. It colors the last uncolored edge (say, yw) red or green so that there are an equal number of red and green edges incident to y and moves to w. Note that vertex x may have two of its incident edges colored or none of them colored prior to the beginning of the subphase. By coloring xy, we have a vertex x that has either only three incident edges colored or one incident edge colored. In either case, we have a vertex where the invariant does not hold, but this is the only vertex where it happens.

The algorithm continues coloring edges alternately until it gets "stuck", that is, reaches a vertex which has all its incident edges colored. This can happen only when the algorithm reaches the initial vertex x. The algorithm checks whether x has the same number of red and green edges; if not, the algorithm aborts this coloring and starts afresh. Therefore, in each subphase the algorithm colors edges that are part of (possibly non-simple) cycles. If all edges are still not yet colored, the algorithm starts another subphase by picking another uncolored edge $x'y'$, and coloring it red or green with equal probability and proceeding in this fashion till all edges are colored. Note that at the end of each subphase, the invariant continues to hold. This concludes the explanation of how the algorithm works.

The following result allows to infer that Phase 3 deals with cycles of a certain minimal length, which is important for the estimate of the running time.

Lemma 4.2. *In each subphase of Phase 3, the algorithm colors the edges of a (possibly non-simple) cycle of length at least four.*

Proof. The fact that in each subphase the algorithm colors edges of a cycle is clear, because the only way a subphase ends is when the algorithm reaches a vertex which has all its incident edges colored. That is, prior to the algorithm having visited that vertex, three of its incident edges have already been assigned colors. From the algorithm description, we have seen that in

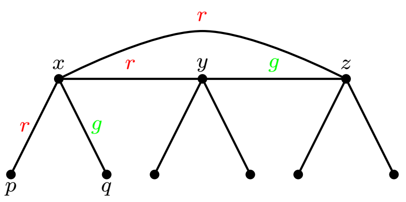


Figure 5: The algorithm picks edge xy at the beginning of a subphase; vertex x is the initial vertex.

each subphase the initial vertex is the only vertex where the invariant is not maintained. This shows that for a subphase to end, the algorithm must cycle back to the initial vertex. This cycle cannot be of length two, as all such cycles have already been colored at the beginning of Phase 3. If this cycle is of length three, then we have the situation depicted in Figure 5. Suppose the algorithm starts a subphase by picking edge xy, colors three edges alternately, and is unable to proceed, because it has reached x once again. This means that the edges xp and xq had already been assigned colors prior to the starting of this subphase. Since the invariant holds at the end of Phases 1 and 2 and at the end of each subphase, this means that one of them is colored red and the other is colored green. Hence, x has three edges incident to it which are of the same color, a situation in which the algorithm aborts the coloring and starts afresh. ■

Theorem 4.3. *If G is a yes-instance of the* Path-Path Edge Set Partition *problem, then the algorithm runs in time $O(1.784^n)$ and fails to find a partition (A, B) of $E(G)$ such that $G[A]$ and $G[B]$ induce paths with exponentially small probability.*

Proof. When the algorithm is at a vertex, three of whose incident edges are uncolored, two of these edges must belong to the same path. Hence, the algorithm correctly guesses the color with probability 2/3. If the algorithm is at a vertex which has two of its incident edges colored, then the color of the fourth edge is automatically decided and no guessing is required. Finally, in each subphase of Phase 3, algorithm guesses the color of an uncolored edge and correctly does so with probability 1/2. However by Lemma 4.2 along with the fact that there are no 2-cycles, in each subphase, the algorithm colors the edges of a cycle of length at least four. Let n_1 be the number of vertices of degree four that are visited by the algorithm in Phases 1 and 2. Then each of these n_1 vertices has at least two incident edges that receive colors. Therefore, the total number of edges that receive colors in Phases 1 and 2 is at least $n_1 - 2$, as the colored edges may induce at most two connected components. The number of edges that are yet to receive colors is at most $(2n - 6) - (n_1 - 2)$. This follows, because the graph has exactly four vertices of degree one and $n - 4$ vertices of degree four which puts the number of edges to $2n - 6$. Now in each subphase of Phase 3, the algorithm makes one guess (with probability 1/2) and colors at least four edges and hence the number of subphases is at most $(2n - n_1 - 4)/4$. The probability that the algorithm succeeds in finding a correct 2-coloring of the edges is at least

$$\left(\frac{2}{3}\right)^{n_1} \left(\frac{1}{2}\right)^{(2n-n_1-4)/4}.$$

For $n \in \mathbf{N}$ and $0 \leq n_1 \leq n$, define $f(n, n_1) = 1.5^{n_1} \cdot 2^{(2n-n_1-4)/4}$. Now $\log_2 f(n, n_1) \leq 0.335n_1 + 0.5n \leq 0.835n$ and hence $f(n, n_1) \leq 2^{0.835n} < 1.784^n$. Hence, if we run the algorithm for $c \cdot f(n, n_1)$ times for some constant c, the probability that the algorithm fails to obtain a good 2-coloring on a yes-instance is at most

$$\left(1 - \frac{1}{f(n, n_1)}\right)^{c \cdot f(n,n_1)} < \left(1 - \frac{1}{1.784^n}\right)^{c \cdot 1.784^n} < \left(\frac{1}{e}\right)^c .$$

By choosing c to be large enough, we can make the error probability exponentially small. ∎

To conclude this section, we show that the idea of the previously described algorithm can be reused to derive a desterministic algorithm.

Theorem 4.4. *There is a deterministic algorithm solving PPP in time $O(1.8906^n)$.*

Proof. Our deterministic algorithm is based on the randomized one given above. However, rather than randomly assigning some colors to some uncolored edges, this algorithm tries all possible assignments.

Let $G = (V, E)$ be a graph. By Lemma 4.1, the maximum degree of G is at most 4, since otherwise we face a no-instance. Assume that G is a yes-instance and let A and B be a partition of $E(G)$ such that $G[A]$ and $G[B]$ are paths. Assume that the edges of A are colored red and the ones of B are colored green. In the following, given a vertex v, we denote by $\tilde{d}(v)$ the number of uncolored edges incident to v.

Description of the algorithm. By Lemma 4.1, G has at most 4 vertices of odd degree. Our algorithm starts by exhaustively coloring all edges incident to vertices of odd degree. If G has no vertices of odd degree, the algorithm arbitrarily picks an edge uv which is colored red. In each step, it recursively applies the following reduction rules (their correctness is straightforward) to obtain a *reduced* instance:

Red1. If u is a vertex with $d(u) = 2$ and $\tilde{d}(u) = 1$, then its unique uncolored incident edge is colored red (resp. green) if its colored incident edge is red (resp. green).

Red2. If u is a vertex with $d(u) = 4$ and $\tilde{d}(u) = 1$, then its unique uncolored incident edge is colored by the unique possible color (which is determined by the color of the colored incident edges).

Red3. If u is a vertex with $d(u) = 4$ and $\tilde{d}(u) = 2$ such that its two colored incident edges have the same color, then the two uncolored incident edges are colored by the opposite color.

As soon as we have a reduced instance, the algorithm picks a vertex v with exactly one incident colored edge, if one exists. W.l.o.g., we assume that the color of this edge is red. Let v_1, v_2 and v_3 be the neighbors of v such that the edges vv_1, vv_2 and vv_3 are uncolored. The algorithm branches on the following two possible cases which are then recursively solved:

(A) edge vv_1 is colored red, and both vv_2 and vv_3 are colored green;

(B) edge vv_1 is colored green and both vv_2 and vv_3 are still uncolored.

If no such vertex v having exactly one incident colored edge exists, then by application of **Red3**, the graph induced by the uncolored edges consists of a collection of cycles. If a solution (extending the current partially colored instance) exists, these cycles have to be alternately colored with the two colors. Thus, the length of these cycle is even and is at least 4. Otherwise, the current partially colored instance is a no-instance and the algorithm stops the exploration of this subproblem. For each cycle, the algorithm tries the two possible alternating coloring.

Running time analysis. To analyse the running-time, let us consider the measure $\mu = w_2 \cdot n_2 + w_3 \cdot n_3 + w_4 \cdot n_4$, where n_i denotes the number of vertices in any reduced instance being incident to exactly i uncolored edges, $2 \leq i \leq 4$, and $w_2 = \frac{1}{3}$, $w_3 = \frac{2}{3}$ and $w_4 = 1$.

Let $T(\mu)$ be an upper-bound on the worst-case running time. We note that the three reduction rules cannot increase the measure.

First assume that v is a vertex with 3 uncolored incident edges. The algorithm branches into two sub-problems **(A)** and **(B)** and we have :

$$T(\mu) \leq T\big(\mu - w_{\tilde{d}(v)} - \sum_{x \in \{v1,v2,v3\}} (w_{\tilde{d}(x)} - w_{\tilde{d}(x)-1})\big) + T\big(\mu - \sum_{x \in \{v,1\}} (w_{\tilde{d}(x)} - w_{\tilde{d}(x)-1})\big)$$

Assume now that we face a collection of vertex-disjoint uncolored cycles of length at least 4. Then, there exists two possible coloring of these cycles and we have

$$T(\mu) \leq 2 \cdot T(\mu - 4 \cdot w_2).$$

Solving these recurrences for all possible values of $\tilde{d}(x)$, and noting that $\mu \leq n$, gives $O(1.8906^n)$ as a running time upper bound. ∎

5 Partition into two trees

Let us recall the definition of TTP: Given a graph $G = (V, E)$, decide whether the edge set E can be partitioned into two sets A and B such that both the edge set A and the edge set B induce a tree in G. Clearly, since A and B induce a tree in G, both sets contain at most $n - 1$ edges. Consequently, the number of edges of G must satisfy $m = |E| \leq 2n - 2$. Otherwise, G cannot have such a partition and is rejected. Thus, we may assume $m \leq 2n - 2$. Thus, a brute force algorithm testing, for every possible partition (A,B) of E, whether both A and B induce a tree, has running time $O^*(2^m) = O^*(4^n)$.

Now, we present an algorithm for Tree-Tree Edge Set Partition that significantly improves upon the running time of $\Theta^*(4^n)$ and relies heavily on the use of matroids. In the first step, the algorithm simply chooses, for each vertex of G, whether it is part of the first tree, the second tree, or both and this in all possible ways; there are 3^n different choices and in each choice we label the vertices of G as follows: If a vertex only belongs to T_1, it is labeled 1, if a vertex only belongs to T_2, it is labeled 2, finally if a vertex belongs to both trees, then it is labeled 3. Hence, if a vertex is labeled 1, then all its incident edges in G belong to T_1. If a vertex is labeled 2, then all its incident edges in G belong to T_2. Finally if a vertex is labeled 3, then in a corresponding and valid edge partition of G it is incident to at least one edge of T_1 and at least one edge of T_2. We claim that it can be checked in polynomial time whether there is an edge partition into two trees T_1 and T_2 for a given $1,2,3$-labeling of G. This would immediately imply that our algorithm has running time $O^*(3^n)$.

Consider a graph $G = (V, E)$ and a $1,2,3$-labeling L. Let V_i, $i = 1,2,3$, be the set of vertices of label i. Assume the edge set of G can be partitioned into two trees T_1 and T_2 respecting the labeling such that for $i = 1,2$, $E(T_i)$ is the edge set of T_i. Hence for $i = 1,2$, $|E(T_i)| = |V_i| + |V_3| - 1$; and consequently $m = |V_1| + |V_2| + 2|V_3| - 2$. If this is not satisfied, the labeling L is not valid and the algorithm rejects this choice respectively labeling immediately.

Now, let us partition the edges of the graph G into three subsets: E_1 is the set of edges which have (at least) an endpoint labeled 1, E_2 is the set of edges which have (at least) an endpoint labeled 2, and E_3 is the set of edges for which both endpoints have label 3. Clearly, all edges of E_1 must belong to T_1 and all edges of E_2 must belong to T_2. If either E_1 or E_2 induces a cycle in G, which is easy to detect, then we reject the labeling L.

To decide whether the edges of E_3 can be partitioned into C_1 and C_2 such that $E_1 \cup C_1$ and $E_2 \cup C_2$ induce a tree in G, we form two graphs G_1 and G_2 from G. G_1 is obtained by deleting all the edges in E_2 and contracting all the edges in E_1. G_2 is formed by deleting all the edges in E_1 and contracting all the edges in E_2. Both G_1 and G_2 have edge sets that are in one-to-one correspondence with E_3, but they may have different vertex sets. Now, $C_1 \subseteq E_3$ must induce a spanning tree of G_1, since G_1 contains no edges of E_2 and decontracting the edges of E_1 in G_1 gives the tree T_1. Similarly, $C_2 \subseteq E_3$ must induce a spanning tree of G_2. This requires that $|E(G_1)| + |E(G_2)| = |V(G_1)| + |V(G_2)| - 2$. If this is not the case, we reject the labeling.

To complete our algorithm, we are using a well-known polynomial-time algorithm for the Matroid Partitioning Problem. For more information on matroids and algorithms on matroids we refer the interested reader to [8]. The matroid partition problem for two matroids $M_1 = (E, \mathcal{I}_1)$ and $M_2 = (E, \mathcal{I}_2)$ asks whether there are independent sets $A_1 \in \mathcal{I}_1$ and $A_2 \in \mathcal{I}_2$ such that A_1 and A_2 is a partition of E. All what remains to solve our problem for a fixed labeling L is to solve a partition problem on two graphic matroids. Recall that the graphic matroid $M = (E, \mathcal{I})$ of a graph $G = (V, E)$ has as ground set the edge set of G and that any subset A of edges of G is an independent set of the matroid, $A \in \mathcal{I}$, if A induces a forest, i.e., an acyclic graph, in G. Let $M_1 = (E_3, \mathcal{I}_1)$ be the graphic matroid of G_1 and let $M_2 = (E_3, \mathcal{I}_2)$ be the graphic matroid of G_2. Now we need to partition E_3 into two sets C_1 and C_2 such that $C_1 \in \mathcal{I}_1$ and $C_2 \in \mathcal{I}_2$. This is the matroid partition problem which has been shown to be polynomial-time solvable by Edmonds [4, 5]. In our case, the only possible partition is into a basis C_1 of M_1 and a basis C_2 of M_2, since C_1 must induce a spanning tree of G_1 and and C_2 must induce a spanning tree of G_2. Consequently, if the matroid partition of M_1 and M_2 is not possible, then we reject the labeling L. Otherwise, the solution C_1 and C_2 of the matroid partition problem can easily be transformed into an edge partition of the graph G into two trees respecting the given labeling.

Finally, we show how to modify and speed up the algorithm. In our algorithm, we need to label the vertices with labels $1, 2$ or 3, and partition the vertex set of the input graph G for some $1,2,3$-labeling into three subsets V_1, V_2 and V_3. This requires that the two trees T_1 and T_2 that we are searching satisfy: the vertices in V_1 belong to T_1 but not to T_2, the vertices of V_2 belong to T_2 but not to T_1, and the vertices of V_3 belong to T_1 and T_2. Consequently, as already pointed out, $m = |V_1| + |V_2| + 2|V_3| - 2$.

We modify the algorithm as follows. If $m \leq 1.5n$, then we simply check each partition of the edge set whether both subsets induce a tree. The running time is $O^*(2^m) = O^*(2^{1.5n})$. Otherwise, $m > 1.5n$. $m = |V_1| + |V_2| + 2|V_3| - 2$ and $|V_1| + |V_2| \leq n$ implies that V_3 has to include at least half of the vertices of G. Consequently, we may restrict to labelings of G in which at least $n/2$ vertices have label 3. How many labelings are this? Suppose we choose $t \leq n/2$ vertices to be labeled 1 or 2 and then we choose for each such vertex whether its label is 1 or 2. Doing this for all $t \leq n/2$, we obtain all labelings that need to be checked and these are at most

$$\sum_{t=0}^{n/2} \binom{n}{t} 2^t \leq n 2^n 2^{n/2}.$$

Since each labeling can be tested in polynomial time, the above described algorithm runs in time $O^*(2^{1.5n})$. Thus, the overall running time of the modified algorithm is $O^*(2^{1.5n})$.

Theorem 5.1. *There is an algorithm solving TTP in time $O(2.8285^n)$.*

6 Conclusions and Final Remarks

We finally observe that, based on a very simple branching algorithm, the problem of partitioning the edge set into a path and a tree PTP can be solved in time $O^*(3^{2n/3}) = O(2.0810^n)$. The whole idea is to start the path by arbitrarily choosing an edge, and then, in each recursive step, to extend the path in every possible way.

Theorem 6.1. *PTP can be solved in $O^*(3^{2n/3})$-time.*

Proof. Let $G = (V, E)$ be a graph. Let us assume that G is not a tree; otherwise, we immediately solve the problem by answering "yes". Also, we assume that $|E| \leq 2n - 2$; otherwise, we immediately answer "no". To solve the problem, our algorithm enumerates all possible paths P in G and then checks whether the remaining edges form a tree.

We use a typical branching approach to enumerate all the possible paths. Let call P the current path that our algorithm is constructing. First it starts by choosing an endpoint u of P and an edge uv of that path. There are at most $n(n-1)$ such edges. Let call uv the *active edge* which is the last edge appended to the current path. Given an active edge uv, the algorithm either decides (i) that P ends at vertex v, or (ii) that P is extended to a new active edge vw, where $w \notin P$ and $vw \in E$. Once it has been decided that two edges uv and vw belong to P, the other edges incident to v have to belong to the tree. We note that, as soon as the algorithm decides that P ends at some vertex v then it checks in polynomial-time whether the edges being not in P induce a tree, and it returns the corresponding answer.

Denoting $T(m)$ the running-time of the algorithm on a graph with m edges, and denoting $d(v)$ the degree of v, the following recurrence described the running time :

$$T(m) = 1 + (d(v) - 1) \cdot T(m - d(v) + 1).$$

A classical analysis show that $T(m) = O^*(3^{m/3}) = O^*(3^{2n/3})$ since $m < 2n$ for any yes-instance.■

Notice that if we partition the vertex set instead of the edge set, the questions tackled in this paper do not make much sense; rather, forbidden graphs should be looked at, as undertaken in [3].

It might be also interesting to consider the question of solving PFES on subcubic graphs. Is that problem still NP-hard? Notice that the possibly related question of finding a feedback vertex set of minimum size is polynomial-time solvable on subcubic graphs, while it is NP-hard in the general case. Furthermore, we did not move into finding exact algorithms for PFES.

References

[1] T. Akiyama, T. Nishizeki, N. Saito. NP-completeness of the Hamiltonian Cycle Problem for bipartite graphs. *Journal of Information Processing*, 3:73–76, 1980.

[2] T. Biedl and F.-J. Brandenburg. Partitions of graphs into trees. *GD 2006*, Springer LNCS, 4372:430–439, 2007.

[3] H. Broersma, F. V. Fomin, J. Kratochvíl, and G. J. Woeginger. Planar graph coloring avoiding monochromatic subgraphs: trees and paths make it difficult. *Algorithmica*, 44(4):343–361, 2006.

[4] J. Edmonds. Minimum partition of a matroid into independent subsets. *J. Res. Nat. Bur. Standards*, 69B, 1965, 67-72.

[5] J. Edmonds. Lehman's switching game and a theorem of Tutte and Nash-Williams. *J. Res. Nat. Bur. Standards*, 69B, 1965, 73-77.

[6] H. N. Gabow and H. H. Westermann. Forests, frames, and games: algorithms for matroid sums and applications. *Algorithmica*, 7:465–497, 1992.

[7] D. A. Holton, B. Manvel, B. D. McKay. Hamiltonian Cycles in cubic 3-connected bipartite planar graphs. *Journal of Combinatorial Theory, Series B*, 38:279–297, 1985.

[8] E. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart & Winston, 1976.

[9] D. Pálvölgyi. Partitionability to two trees is NP-complete. arXiv:1002.3937v1, 2010.

[10] T. J. Schaefer. The complexity of satisfiability problems. *STOC 1978*, pp. 216–226, 1978.