Improved Lower Bounds for Graph Embedding Problems

Hans L. Bodlaender[1,2] and Tom C. van der Zanden[1(✉)]

[1] Department of Computer Science, Utrecht University, Utrecht, The Netherlands
{H.L.Bodlaender,T.C.vanderZanden}@uu.nl

[2] Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract. In this paper, we give new, tight subexponential lower bounds for a number of graph embedding problems. We introduce two related combinatorial problems, which we call String Crafting and Orthogonal Vector crafting, and show that these cannot be solved in time $2^{o(|s|/\log|s|)}$, unless the Exponential Time Hypothesis fails.

These results are used to obtain simplified hardness results for several graph embedding problems, on more restricted graph classes than previously known: assuming the Exponential Time Hypothesis, there do not exist algorithms that run in $2^{o(n/\log n)}$ time for Subgraph Isomorphism on graphs of pathwidth 1, Induced Subgraph Isomorphism on graphs of pathwidth 1, Graph Minor on graphs of pathwidth 1, Induced Graph Minor on graphs of pathwidth 1, Intervalizing 5-Colored Graphs on trees, and finding a tree or path decomposition with width at most c with a minimum number of bags, for any fixed $c \geq 16$.

$2^{\Theta(n/\log n)}$ appears to be the "correct" running time for many packing and embedding problems on restricted graph classes, and we think String Crafting and Orthogonal Vector Crafting form a useful framework for establishing lower bounds of this form.

1 Introduction

Many NP-complete graph problems admit faster algorithms when restricted to planar graphs. In almost all cases, these algorithms have running times that are exponential in a square root function of either the size of the instance n or some parameter k (e.g. $2^{O(\sqrt{n})}$, $n^{O(\sqrt{k})}$ or $2^{O(\sqrt{k})}n^{O(1)}$) and most of these results are tight, assuming the Exponential Time Hypothesis. This seemingly universal behaviour has been dubbed the "Square Root Phenomenon" [1]. The open question [2] of whether the Square Root Phenomenon holds for Subgraph Isomorphism in planar graphs, has recently been answered in the negative: assuming the Exponential Time Hypothesis, there is no $2^{o(n/\log n)}$-time algorithm for Subgraph Isomorphism, even when restricted to (planar) graphs

H.L. Bodlaender—The research of this author was partially supported by the NETWORKS project, funded by the Netherlands Organization for Scientific Research NWO.

D. Fotakis et al. (Eds.): CIAC 2017, LNCS 10236, pp. 92–103, 2017.
DOI: 10.1007/978-3-319-57586-5_9

of pathwidth 2 [3]. The same lower bound holds for Induced Subgraph and (Induced) Minor and is in fact tight: the problems admit $2^{O(n/\log n)}$-time algorithms on H-minor free graphs [3].

The lower bounds in [3] follow by reductions from a problem called String 3-Groups. We introduce a new problem, String Crafting, and establish a $2^{\Omega(|s|/\log|s|)}$-time lower bound under the ETH for this problem by giving a direct reduction from 3-Satisfiability. Using this result, we show that the $2^{\Omega(|s|/\log|s|)}$-time lower bounds for (Induced) Subgraph and (Induced) Minor hold even on graphs of pathwidth 1.

Alongside String Crafting, we introduce the related Orthogonal Vector Crafting problem. Using this problem, we show $2^{\Omega(|n|/\log|n|)}$-time lower bounds for deciding whether a 5-coloured tree is the subgraph of an interval graph (for which the same colouring is proper) and for deciding whether a graph admits a tree (or path) decomposition of width 16 with at most a given number of bags.

For any fixed k, Intervalizing k-Coloured Graphs can be solved in time $2^{O(n/\log n)}$ [4]. Bodlaender and Nederlof [5] conjecture a lower bound (under the Exponential Time Hypothesis) of $2^{\Omega(n/\log n)}$ time for $k \geq 6$; we settle this conjecture and show that it in fact holds for $k \geq 5$, even when restricted to trees. To complement this result for a fixed number of colours, we also show that there is no algorithm solving Intervalizing Coloured Graphs (with an arbitrary number of colours) in time $2^{o(n)}$, even when restricted to trees.

The minimum size path and tree decomposition problems can also be solved in $2^{O(n/\log n)}$ time on graphs of bounded treewidth. This is known to be tight under the Exponential Time Hypothesis for $k \geq 39$ [5]. We improve this to $k \geq 16$; our proof is also simpler than that in [5].

Our results show that String Crafting and Orthogonal Vector Crafting are a useful framework for establishing lower bounds of the form $2^{\Omega(n/\log n)}$ under the Exponential Time Hypothesis. It appears that for many packing and embedding problems on restricted graph classes, this bound is tight.

For some omitted proofs, we refer to the full version of this paper [6].

2 Preliminaries

Strings. We work with the alphabet $\{0,1\}$; i.e., strings are elements of $\{0,1\}^*$. The length of a string s is denoted by $|s|$. The i^{th} character of a string s is denoted by $s(i)$. Given a string $s \in \{0,1\}^*$, $\overline{s}$ denotes binary complement of s, that is, each occurrence of a 0 is replaced by a 1 and vice versa; i.e., $|s| = |\overline{s}|$, and for $1 \leq i \leq |s|$, $\overline{s}(i) = 1 - s(i)$. E.g., if $s = 100$, then $\overline{s} = 011$. With s^R, we denote the string s in reverse order; e.g., if $s = 100$, then $s^R = 001$. The concatenation of strings s and t is denoted by $s \cdot t$. A string s is a *palindrome*, when $s = s^R$. By 0^n (resp. 1^n) we denote the string that consists of n 0's (resp. 1's).

Graphs. Given a graph G, we let $V(G)$ denote its vertex set and $E(G)$ its edge set. Let $Nb(v)$ denote the open neighbourhood of v, that is, the vertices adjacent to v, excluding v itself. We assume all graphs are simple.

Treewidth and Pathwidth. A *tree decomposition* of a graph $G = (V, E)$ is a tree T with vertices $t_1, \ldots, t_s$ with for each vertex t_i a *bag* $X_i \subseteq V$ such that for all $v \in V$, the set $\{t_i \in \{t_1, \ldots, t_s\} \mid v \in X_i\}$ is non-empty and induces a connected subtree of T and for all $(u, v) \in E$ there exists a bag X_i such that $\{u, v\} \in X_i$. The *width* of a tree decomposition is $\max_i\{|X_i| - 1\}$ and the *treewidth* of a graph G is the minimum width of a tree decomposition of G. A *path decomposition* is a tree decomposition where T is a path, and the pathwidth of a graph G is the minimum width of a path decomposition of G.

A graph is a *caterpillar tree* if it is connected and has pathwidth 1.

Subgraphs and Isomorphism. H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$; we say the subgraph is *induced* if moreover $E(H) = E(G) \cap \{\{u, v\} \mid u, v \in V(H)\}$. We say a graph H is *isomorphic* to a graph G if there is a bijection $f : V(H) \to V(G)$ so that $(u, v) \in E(H) \iff (f(u), f(v)) \in E(G)$.

Contractions, Minors. We say a graph G' is obtained from G by *contracting* edge (u, v), if G' is obtained from G by replacing vertices u, v with a new vertex w which is made adjacent to all vertices in $Nb(u) \cup Nb(v)$. A graph G' is a minor of G if a graph isomorphic to G' can be obtained from G by contractions and deleting vertices and/or edges. G' is an *induced minor* if we can obtain it by contractions and deleting vertices (but not edges).

We say G' is an *r-shallow minor* if G' can be obtained as a minor of G and any subgraph of G that is contracted to form some vertex of G' has radius at most r (that is, there is a central vertex within distance at most r from any other vertex in the subgraph). Finally, G' is a *topological minor* if we can subdivide the edges of G' to obtain a graph G'' that is isomorphic to a subgraph of G (that is, we may repeatedly take an edge (u, v) and replace it by a new vertex w and edges (u, w) and (w, v)).

For each of (induced) subgraph, induced (minor), topological minor and shallow minor, we define the corresponding decision problem, that is, to decide whether a pattern graph P is isomorphic to an (induced) subgraph/(induced) minor/topological minor/shallow minor of a host graph G.

3 String Crafting and Orthogonal Vector Crafting

We now formally introduce the String Crafting problem:

String Crafting
Given: String s, and n strings $t_1, \ldots, t_n$, with $|s| = \sum_{i=1}^{n} |t_i|$.
Question: Is there a permutation $\Pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, such that the string $t^{\Pi} = t_{\Pi(1)} \cdot t_{\Pi(2)} \cdots t_{\Pi(n)}$ fulfils that for each i, $1 \le i \le |s|$, $s(i) \ge t^{\Pi}(i)$.

i.e., we permute the collection of strings $\{t_1, t_2, \ldots t_n\}$, then concatenate these, and should obtain a resulting string t^{Π} (that necessarily has the same length as s) such that on every position where t^{Π} has a 1, s also has a 1.

We also introduce the following variation of String Crafting, where, instead of requiring that whenever t^{Π} has a 1, s has a 1 as well, we require that whenever t^{Π} has a 1, s has a 0 (i.e. the strings t^{Π} and s, viewed as vectors over the reals, are orthogonal). These problems are closely related, and have the same complexity. However, sometimes one problem will be more convenient as a starting point for a reduction than the other.

Orthogonal Vector Crafting
Given: String s, and n strings $t_1, \ldots, t_n$, with $|s| = \sum_{i=1}^{n} |t_i|$.
Question: Is there a permutation $\Pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, such that the string $t^{\Pi} = t_{\Pi(1)} \cdot t_{\Pi(2)} \cdots t_{\Pi(n)}$ fulfils that for each i, $1 \le i \le |s|$, $s(i) \cdot t^{\Pi}(i) = 0$, i.e., when viewed as vectors, s is orthogonal to t^{Π}.

Theorem 1. *Suppose the Exponential Time Hypothesis holds. Then there is no algorithm that solves the* String Crafting *problem in* $2^{o(|s|/\log|s|)}$ *time, even when all strings* t_i *are palindromes and start and end with a* 1.

Proof. Suppose we have an instance of 3-Satisfiability with n variables and m clauses. We number the variables x_1 to x_n and for convenience, we number the clauses C_{n+1} to C_{n+m+1}.

We assume by the sparsification lemma that $m = O(n)$ [7, Corollary 2].

Let $q = \lceil \log(n+m) \rceil$, and let $r = 4q + 2$. We first assign an r-bit number to each variable and clause; more precisely, we give a mapping $id : \{1, \ldots, n+m\} \to \{0,1\}^r$. Let $nb(i)$ be the q-bit binary representation of i. We set, for $1 \le i \le n+m$:

$$id(i) = 1 \cdot nb(i) \cdot \overline{nb(i)} \cdot \overline{nb(i)}^R \cdot nb(i)^R \cdot 1$$

Note that each $id(i)$ is an r-bit string that is a palindrome, ending and starting with a 1.

We first build s, by taking the concatenation of n strings, one for each variable.

Suppose the literal x_i appears c_i times in a clause, and the literal $\neg x_i$ appears d_i times in a clause. Set $f_i = c_i + d_i$. Assign the following strings to the pair of literals x_i and $\neg x_i$:

– a^{x_i} is the concatenation of the id's of all clauses in which x_i appears, followed by d_i copies of the string $1 \cdot 0^{r-2} \cdot 1$.
– $a^{\neg x_i}$ is the concatenation of the id's of all clauses in which $\neg x_i$ appears, followed by c_i copies of the string $1 \cdot 0^{r-2} \cdot 1$.
– $b^i = id(i) \cdot a^{x_i} \cdot id(i) \cdot a^{\neg x_i} \cdot id(i)$.

Now, we set $s = b^1 \cdot b^2 \cdots b^{n-1} \cdot b^n$.

We now build the collection of strings t_i. We have three different types of strings:

– *Variable selection:* For each variable x_i we have one string of length $(f_i + 2)r$ of the form $id(i) \cdot 0^{r \cdot f_i} \cdot id(i)$.
– *Clause verification:* For each clause C_i, we have a string of the form $id(i)$.

– *Filler strings:* A filler string is of the form $1 \cdot 0^{r-2} \cdot 1$. We have $n + 2m$ filler strings.

Thus, the collection of strings t_i consists of n variable selection strings (of total length $(3m + 2n)r$), m clause verification strings (of length r), and $n + 2m$ filler strings (also of length r). Notice that each of these strings is a palindrome and ends and starts with a 1.

The idea behind the reduction is that s consists of a list of variable identifiers followed by which clauses a true/false assignment to that variable would satisfy. The variable selection gadget can be placed in s in two ways: either covering all the clauses satisfied by assigning true to the variable, or covering all the clauses satisfied by assigning false. The clause verification strings then fit into s only if we have not covered all of the places where the clause can fit with variable selection strings (corresponding to that we have made some assignment that satisfies the clause).

Furthermore, note that since $\Sigma_{i=1}^{n} f_i = 3m$, the length of s is $(3n + 6m)r$, the combined length of the variable selection strings is $(2n + 3m)r$, the combined length of the clause verification strings is mr, and the filler strings have combined length $(n + 2m)r$.

In the following, we say a string t_i is mapped to a substring s' of s if s' is the substring of s corresponding to the position (and length) of t_i in t^{Π}.

Lemma 1. *The instance of* 3-Satisfiability *is satisfiable, if and only if the constructed instance of* String Crafting *has a solution.*

Proof. First, we show the reverse implication. Suppose we have a satisfying assignment to the 3-Satisfiability instance. Consider the substring of s formed by b^i, which is of the form $id(i) \cdot a^{x_i} \cdot id(i) \cdot a^{\neg x_i} \cdot id(i)$. If in the satisfying assignment x_i is true, we choose the permutation Π so that variable selection string $id(i) \cdot 0^{r \cdot f_i} \cdot id(i)$ corresponding to x_i is mapped to the substring $id(i) \cdot a^{\neg x_i} \cdot id(i)$; if x_i is false, we map the variable selection string onto the substring $id(i) \cdot a^{x_i} \cdot id(i)$. A filler string is mapped to the other instance of $id(i)$ in the substring.

Now, we show how the clause verification strings can be mapped. Suppose clause C_j is satisfied by a literal x_i (resp. $\neg x_i$). Since x_i is true (resp. false), the substring a^{x_i} (resp. $a^{\neg x_i}$) of s is not yet used by a variable selection gadget and contains $id(j)$ as a substring, to which we can map the clause verification string corresponding to C_j.

Note that in s now remain a number of strings of the form $1 \cdot 0^{r-2} \cdot 1$ and a number of strings corresponding to id's of clauses, together $2m$ such strings, which is exactly the number of filler strings we have left. These can thus be mapped to these strings, and we obtain a solution to the String Crafting instance. It is easy to see that with this construction, s has a 1 whenever the string constructed from the permutation does.

Next, for the forward implication, consider a solution Π to the String Crafting instance. We require the following lemma:

Lemma 2. *Suppose that $t_i = id(j)$. Then the substring w of s corresponding to the position of t_i in t^{Π} is $id(j)$.*

Proof. Because the length of each string is a multiple of r, w is either $id(k)$ for some k, or the string $1 \cdot 0^{r-2} \cdot 1$. Clearly, w can not be $1 \cdot 0^{r-2} \cdot 1$ because the construction of $id(i)$ ensures that it has more than 2 non-zero characters, so at some position w would have a 1 where w' does not. Recall that $id(i) = 1 \cdot nb(i) \cdot \overline{nb(i)} \cdot \overline{nb(i)}^R \cdot nb(i)^R \cdot 1$. If $j \neq k$, then either at some position $nb(k)$ has a 0 where $nb(j)$ has a 1 (contradicting that Π is a solution) or at some position $\overline{nb(k)}$ has a 0 where $\overline{nb(j)}$ has a 1 (again contradicting that Π is a solution). Therefore $j = k$. □

Clearly, for any i, there are only two possible places in t^{Π} where the variable selection string $id(i) \cdot 0^{r \cdot f_i} \cdot id(i)$ can be mapped to: either in the place of $id(i) \cdot a^{x_i} \cdot id(i)$ in s or in the place of $id(i) \cdot a^{\neg x_i} \cdot id(i)$, since these are the only (integer multiple of r) positions where $id(i)$ occurs in s. If the former place is used we set x_i to false, otherwise we set x_i to true.

Now, consider a clause C_j, and the place where the corresponding clause verification gadget $id(j)$ is mapped to. Suppose it is mapped to some substring of $id(i) \cdot a^{x_i} \cdot id(i) \cdot a^{\neg x_i} \cdot id(i)$. If $id(j)$ is mapped to a substring of a^{x_i} then (by construction of a^{x_i}) x_i appears as a positive literal in C_j and our chosen assignment satisfies C_j (since we have set x_i to true). Otherwise, if $id(j)$ is mapped to a substring of $a^{\neg x_i}$ x_i appears negated in C_j and our chosen assignment satisfies C_j (since we have set x_i to false).

We thus obtain a satisfying assignment for the 3-Satisfiability instance. □

Since in the constructed instance, $|s| = (3n + 6m)r$ and $r = O(\log n), m = O(n)$, we have that $|s| = O(n \log n)$. A $2^{o(|s|/\log |s|)}$-time algorithm for String Crafting would give a $2^{o(n \log n/\log (n \log n))} = 2^{o(n)}$-time algorithm for deciding 3-Satisfiability, violating the ETH. □

Note that we can also restrict all strings t_i to start and end with a 0 by a slight modification of the proof.

Theorem 2. *Assuming the Exponential Time Hypothesis,* Orthogonal Vector Crafting *can not be solved in* $2^{o(|s|/\log |s|)}$ *time, even when all strings* t_i *are palindromes and start and end with a* 1.

Proof. This follows from the result for String Crafting, by taking the complement of the string s. □

Again, we can also restrict all strings t_i to start and end with a 0.

As illustrated by the following theorem, these lower bounds are tight. The algorithm is a simpler example of the techniques used in [3–5].

Theorem 3. *There exists algorithms, solving* String Crafting *and* Orthogonal Vector Crafting *in* $2^{O(|s|/\log |s|)}$.

4 Lower Bounds for Graph Embedding Problems

Theorem 4. *Suppose the Exponential Time Hypothesis holds. Then there is no algorithm solving* Subgraph Isomorphism *in* $2^{o(n/\log n)}$ *time, even if* G *is a caterpillar tree of maximum degree* 3 *or* G *is connected, planar, has pathwidth* 2 *and has only one vertex of degree greater than* 3 *and* P *is a tree.*

Proof. By reduction from String Crafting. We first give the proof for the case that G is a caterpillar tree of maximum degree 3, We construct G from s as follows: we take a path of vertices $v_1, \ldots, v_{|s|}$ (*path vertices*). If $s(i) = 1$, we add a *hair vertex* h_i and edge (v_i, h_i) to G (obtaining a caterpillar tree). We construct P from the strings t_i by, for each string t_i repeating this construction, and taking the disjoint union of the caterpillars created in this way (resulting in a graph that is a forest of caterpillar trees, i.e., a graph of pathwidth 1). An example of this construction is depicted in Fig. 1. The constructed instance of G contains P as a subgraph, if and only if the instance of String Crafting has a solution: the order in which the caterpillars are embedded in G gives the permutation of the strings: when a caterpillar in P has a hair, G must have a hair at the specific position, which implies that a position with a 1 in the constructed string t must be a position where s also has a 1.

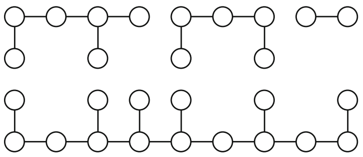


Fig. 1. Simplified example of the graphs created in the hardness reduction for Theorem 4. The bottom caterpillar represents the host graph (corresponding to string s), the top caterpillars represent the strings t_i and form the guest graph. Here $s = 101110101$ and $t_1 = 1010, t_2 = 101$ and $t_3 = 00$.

Since the constructed instance has $O(|s|)$ vertices, this establishes the first part of the lemma. For the case that G is connected, we add to the graph G constructed in the first part of the proof a vertex u and, for each path vertex v_i, an edge (v_i, u). To P we add a vertex u' that has an edge to some path vertex of each component. By virtue of their high degrees, u must be mapped to u' and the remainder of the reduction proceeds in the same way as in the first part of the proof. □

This proof can be adapted to show hardness for a number of other problems:

Theorem 5. *Suppose the Exponential Time Hypothesis holds. Then there is no algorithm solving* Induced Subgraph, (Induced) Minor, Shallor Minor *or* Topological Minor *in* $2^{o(n/\log n)}$ *time, even if* G *is a caterpillar tree of maximum degree* 3 *or* G *is connected, planar, has pathwidth* 2 *and has only one vertex of degree greater than* 3 *and* P *is a tree.*

5 Tree and Path Decompositions with Few Bags

In this section, we study the following problem, and its analogously defined variant Minimum Size Path Decomposition (k-MSPD). Theorem 6 is an improvement over Theorem 3 of [5], where the same was shown for $k \geq 39$; our proof is also simpler.

Minimum Size Tree Decomposition of width k (k-MSTD)
Given: A graph G, integer b.
Question: Does G have a tree decomposition of width at most k, that has at most b bags?

Theorem 6. *Let $k \geq 16$. Suppose the Exponential Time Hypothesis holds, then there is no algorithm for k-MSPD or k-MSTD using $2^{o(n/\log n)}$ time.*

Proof. By reduction from Orthogonal Vector Crafting. We begin by showing the case for MSPD, but note the same reduction is used for MSTD.

For the string s, we create a connected component in the graph G as follows: for $1 \leq i \leq |s|+1$ we create a clique C_i of size 6, and (for $1 \leq i \leq |s|$) make all vertices of C_i adjacent to all vertices of C_{i+1}. For $1 \leq i \leq |s|$, if $s(i) = 1$, we create a vertex s_i and make it adjacent to the vertices of C_i and C_{i+1}.

For each string t_i, we create a component in the same way as for s, but rather than using cliques of size 6, we use cliques of size 2: for each $1 \leq i \leq n$ and $1 \leq j \leq |t_i|+1$ create a clique $T_{i,j}$ of size 2 and (for $1 \leq j \leq |t_i|$) make all vertices of $T_{i,j}$ adjacent to all vertices of $T_{i,j+1}$. For $1 \leq j \leq |t_i|$, if $t_i(j) = 1$, create a vertex $t_{i,j}$ and make it adjacent to the vertices of $T_{i,j}$ and $T_{i,j+1}$.

An example of the construction (for $s = 10110$ and $t_1 = 01001$) is shown in Fig. 2. We now ask whether a path decomposition of width 16 exists with at most $|s|$ bags. Due to space constraints, we omit the correctness proof of the construction. □

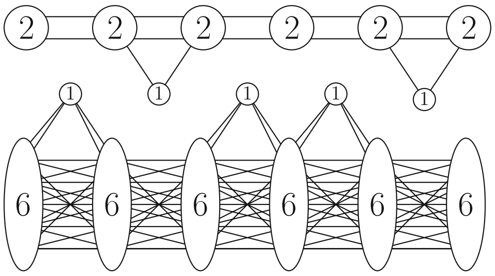


Fig. 2. Simplified example of the graph created in the hardness reduction for Theorem 6. The circles and ellipses represent cliques of various sizes. The component depicted in the top of the picture corresponds to $t_1 = 01001$, while the component at the bottom corresponds to $s = 10110$.

6 Intervalizing Coloured Graphs

In this section, we consider the following problem:

> Intervalizing Coloured Graphs
> **Given:** A graph $G = (V, E)$ together with a proper colouring $c : V \to \{1, 2, \dots, k\}$.
> **Question:** Is there an interval graph G' on the vertex set V, for which c is a proper colouring, and which is a supergraph of G?

Intervalizing Coloured Graphs is known to be NP-complete, even for 4-coloured caterpillars (with hairs of unbounded length) [8]. In contrast with this result we require five colours instead of four, and the result only holds for trees instead of caterpillars. However, we obtain a $2^{\Omega(n/\log n)}$ lower bound under the Exponential Time Hypothesis, whereas the reduction in [8] is from Multiprocessor Scheduling and to our knowledge, the best lower bound obtained from it is $2^{\Omega(\sqrt[5]{n})}$ (the reduction is weakly polynomial in the length of the jobs, which is $\Theta(n^4)$, as following from the reduction from 3-Partition in [9]). In contrast to these hardness results, for the case with 3 colours there is an $O(n^2)$ time algorithm [10,11].

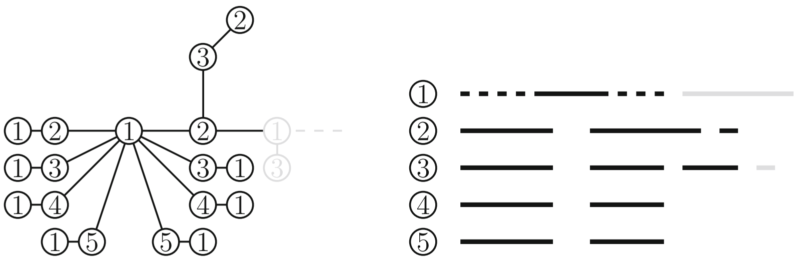
Theorem 7. Intervalizing Coloured Graphs *does not admit a* $2^{o(n/\log n)}$*-time algorithm, even for* 5*-coloured trees, unless the Exponential Time Hypothesis fails.*

Proof. Let $s, t_1, \dots, t_n$ be an instance of Orthogonal Vector Crafting. We construct $G = (V, E)$ in the following way:

S-String Path. We create a path of length $2|s| - 1$ with vertices $p_0, \dots p_{2|s|-2}$, and set $c(p_i) = 1$ if i is even and $c(p_i) = 2$ if i is odd. Furthermore, for even $0 \leq i \leq 2|s| - 2$, we create a neighbour n_i with $c(n_i) = 3$.

Barriers. To each endpoint of the path, we attach the *barrier gadget*, depicted in Fig. 3. The gray vertices are not part of the barrier gadget itself, and represent p_0 and n_0 (resp. $p_{2|s|-2}$ and $n_{2|s|-2}$). Note that the barrier gadget operates on similar principles as the barrier gadget due to Alvarez et al. [8]. We shall refer to the barrier attached to p_0 as the left barrier, and to the barrier attached to $p_{2|s|-2}$ as the right barrier.

The barrier consists of a central vertex with colour 1, to which we connect eight neighbours (*clique vertices*), two of each of the four remaining colours. Each of the clique vertices is given a neighbour with colour 1. To one of the clique vertices with colour 2 we connect a vertex with colour 3, to which a vertex with colour 2 is connected (*blocking vertices*). This clique vertex shall be the barrier's *endpoint.* Note that the neighbour with colour 1 of this vertex is not considered part of the barrier gadget, as it is instead a path vertex. We let C_l (e_l) denote the center (endpoint) of the left barrier, and C_r (e_r) the center (endpoint) of the right barrier.



(a) Barrier Gadget (b) Interval Representation

Fig. 3. (a) Barrier Gadget. The gray vertices are not part of the barrier gadget itself, and show how it connects to the rest of the graph. (b) How the barrier gadget may (must) be intervalized.

T-String Paths. Now, for each string t_i, we create a path of length $2|t_i|+1$ with vertices $q_{i,0},\ldots,q_{i,2|t_i|}$ and set $c(q_{i,j})=3$ if j is odd and set $c(q_{i,j})=2$ if j is even. We make $q_{i,1}$ adjacent to U. Furthermore, for odd $1\le j\le 2|t_i|-1$, we create a neighbour m_j with $c(m_j)=1$. We also create two *endpoint vertices* of colour 3, one of which is adjacent to $q_{i,0}$ and the other to $q_{i,2|t_i|}$,

Connector Vertex. Next, we create a *connector vertex* of colour 5, which is made adjacent to p_1 and to $q_{i,1}$ for all $1\le i\le n$. This vertex serves to make the entire graph connected.

Marking Vertices. Finally, for each $1\le i\le |s|$ (resp. for each $1\le i\le n$ and $1\le j\le |t_i|$), if $s(i)=1$ (resp. $t_i(j)=1$), we give p_{2i-1} (resp. $q_{i,2j-1}$) two neighbours (called the *marking vertices*) with colour 4. For each of the marking vertices, we create a neighbour with colour 3.

This construction is depicted in Fig. 4. In this example $s=10100$, $t_1=01$ and $t_2=001$. Note that this instance of Orthogonal Vector Crafting is illustrative, and does not satisfy the restrictions required in the proof.

Informally, the construction works as follows: the barriers at the end of the path of p-vertices can not be passed by the remaining vertices, meaning we have to "weave" the shorter q-paths into the long p-path. The colours enforce that the paths are in "lockstep", that is, we have to traverse them at the same speed. We have to map every q-vertex with colour 3 to a p-vertex with colour 2, but the marking vertices prevent us from doing so if both bitstrings have a 1 at that particular position.

Due to space constraints, we omit the correctness proof of the construction.

The number of vertices of G is linear in $|s|$, and we thus obtain a $2^{o(n/\log n)}$ lower bound under the Exponential Time Hypothesis. □

Note that the graph created in this reduction only has one vertex of super-constant degree. This is tight, since the problem is polynomial-time solvable for bounded degree graphs (for any fixed number of colours) [12].

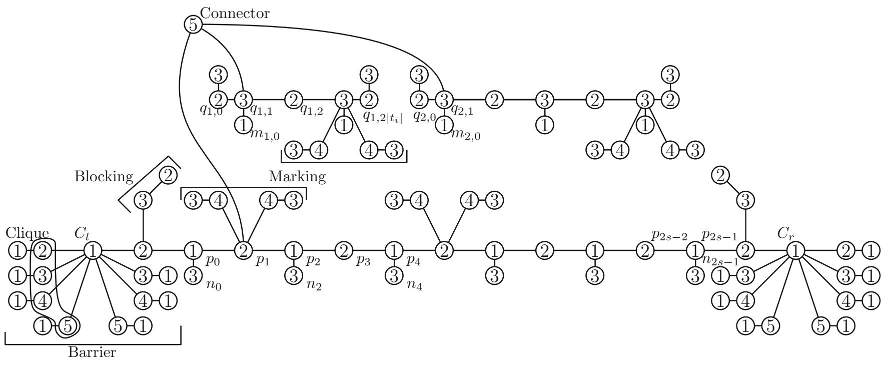


Fig. 4. Example of the graph created in the hardness reduction for Theorem 7.

To complement this result for a bounded number of colours, we also show a $2^{\Omega(n)}$-time lower bound for graphs with an unbounded number of colors, assuming the ETH. Note that this result implies that the algorithm from [4] is optimal.

Theorem 8. *Assuming the Exponential Time Hypothesis, there is no algorithm solving* Intervalizing Coloured Graphs *in time* $2^{o(n)}$*, even when restricted to trees.*

7 Conclusions

In this paper, we have shown for several problems that, under the Exponential Time Hypothesis, $2^{\Theta(n/\log n)}$ is the best achievable running time - even when the instances are very restricted (for example in terms of pathwidth or planarity). For each of these problems, algorithms that match this lower bound are known and thus $2^{\Theta(n/\log n)}$ is (likely) the asymptotically optimal running time.

For problems where planarity or bounded treewidth of the instances (or, through bidimensionality, of the solutions) can be exploited, the optimal running time is often $2^{\Theta(\sqrt{n})}$ (or features the square root in some other way). On the other hand, each of problems studied in this paper exhibits some kind of "packing" or "embedding" behaviour. For such problems, $2^{\Theta(n/\log n)}$ is often the optimal running time. We have introduced two artificial problems, String Crafting and Orthogonal Vector Crafting, that form a useful framework for proving such lower bounds.

It would be interesting to study which other problems exhibit such behaviour, or to find yet other types of running times that are "natural" under the Exponential Time Hypothesis. The loss of the $\log n$-factor in the exponent is due to the fact that $\log n$ bits or vertices are needed to "encode" n distinct elements; it would be interesting to see if there are any problems or graph classes where a more compact encoding is possible (for instance only $\log^{1-\epsilon} n$ vertices required, leading to a tighter lower bound) or where an encoding is less compact (for instance $\log^2 n$ vertices required, leading to a weaker lower bound) and whether this can be exploited algorithmically.

Acknowledgement. We thank Jesper Nederlof for helpful comments and discussions.

References

1. Marx, D.: The square root phenomenon in planar graphs. In: Fellows, M., Tan, X., Zhu, B. (eds.) AAIM/FAW -2013. LNCS, vol. 7924, p. 1. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38756-2_1
2. Marx, D.: What's next? Future directions in parameterized complexity. In: Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D. (eds.) The Multivariate Algorithmic Revolution and Beyond. LNCS, vol. 7370, pp. 469–496. Springer, Heidelberg (2012). doi:10.1007/978-3-642-30891-8_20
3. Bodlaender, H.L., Nederlof, J., van der Zanden, T.C.: Subexponential time algorithms for embedding H-minor free graphs. In: Chatzigiannakis, I., Mitzenmacher, M., Rabani, Y., Sangiorgi, D. (eds.) 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016), vol. 55, Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, pp. 9: 1–9: 14 (2016)
4. Bodlaender, H.L., van Rooij, J.M.M.: Exact algorithms for Intervalizing Coloured Graphs. Theor. Comput. Syst. **58**(2), 273–286 (2016)
5. Bodlaender, H.L., Nederlof, J.: Subexponential time algorithms for finding small tree and path decompositions. In: Bansal, N., Finocchi, I. (eds.) ESA 2015. LNCS, vol. 9294, pp. 179–190. Springer, Heidelberg (2015). doi:10.1007/978-3-662-48350-3_16
6. Bodlaender, H.L., van der Zanden, T.C.: Improved lower bounds for graph embedding problems. arXiv preprint (2016). arXiv:1610.09130
7. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. **63**(4), 512–530 (2001)
8. Àlvarez, C., Diáz, J., Serna, M.: The hardness of intervalizing four colored caterpillars. Discret. Math. **235**(1), 19–27 (2001)
9. Jansen, K., Land, F., Land, K.: Bounding the running time of algorithms for scheduling and packing problems. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 439–450. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40104-6_38
10. Bodlaender, H.L., de Fluiter, B.: Intervalizing k-colored graphs. Technical report UU-CS-1995-15, Department of Information and Computing Sciences, Utrecht University (1995)
11. Bodlaender, H.L., de Fluiter, B.: On intervalizing k-colored graphs for DNA physical mapping. Discret. Appl. Math. **71**(1), 55–77 (1996)
12. Kaplan, H., Shamir, R.: Bounded degree interval sandwich problems. Algorithmica **24**(2), 96–104 (1999)