Hardness results for approximating the bandwidth [*]

Chandan Dubey [†], Uriel Feige [‡], Walter Unger[§]

June 9, 2009

Abstract

The bandwidth of an n-vertex graph G is the minimum value b such that the vertices of G can be mapped to distinct integer points on a line without any edge being stretched to a distance more than b. Previous to the work reported here, it was known that it is NP-hard to approximate the bandwidth within a factor better than 3/2. We improve over this result in several respects. For certain classes of graphs (such as cycles of cliques) for which it is easy to approximate the bandwidth within a factor of 2, we show that approximating the bandwidth within a ratio better than 2 is NP-hard. For caterpillars (trees in which all vertices of degree larger than two lie on one path) we show that it is NP-hard to approximate the bandwidth within any constant, and that an approximation ratio of $c\sqrt{\log n/\log\log n}$ will imply a quasi-polynomial time algorithm for NP (when c is a sufficiently small constant).

1 Introduction

Let $G = (V, E)$ be an undirected graph on n vertices, which without loss of generality in our context can be assumed to be connected. A linear arrangement (a.k.a. layout, or numbering) of G is a one to one mapping $f : V \to \{1, \ldots, n\}$. The bandwidth of the layout f, $b_G(f)$ is defined as

$$b_G(f) = \max_{(u,v)\in E} |f(u) - f(v)|$$

The bandwidth of a graph, denoted as b_G or simply b, when G is clear from the context is

$$b_G = \min_f b_G(f)$$

The bandwidth problem is NP-hard, and in this paper we present new hardness of approximation results for it. To aid in the discussion of previous results and our results, we present a few classes of graphs.

Definition 1 *A caterpillar is a tree composed of a path called the backbone and other paths connected to the backbone called strands (a.k.a. hairs). In other words, a caterpillar is a tree on which all vertices of degree greater than 2 lie on a single path.*

For caterpillars, it is natural to consider linear arrangements that are monotone with respect to the backbone vertices. The minimum bandwidth among all such linear arrangements is referred to as the *unfolded bandwidth*. However, the bandwidth problem per-se does not require the linear

[*] A preliminary version of this work appeared in [32]

[†] `cdubey@gmail.com`

[‡] The Weizmann Institute. `uriel.feige@weizmann.ac.il`. Supported in part by The Israel Science Foundation (grant No. 873/08).

[§] RWTH Aachen, Lehrstuhl für Informatik I, D-52056 Aachen, `quax@cs.rwth-aachen.de`. Partially supported by the EU within the 6th Framework Programme under contract 001907 (DELIS).

arrangement to be monotone with respect to the backbone vertices. We say that a linear arrangement is *folded* at backbone vertex v, if the two neighbors of v on the backbone both lie to the left of v (that is, are mapped to smaller numbers than v is), or both lie to the right of v (are mapped to larger numbers than v is).

Definition 2 *A* clique cycle graph *is a graph composed of a set of cliques, where each clique has a special vertex, and the special vertices are connected as a cycle.*

Definition 3 *A* ringed caterpillar *is a caterpillar with the endpoints of the backbone joined by an edge.*

Definition 4 *A* circular arc graph *is a graph whose vertices are intervals (a.k.a. arcs) on a circle and there is an edge between two vertices if the corresponding intervals intersect. A* unit *(or* uniform*) circular arc graph is one in which all intervals are of the same length (and in particular, no interval strictly contains another interval).*

Definition 5 *A δ-*dense *graph is a graph in which the degree of every vertex is at least δn.*

Definition 6 *An* asteroidal triple *in a graph is a set of three distinct vertices such that removing any one of them and all its neighbors leaves the other two in the same connected component. (Equivalently, for any two of them there is a path connecting them and not passing through the neighborhood of the other vertex.) An* asteroidal triple free graph *or* AT-free *graph for short is a graph with no asteroidal triples.*

Observe for example that a caterpillar with strand length 1 is an AT-free graph.

The local density of a graph is an easily computable lower bound on the bandwidth of a graph.

Definition 7 *For a vertex v in a graph and a positive integer r, let $N_r(v)$ denote the set of vertices at distance at most r from v. Then the* local density *of a graph G is*

$$D_G = \max_{v,r} \frac{|N_r(v)|}{2r}$$

It can easily be seen that for every graph $b_G \geq D_G$.

1.1 Related work

Before discussing related work, let us clarify the historical perspective. A preliminary version of the current manuscript [32] appeared already in 1998. A combination of the level of complication of the proofs and space limitations imposed on the preliminary version led to the situation that the most interesting claims in the preliminary version appeared without proofs. The current version attempts to remedy this situation by presenting the full proofs. The proofs as written in the current version are based on ideas similar to those discussed in the preliminary version, but the details and the notation are different. Some of the results presented in the current version (the growth rate of the hardness ratio as a function of n) were not stated in the preliminary version. Some of the related work that will be surveyed (most notably [11]) appeared in between the preliminary version and the current version of this work. Hence the historic order among results is not clear-cut, but this does not lead to confusion regarding the contribution of the work reported here compared to work of others.

For general graphs, the following results are known. There are graphs (even trees) for which the local density is not a tight lower bound on the bandwidth, in the sense that $b_G \geq \Omega(D_G \log n)$. On the other hand, there are relatively simple polynomial time (randomized) algorithms that compute a linear arrangement of bandwidth at most $O(D_G(\log n)^{7/2})$ [8, 23], and $O(D_G(\log n)^{5/2})$ for trees [14]. A somewhat better approximation ratio (but against the actual bandwidth rather than the local density) can be obtained using mathematical programming, namely $O(b_G(\log n)^3(\log\log n)^{1/4})$ [7, 24]. The analysis of all these algorithms is based on the notion of *volume respecting embeddings* that was introduced in [8]. On the negative side, Blache et. al. [2] showed that it is hard to approximate the bandwidth better than a factor of $\frac{4}{3} - \epsilon$ on trees and $\frac{3}{2} - \epsilon$ on general graphs (in both cases, $\epsilon > 0$ can be taken to be an arbitrarily small constant).

The bandwidth of caterpillars received a lot of attention. For caterpillars of strand length at most 2, the bandwidth problem can be solved in polynomial time [1]. Monien [27] showed that bandwidth of caterpillars of strand length at most 3 is NP-complete. The reduction crucially uses gadgets which forces the backbone to fold. In [4] it was shown that there are caterpillars with $b_G \geq \Omega(D_G \log n/\log\log n)$. Feige and Talwar [11] showed that the unfolded bandwidth of caterpillars can be approximated within a constant factor, that the gap between folded and unfolded bandwidth of caterpillars is at most $O(\log n/\log\log n)$, and that for some caterpillars this gap is indeed as large as $\Omega(\log n/\log\log n)$. They also showed an algorithm that provides an unfolded linear arrangement of bandwidth at most $O(D_G \log n/\log\log n)$. The combination of these results implies that there is a polynomial time algorithm that approximates the bandwidth of caterpillars within a ratio of $O(\log n/\log\log n)$, and that further improvement would require both folding of caterpillars and comparing against a lower bound which is tighter than the local density lower bound.

Other classes of graphs that are mentioned in this paper also received attention in the past. In [21] it is shown that the bandwidth of asteroidal-triple-free graphs can be approximated within a factor of 2. A factor 2 approximation for circular arc graphs and some other related graphs appears in [22]. A 3-approximation algorithm for δ-dense graphs appears in [17] .

There are many other results known on the bandwidth, including exact algorithms for very restricted classes of graphs, approximation algorithms for some special classes of graphs, heuristics for general graphs, and algorithms with super-polynomial running time. Though we do not discuss all these results, we attempted to include many of these known results in the references to this paper.

1.2 Our results

Our main results are the following.

Theorem 1 *For every $\epsilon > 0$, it is NP-hard to approximate the bandwidth of ringed caterpillars of hair length 1 within a ratio of $2 - \epsilon$.*

The most significant feature of Theorem 1 is that it is tight in the sense that there are simple algorithms that do approximate the bandwidth of ringed caterpillars with hair length 1 within a ratio of 2 [22]. The proof of the theorem easily extends to other classes of graphs for which approximation ratio 2 is known, such as cycles of cliques and circular arc graphs with uniform arc lengths.

Theorem 2 *For every constant $c > 0$, it is NP-hard to approximate the bandwidth of caterpillars of hair length 1 within a ratio of c. Moreover, under the assumption that NP (say, 3SAT, to be specific) cannot be solved in time $n^{O(logn/\log\log n)}$, there is no polynomial time algorithm that approximates the bandwidth of caterpillars within a ratio better than $c'\sqrt{\frac{\log n}{\log\log n}}$, where $c' > 0$ is some sufficiently small constant.*

There are two significant features in Theorem 2. One is that it establishes NP-hardness for any constant ratio. The other is that it also addresses nonconstant approximation ratios. The question of what it would take to obtain a hardness ratio that matches (up to constant factors) the known approximation ratio $O(\log n/\log\log n)$ [11] will be discussed in Section 4.12.

Similar to the approach of [2], our proof of Theorem 1 is based on a direct reduction from an NP-hard problem (such as 3SAT), and does not attempt to build on any previous hardness of approximation result. In particular, we make no use of the PCP theorem and its variants which is a common methodology for proving hardness of approximation results. To simplify the presentation, we choose an NP-hard problem with more symmetries than 3SAT, namely, the problem BM4SAT (balanced monotone 4SAT, see Section 2.2). The main new challenges in the proof of Theorem 1 compared to the proofs in [2] is that in Theorem 1 the reduction needs to produce ringed caterpillars for which the optimal linear arrangement involves extensive folding of the backbone. (Finding the optimal layout of a ringed caterpillar with strand length 1 can be done in polynomial time if it is known that the number of folding locations is bounded by a constant.) In [2], no notion of folding is required (since the graphs produced by the reduction were more general than the simple graphs in our reductions). Beyond the issue of forcing the folding, the improvement of approximation ratio from 3/2 in [2] to 2 in Theorem 1 is mainly a result of paying more attention to details and extracting the maximum out of the proof technique.

The proof of Theorem 2 is considerably more complicated than the proof of Theorem 1. The idea is to start by proving a constant hardness of approximation result for some constant larger than 1 (say, 3/2), and then recursively amplifying it by creating large caterpillars in which smaller caterpillars are embedded. The base case of this recursive construction appears to be conceptually not that different from the proofs of Theorem 1. However, this analogy is misleading. Setting things up so that the recursion works properly and indeed amplifies the hardness result involves many additional issues that significantly complicate the proofs. To illustrate a major source of the difficulty, compare the issue of hardness amplification for bandwidth with that of hardness amplification for the maximum clique problem. For maximum clique, a certain graph product results in squaring the size of the clique [13], and hence squaring also the hardness of approximation gap. By repeatedly applying this product, the gap can be made arbitrarily large. For the bandwidth problem, no such graph product can exist, as it would lead (by taking a super-constant number of iterations) to hardness of approximation ratios larger than the approximation ratios that one can achieve algorithmically, implying that NP has quasi-polynomial time algorithms. Indeed, our recursive approach does not increases uniformly the bandwidth of all graphs. For some graphs it increases the bandwidth (by a factor of $\Omega(k)$ after k recursive applications) and for others it does not. The hardness of approximation gap is created because original *yes* instances of the BM4SAT problem give rise to caterpillars for which recursion does not increase the bandwidth, whereas *no* instances give rise to caterpillars for which recursion does increase the bandwidth. The need to reach such a distinction comes up already in the base case of the recursion, and additional levels of difficulty are introduced with every level of the recursion. For example, the number of different types of "gadgets" used by our construction grows as the number of levels of the recursion grows.

The preliminary version [32] of our paper states some additional hardness of approximation results. We chose not to include proofs of these results in this manuscript since the additional value that they provide does not seem to justify making this manuscript longer than it already is.

A different approach for proving hardness of approximation results for bandwidth was suggested by Shimon Kogan (private communication, 2004). It uses a reduction from the problem of Balanced Bipartite Independent Set (BBIS). This last problem is currently known not to be approximable unless NP has subexponential time algorithms [10, 18]. Using the principles outlined by Simon Kogan, we prove the following proposition that addresses questions that were left open in [21, 17].

Proposition 3 *There is no PTAS for the bandwidth problem on asteroidal-triple-free graphs (and neither on graphs of minimum degree $n/2$) unless NP has subexponential time algorithms.*

Possibly, the statement of Proposition 3 can be strengthened so as to show NP-hardness. However, we chose to include this proposition in this weak form because its proof is simple, and illustrates a different approach for showing that approximating the bandwidth is a difficult task.

1.3 Organization of this paper

In Section 2 we present components that are common to the proofs of both of our main theorems. This includes the notion of bucketwidth, the problem of BM4SAT, various basic subgraphs of caterpillars (such as paths, brushes), the principles behind basic gadgets (keys and holes), and the intended interpretation of various segments of a caterpillar that give a correspondence between the structure of a caterpillar and the original BM4SAT instance. Section 2 also includes a special subsection 2.7 that contains much of the terminology that is used in the proofs of Theorems 1 and 2, for easy reference in case of need.

In Section 3 we prove Theorem 1. This proof can also serve as an introduction to the more difficult proof of Theorem 2 which appears in Section 4. Section 5 contains the proof of Proposition 3 and can be easily understood without reading any of the other sections.

2 Preliminaries

2.1 Bandwidth versus bucketwidth

Definition 8 *A bucket arrangement of a graph is a placement of its vertices into consecutive buckets, such that the endpoints of an edge are either in the same bucket or in adjacent buckets. The bucketwidth is the number of vertices in the most loaded bucket. The bucketwidth of a graph is the minimum bucketwidth of all bucket arrangements of the graph.*

The following lemma is well known [11]. We give a proof for completeness.

Lemma 4 *Let b denote the bandwidth of a graph G and bw its bucketwidth. Then $bw \leq b \leq 2bw$.*

Proof: Given a bucket arrangement for a graph G with bucketwidth bw, create a linear arrangement bucket by bucket, where vertices in the same bucket are ordered in an arbitrary order. This gives a linear arrangement with bandwidth at most $2bw$.

Given a linear arrangement with bandwidth b, create a bucket layout of bucketwidth at most bw by dividing the vertices in the linear arrangement in groups of size b and putting each in its own bucket. $\square$

While proving a hardness of approximation which is "large" (Section 4), it will be more convenient to present our result as one concerning the hardness of approximating bucketwidth of caterpillars, rather than bandwidth. Recall that by Lemma 4, hardness of approximating bucketwidth within a factor of ρ implies hardness of approximating bandwidth within a factor of $\rho/2$. Since in our case we will be able to make ρ an arbitrarily large constant (and even super-constant), $\rho/2$ will be a value of the same nature. Hence the same hardness results that we will prove for bucketwidth will apply also to bandwidth.

In fact, even when proving a hardness of factor 2 (Section 3) it will be more convenient for us to first consider bucketwidth, and then translate the result to bandwidth by showing that for the particular graphs that are constructed by our reductions, the bucketwidth and bandwidth are roughly the same (rather than being a factor of 2 from each other as might happen in Lemma 4).

2.2 BM4SAT

Our hardness of approximation result for bucketwidth will be by a reduction from BM4SAT. We shall use $2n$ to denote the number of variables in the input BM4SAT instance, and m to denote the number of clauses. The term *yes* instance will always refer to the BM4SAT instance being satisfiable, whereas the term *no* instance will refer to the BM4SAT instance not being satisfiable.

Balanced Monotone 4-Satisfiability (BM4SAT)

Input A boolean formula $\xi = \bigwedge_{j=1}^{m}(v_{j_1} \vee v_{j_2} \vee v_{j_3} \vee v_{j_4})$ on variables $v_1, v_2, \ldots, v_{2n}$, where each $v_{j_i} \in \{v_1, \ldots, v_{2n}\}$. The terms of the form $(v_{j_1} \vee v_{j_2} \vee v_{j_3} \vee v_{j_4})$ are called clauses and are denoted as C_j.

Problem Given a boolean formula ξ, in a *yes* instance of the problem there exists an assignment A of the variables such that it sets exactly n variables to *true* and in every clause exactly two variables are set to *true*. In a *no* instance of the problem, no assignment of the variables satisfies exactly two variables in each clause.

Proposition 5 BM4SAT *is NP-complete.*

We first define the UNIFORM 3-HYPERGRAPH 2-COLORING problem, which we later reduce to BM4SAT. A hypergraph $H = (X, Y)$ is a finite set X of *vertices* and a collection Y of non-empty subsets of X called *edges*. A *Uniform 3-hypergraph* is a hypergraph for which all edges are of size exactly 3. A 2-coloring is a mapping $\phi : X \to \{0, 1\}$ such that no edge of H has all vertices of the same color. The UNIFORM 3-HYPERGRAPH 2-COLORING (U3H2C) problem is known to be NP-complete (this problem is a special case of SET SPLITTING and was shown to be NP-complete by Lovász [25]; see also [13]).

Proof: Given an instance $H = (X, Y)$ of U3H2C we define a BM4SAT instance ξ_H as follows:

- For each vertex $x \in X$, we introduce a variable v_x. For each edge $y \in Y$, we introduce a variable v_y. Also, for each element $z \in X \cup Y$ we introduce a variable v_z.
- For each edge $y \in Y$ with $y = \{x_1, x_2, x_3\}$ we define a clause $v_{x_1} \vee v_{x_2} \vee v_{x_3} \vee v_y$. The instance ξ_H is a conjunction of all clauses of above kind. Note that variables of type v_z do not appear in any clause.

There is a natural correspondence between the assignment of variables in ξ_H and a 2-coloring of H. Assume that there are $2n$ variables and m clauses in the BM4SAT instance.

- On a *yes* instance of ξ_H there exists an assignment of variables which satisfies exactly two variables in each clause and also exactly n variables overall. Define a coloring ϕ which maps those v_x variables set to *true* to the color 0. As each edge in H has a corresponding clause in ξ_H and exactly one extra vertex is introduced for every edge, it has two vertices of different colors.
- Now we show that a 2-colorable hypergraph H transforms to a *yes* instance of BM4SAT. To do this we show an assignment of the variables given the mapping $\phi : X \to \{0, 1\}$. The assignment sets all vertices with color 0 to *true*. For each edge $y \in Y$ we look at the number of vertices set to 0. The variable v_y is set to true if y has more 1 colored vertices than 0 colored vertices and vice versa. The extra vertices v_z are asigned arbitrary values in a way that the number of variables assigned to *true* is exactly equal to number of variables assigned $false$. This assignment sets exactly two variables in each clause and n variables overall to *true*.

□

2.3 Folding

As shown in [11], the unfolded bandwidth of a caterpillar can be approximated within a constant factor. Hence to prove our stronger hardness of approximation results (Theorem 2), it is inevitable that we shall construct caterpillars in which minimizing the bucketwidth requires folding the backbone. Folding the backbone causes distinct vertices of the backbone to share the same bucket. Depending on where the backbone is folded, there are different choices of which vertices are those who share the same bucket. At a very high level, the hardness of approximation results that we prove are based on the difficulty of deciding in which locations to fold the backbone, and consequently, which are the groups of vertices that are placed in the same buckets. An over-simplification of our approach is to think of some of the backbone vertices as *light* and some as *heavy* (Section 2.4). The bucketwidth of a bucket arrangement will be roughly proportional to the maximum number of heavy backbone vertices that are placed in the same bucket. On *yes* instances our reduction would produce caterpillars that can be folded in such a way that every bucket contains at most one heavy backbone vertex (together with arbitrarily many light backbone vertices). On *no* instances our reduction would produce caterpillars for which no matter how they are folded, there will be a bucket with many heavy backbone vertices. In both cases, if the backbone is not folded at all, the bucketwidth will be very large (hence the need to fold), but the gadget ensuring this last property in not based on the notion of heavy backbone vertices. Instead it is based on another class of backbone vertices that we call *folding* vertices. These folding vertices are the only backbone vertices in the caterpillar that are attached to strands of length greater than 1 (see Section 4.2 for more details).

The notion of folding is crucial in a similar way to the proof of Theorem 1. For example, it is easy to see that dynamic programming can be used to compute the bucketwidth of a ringed caterpillar of strand length one in time that is $n^{O(f)}$, where f is the maximum number of backbone vertices in which folding occurs. We note that for ringed caterpillars, it is inevitable that the backbone be folded in at least two locations. This will suffice in order to make the use of special folding vertices unnecessary. Hence the ringed caterpillars produced by our reductions will indeed only have strands of length 1.

2.4 Paths and brushes

We introduce here a notion of *light* and *heavy* backbone vertices. A light backbone vertex is one that has no strands connected to it, and a heavy backbone vertex is one that has many strands connected to it. To localize the effect of these strands, they will all be of the minimum possible length, namely, length 1. The number of such strands will be denoted by b, which roughly corresponds to our intended bucketwidth.

In addition to dealing with individual backbone vertices that are light or heavy, it will be convenient for us to deal also with whole consecutive portions of the backbone that are light or heavy.

For light portions of the caterpillars, we introduce the notion of a *path.* For an integer value $x > 0$, a *path* P_x is a sequence of x consecutive backbone vertices, none of which has any strands connected to it. (Fig 1)

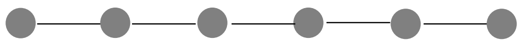

Figure 1: A path of length 6

For heavy portions of the caterpillars, we introduce the notion of a *brush.* For an integer value $x > 0$, a *brush* B_x is a sequence of x consecutive backbone vertices, each of which has exactly b strands of length one connected to it. (Fig 2)

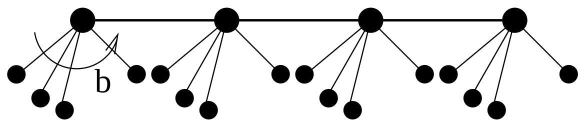

Figure 2: A brush of length 4. Each vertex in the backbone has b vertices attached to it.

2.5 Keys and holes – simplified version

A very useful abstraction in our reduction (and earlier reductions of [2]) is the notion of *keys* and *holes* (where the word hole is used in the sense of a keyhole). Later on we shall describe several types of key-hole pairs. Here we present the most basic key-hole construct.

Fix two positive integers x and y with y much larger than x. Then a portion of the caterpillar of the form $P_yB_xP_y$ can serve as a key, and a portion of the caterpillar of the form $B_yP_xB_y$ can serve as the corresponding hole. The key and hole are said to be *aligned* in a bucket arrangement if there is a sequence of $y + x + y$ consecutive buckets such that the backbone vertices of the key are placed in order in these buckets, and the backbone vertices of the hole are placed in order in these buckets. Note that in this case each bucket contains at most one heavy vertex (a backbone vertex from a brush) from the key-hole pair in each bucket. The key and hole are said to be *misaligned* in a bucket arrangement if there is at least one bucket with two heavy backbone vertices in it (either both from the key, or both from the hole, or one from the key and one from the hole).

Later we will construct more complicated key-hole pairs. The keys will still involve basic constructs of the form $P_yB_xP_y$ (for various values of x and y), and we call the brush B_x a *tooth*. The holes will involve basic constructs of the form $B_yP_xB_y$, and we call the path P_x a *dent*. Various types of key-hole pairs can differ by the number of teeth that they have (and hence also by the number of dents), and by the size of the teeth (the values of x and y). We shall also need to refine the notion of a key and a hole being misaligned, so that it can be used also in recursive constructions.

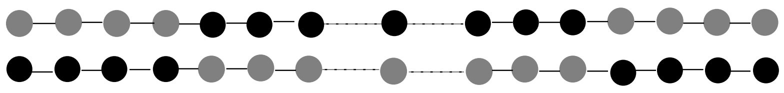

Figure 3: An aligned key-hole pair. The heavy vertices are displayed as darker than the light vertices. Each bucket has exactly one heavy vertex.

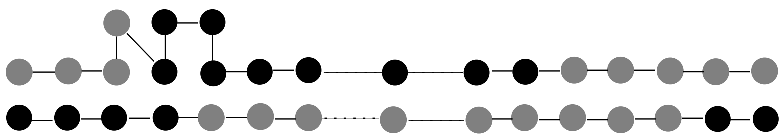

Figure 4: A misaligned key-hole pair. Some buckets have 2 or more heavy vertices and others have none. We count only the heavy vertices in each bucket.

2.6 Segments of the caterpillar

In our reduction, the caterpillar will have several disjoint *segments*, where a segment is a set of consecutive backbone vertices together with all strands that are connected to them. Each segment has a corresponding interpretation in the BM4SAT instance. Here we shall present the main segments, but leave the description of their exact structure until later. It suffices at this stage to think of each segment as a sequence of paths and brushes of various sizes (that depend on the nature of the segment).

There will be two segments that we call *formula segments*. One of them is the *SAT segment* and the other the *UNSAT segment*. These two segments will be mirror images of each other in the sense that they follow the same pattern of paths and brushes, but in opposite directions. This pattern will encode the structure of the BM4SAT formula in the following sense. The SAT segment will mostly be composed of brushes. It will be partitioned into m subsegments, one for each clause. Each subsegment will contain (among other things) two holes that we call *functional holes*, corresponding to the requirement that in order to satisfy the clause exactly two of its variables need to be set to true. The UNSAT segment has the same structure (in reverse direction), and this time the two functional holes in a clause subsegment correspond to the requirement that two variables need to be set to false.

There will be $2n$ segments that we call *variable segments*, one for every variable. Each variable segment will mostly be composed of paths and will have (among other things) a number of keys (that we call *functional keys*) that equals the number of clauses in which the variable appears. The spacing between these keys (number of backbone vertices separating them) along one variable segment will roughly match the spacing in a formula segment between the clause subsegments that correspond to the clauses in which the variable appears.

All the segments described above belong to the backbone of one caterpillar. The caterpillar in our reduction will include gadgets that force it to fold in a certain way (or else pay in the bucketwidth). The consequence of the folding is that one may think of the bucket arrangement as having an *active region* that is visited by all segments. In a sense, one may think of all segments as starting at the same bucket in the middle of the active region, with the SAT segment laid out to the right and the UNSAT segment laid out to the left. For each variable segment, the intention is that if the variable is set to true it is laid out to the right, and if it is set to false it is laid out to the left. For *yes* instances, all functional keys would find a corresponding functional hole to fit in. For *no* instances, three functional keys (from three different variables) will end up on one clause subsegment (that has only two functional holes), because some clause has either at least three variables set to true (and then the clause subsegment will be in the SAT segment) or at least three variables set to false (and then the clause subsegment will be in the UNSAT segment). This will give us the required gap in the bandwidth.

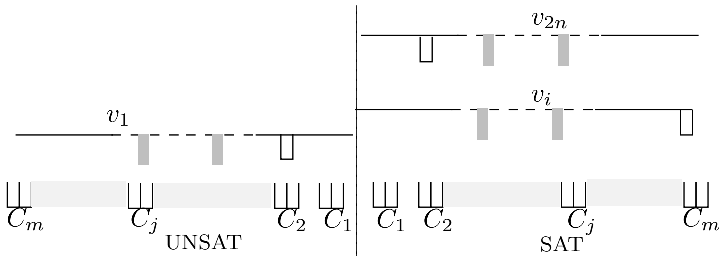

Figure 5: (a) There are two formula segments which are mirror images of each other. (b) Each clause subsegment has two functional holes. (c) Each variable segment is either laid out to the right or left.

As the above description shows, various segments of the caterpillar have a corresponding interpretation in the BM4SAT formula, and we are able to express the action of setting a variable to true and to false, to check how many variables satisfy a clause, and to penalize the bucketwidth if there is a clause in which the number of variables set to true is not exactly two. This is the *functional* aspect of our reduction. However, the reduction also has many *structural* aspects which force the caterpillar to fold in such a way that allow the functional aspects to manifest themselves. The structural gadgets of the reduction will be described later.

2.7 Terminology

Let G_1 and G_2 be two caterpillars. The graph G_1G_2 or G^2 (in case $G_1 = G_2 = G$) is defined by concatenating the backbones of the two caterpillars ie, the rightmost backbone vertex of G_1 is attached to the leftmost backbone vertex of G_2 by an edge.

Given two caterpillars G and H, the caterpillar G *spans* the caterpillar H in a bucket arrangement if the backbone of G contributes to all the buckets to which the backbone of H contributes to.

For easy reference in case of need, we list here many of the terms that we use in the proofs of Theorems 1 and 2, a short reminder as to what they mean, and the sections in which they are defined. Note that some of terms are defined only in subsequent sections.

- n. Half the number of variables in the BM4SAT instance. (Section 2.2.)
- m. The number of clauses in the BM4SAT instance. (Section 2.2.)

- *yes* and *no* instances. Always refer to the BM4SAT instance that is the source of the reduction. (Section 2.2.)
- *light* vertex. A backbone vertex with no strands. (Section 2.3.)
- *heavy* vertex. A backbone vertex with b unit length strands. (Sections 2.3 and 2.4.)
- *path.* A consecutive sequence of light backbone vertices. (Section 2.4.)
- *brush.* A consecutive sequence of heavy backbone vertices. (Section 2.4.)
- b. The number of strands connected to a heavy vertex (unit length strands) or folding vertex (long strands). Will roughly correspond to the bucketwidth in case of *yes* instances. b is much larger than n, and assumed to be odd. (Section 2.4.)
- *keys* and *holes.* Types of gadgets that are introduced in Section 2.5, and elaborated on in later sections.
- *teeth* and *dents.* Subparts of keys and holes (Section 2.5).
- *span.* See above (Section 2.7).
- i*-coupling.* Set of buckets in which a key K_i has fit a hole H_i (Section 3.1).
- *folding* gadget. A very long brush followed by an equally long path. (Section 3.2. Used for ringed caterpillars of hair length 1.)
- *folding* vertex. A backbone vertex with b long strands connected to it. (Sections 2.3 and 4.2. Used for caterpillars.)
- *functional* versus *structural.* A functional gadget encodes a decision regarding setting a BM4SAT variable to true or false. A structural gadget forces folding of the caterpillar in a way that makes the functional gadgets effective. (See end of Section 2.6 and beginning of Section 3.1.)
- *segments* of a caterpillar. Connected subgraphs of the caterpillar that can be viewed as high level building blocks. The two *formula segments* encode the clauses, and the $2n$ *variable segments* encode the variables. (Section 2.6.) The *middle segment* lies between the two formula segments and enforces a certain alignment between variable segments and the formula segments. (Section 3.2.)
- *subsegments.* Each formula segment is composed of m clause subsegments. Likewise, every variable segment is composed of m clause subsegments. (Section 2.6.)
- *active* region. The set of buckets that contains the formula segments. Folding gadgets forces the variable segments to also reach the active region. (Section 2.6.)
- *shallow* layout. A bucket arrangement whose bucketwidth is smaller than the intended bucketwidth for *no* instances. (Section 3.5.)
- *canonical* layout. A bucket arrangement in which the structural folds occur at their intended locations. (Section 3.5.)
- G_i. The caterpillar produced at the ith level of the recursive construction. (Section 4.1.)
- g_i. Length of backbone of caterpillar G_i. (Section 4.1.)
- k-pile. A bucket that suffers k *hits.* (Section 4.3.)

- *hit.* Either a heavy vertex, or $b/3$ strand vertices whose strands are attached to the same folding vertex. (Section 4.3.)
- X. A caterpillar of the form $G_{k-1}B_{\Delta}$. (Section 4.6.)
- *stack.* If two or more G_{k-1} share a set of buckets then a *stack* is formed. The number of G_{k-1} contributing to this set is called the *height* of the stack. (Section 4.6.)

3 Ringed caterpillars

In this section, we prove Theorem 1, showing the hardness of approximating the bandwidth of ringed caterpillars of strand-length 1 within a factor of $(2 - \epsilon)$, for every constant $\epsilon > 0$. This result is an immediate consequence of the following theorem, which we prove in this section.

Theorem 6 *Given a boolean formula $\xi = \bigwedge_{j=1}^{m}(v_{j_1} \vee v_{j_2} \vee v_{j_3} \vee v_{j_4})$ and a constant $\epsilon > 0$, there is an efficient construction of a graph $G_\xi = (V, E)$ such that*

- *G_ξ is a ringed caterpillar of hair length at most 1.*
- *The number of vertices in G_ξ is bounded by $poly(n)b$.*
- *If ξ is a* yes *instance of BM4SAT then the bandwidth of G : $b(G_\xi) \leq b + cn$, where c is a constant.*
- *If ξ is a* no *instance of BM4SAT then the bucketwidth of $G : bw(G_\xi) > (2 - \epsilon)b$.*

Theorem 6 distinguishes between bandwidth and bucketwidth. From Lemma 4 and Theorem 6 we conclude that if ξ is a *yes* instance of BM4SAT then G_ξ has bandwidth at most $b + cn$ and if ξ is a *no* instance the bandwidth is at least $(2 - \epsilon)b$. As b can be made much larger than n (e.g., $b = O(n^2)$), this implies a $(2 - \epsilon)$ hardness result for bandwidth on ringed caterpillars of hair length 1.

3.1 Designing keys and holes

Recall Section 2.5 which introduced the concept of keys and holes.

At a high level, one can make a distinction between two classes of keys and holes, depending on their intended usage. One class is that of *structural* keys and holes. Their purpose is to force the caterpillar to fold in certain ways, regardless of whether the caterpillar corresponds to a *yes* or a *no* instance. The other class is that of the *functional* ones. The decision of which hole to put a functional key in will roughly correspond to a decision whether to set a variable of the BM4SAT instance to *true* or *false.*

In this section the construction of structural and functional keys are based on the same idea, and the choice of which type of key-hole pair is used for structural purposes and which type is used for functional purposes to quite arbitrary. In contrast, in Section 4, function keys and structural keys will need to have additional properties that are not required here, and these will lead to different types of keys for each class.

Before presenting the types of keys and holes used in this section, we present two technical lemmas whose goal is to (formally) establish (the intuitive fact) that there are a limited number of ways in which brushes can be laid out in a bucket arrangement without suffering a high penalty in the bucketwidth.

Lemma 7 *Let B_1 and B_2 be two brushes and in a particular bucket layout both of them contribute to a sequence of buckets $b_{i+1}, b_{i+2}, \ldots, b_{i+\ell}$ with $\ell \geq \frac{4}{\epsilon}$. Then the bucketwidth of this layout is larger than $(2 - \epsilon)b$.*

Proof: Each brush contributes at least one heavy backbone vertex in each bucket $[b_{i+1}, b_{i+\ell}]$. Hence, the $2b\ell$ respective strand vertices go into at most $\ell + 2$ buckets. By averaging over these buckets, the bucketwidth is at least:

$$\begin{aligned} bw &\geq \frac{2\ell + 2b\ell}{\ell + 2} \\ &= (2 - \frac{4}{\ell + 2})(b + 1) \\ &> (2 - \epsilon)b \end{aligned}$$

□

We next show that a brush cannot be folded on itself ie., all backbone vertices within the brush lie roughly between the leftmost backbone vertex (the bucket in which it lies) and the rightmost backbone vertex (the bucket in which it lies) in every bucket arrangement with bucketwidth at most $(2 - \epsilon)b$.

Lemma 8 *Let B_x be a brush of length x. Consider an arbitrary bucket arrangement with bucketwidth at most $(2 - \epsilon)b$. Let the bucket in which the rightmost and the leftmost backbone vertices of B_x lie in be r and l respectively. Without loss of generality assume that $l \leq r$. Then:*

a. Every backbone vertex of B_x lies in $[l - \ell, r + \ell]$, where $\ell = \frac{4}{\epsilon}$.

b. $|r - l| \geq \frac{x}{2}(1 + \frac{\epsilon}{2}) - O(\frac{1}{\epsilon})$.

Proof: To prove the first part of the lemma, assume for the sake of contradiction that the backbone contributes to buckets $r + \ell + 1$ (or $l - \ell - 1$). This implies that the backbone contributes at least two vertices per bucket from r to $r + \ell$ (or to $l - \ell$ to l). Similar to Lemma 7, this will imply a bucketwidth larger than $(2 - \epsilon)b$.

The second part of the lemma follows from averaging. As all backbone vertices lie within $[l - \ell, r + \ell]$, we have that

$$\begin{aligned} (2 - \epsilon)b &\geq \frac{(b + 1)x}{(r - l) + 2\ell + 2} \\ |r - l| &\geq \frac{x}{2 - \epsilon} - 2\ell - 2 \\ &\geq \frac{x}{2}(1 + \frac{\epsilon}{2}) - O(\frac{1}{\epsilon}) \end{aligned}$$

□

We now design three different key-hole pairs (Fig 6). For this we use parameters $\ell > \Omega(1/\epsilon)$, $\lambda > \Omega(\ell/\epsilon)$ and $\delta > \Omega(\lambda/\epsilon)$. For a key-hole pair (K_i, H_i) define h_i as the length of the backbone of H_i (and K_i).

$$\begin{array}{ll} K_1 = P_\delta B_\lambda P_\delta & H_1 = B_\delta P_\lambda B_\delta \\ K_2 = P_\delta B_\lambda P_\lambda B_\lambda P_\delta & H_2 = B_\delta P_\lambda B_\lambda P_\lambda B_\delta \\ K_3 = P_\delta B_\lambda P_\lambda B_\lambda P_\lambda B_\lambda P_\delta & H_3 = B_\delta P_\lambda B_\lambda P_\lambda B_\lambda P_\lambda B_\delta \end{array}$$

Figure 6: The key-hole pairs

A key is made of structures of the form PBP and a hole is made of structures of the form BPB, where P and B are paths and brushes of some length. We say that a key K *fits* a hole H if every tooth of K is spanned by a dent of H and the bucketwidth of the arrangement is $b+2$. A key K does not fit a hole H if in every bucket arrangement where a tooth of K is spanned by a dent of H, the bucketwidth of the arrangement is $> (2-\epsilon)b$. For a bucket arrangement, an *i-coupling* is a key K_i fitting a hole H_i.

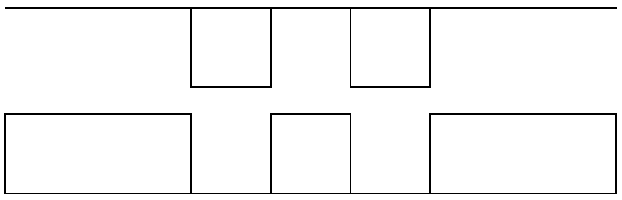
Figure 7: The key K_2 fits the hole H_2

Lemma 9 *For the key-hole pairs as defined in Figure 6,*

a. A key K_i fits a hole $H_{j\geq i}$.

b. A key K_i cannot fit a hole $H_{j<i}$

c. An i-coupling cannot span another tooth. (Once a hole is coupled with its respective key, no additional keys fit it).

Proof:

a. We give a bucket arrangement with bucketwidth $b+2$. Lay out the backbone vertices of H_j in consecutive buckets. Likewise lay out the backbone vertices of K_j in consecutive buckets, with the leftmost backbone vertex of K_i at the same bucket as the leftmost backbone vertex of H_j, and similarly for the rightmost vertices. Put every strand vertex in the same bucket as the backbone vertex it is attached to. This layout has bucketwidth $b+2$; b from the strands and one each from the backbones of K_i and H_j.

b. We show that K_3 does not fit H_2. (The other cases to consider are proved in a similar manner.) The brush B_λ of H_2 occupies ar least roughly $\lambda/2$ buckets (by Lemma 8). The layout of the two dents in H_2 cannot overlap (again, a consequence of the layout of the brush B_λ and Lemma 8). Each tooth of K_3 can overlap with at most one dent of H_2 (as otherwise it crosses over the brush B_λ, contradicting Lemma 7). Hence two teeth of K_3 need to be placed in one dent of H_2. They can use up to ℓ buckets to either side of the set of buckets in which the dent is placed (by Lemma 7). Hence two brushes each of size λ are placed in at most $\lambda+2\ell$ buckets, and a standard averaging arguments (using the fact that $\ell < \epsilon\lambda$) shows that at least one bucket must contain more than $(2-\epsilon)b$ vertices

c. The proof is based on the number of teeth being at least one more than the number of dents. Details are similar to the previous item, and hence omitted.

□

The key-hole pair (K_1, H_1) will be used as a functional key-hole pair and the key-hole pairs (K_2, H_2) and (K_3, H_3) will be used as structural key-hole pairs.

3.2 Forcing the main structural folds

The high level structure of the ringed caterpillar G_ξ will be as follows. It will have $2n+2$ segments connected in a ring. This includes two formula segments which are adjacent, the SAT segment and the UNSAT segment, and $2n$ variable segments. In the corresponding bucket arrangement, we call the buckets used by the formula segments the *active region.* We will need to force a fold between the formula segments and the variable segments so that the variable segments will also fall in the active region. In fact, each variable segment will be similar in length to a formula segment, and we shall need to enforce many folds, roughly one per variable.

To ensure that the variable segments are placed in the active area, we shall place a *folding gadget* (formally defined in Section 3.3) between the UNSAT segment and its adjacent variable segment and between the SAT segment and its adjacent variable segment. The folding gadget will be composed of a very long brush and a slightly longer path. The only way to avoid large bucketwidth (and still close the ring) would be to layout one long brush to the left of the active region the other long brush to the right of the active region, and use the long paths to return to the active region.

Proposition 10 *(See Section 3.5) For* no *instance, either all variable segments fall in the active region or the bucketwidth is at least* $2-\epsilon)b$.

We need each of the $2n$ variable segments to start in between the SAT and UNSAT segments. It can then be laid out either to the left which corresponds to setting the variable to *false*, or to the right which corresponds to setting it to *true*. For this reason we introduce what we call a *middle segment.* It is formed by a sequence of $2n$ holes of type H_3. Each of the $2n$ variable segments starts with a key of type K_3. The hole H_3 appears nowhere else in the graph G_ξ. This insures that the variable always starts at the middle segment. We put the middle segment between the SAT and UNSAT segments.

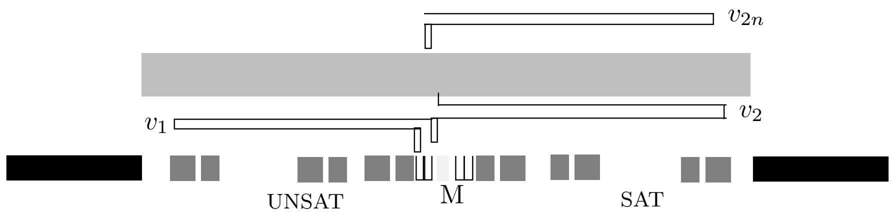

Figure 8: (a) Formula segments lie between folding gadget (represented in black). (b) Variable segments lie in the active region. (c) M is the middle segment.

3.3 The full graph

Recall that we wish to obtain a hardness of approximation ratio of $2-\epsilon$. In Section 3.1 we introduced a parameter $\ell = \Omega(1/\epsilon)$. We shall also use parameters denoted by Greek letters where $\alpha > \beta > \gamma > \delta > \lambda$. We require that $\alpha > \Omega(\beta/\epsilon)$, $\gamma > \Omega(n\delta/\epsilon)$, $\delta > \Omega(\lambda/\epsilon)$ and $\lambda > \Omega(\ell/\epsilon)$.

We now describe the various parts of our graph. Note that we shall be using the concatenation notation introduced in Section 2.7. Recall also that B_x is a brush of length x, P_y a path of length y, and K_i and H_i are various types of keys and holes. We shall use h_i to denote the length of the backbone of a hole of type H_i.

SAT: is $C_1 \ldots C_m$, where C_j is the clause subsegment for the clause j.
UNSAT: is the mirror image of SAT ie., $C_m \ldots C_1$.
CLAUSE SUBSEGMENT: is $B_\gamma H_1^2 B_\gamma H_2^n$.
MIDDLE SEGMENT: is $(H_3)^{2n}$.

Folding Gadget: is $B_\alpha P_{\alpha+\beta}$. Here β is the sum of the length of the backbone of SAT (or UNSAT) segment.
Hole Group: Is a group of holes concatenated by the end points of their backbones. The graph G_ξ has hole groups of three types: $(H_1)^2$, $(H_2)^n$ and $(H_3)^{2n}$.
Slackness: We shall not distinguish between different holes in the same hole group. Hence for example, a key K_2 may be placed in any of the n holes in a hole group $(H_2)^n$. For this reason we introduce the concept of slackness around each key, where the amount of slackness depends on the type of hole group. The slackness will be of type 1, 2 or 3, and be in form of paths of length P_{2h_1}, P_{nh_2} and P_{2nh_3}, corresponding to hole groups $(H_1)^2$, $(H_2)^n$ and $(H_3)^{2n}$ respectively. Keys of types K_1, K_2 and K_3 appear in the variable segments. Every key of type i will have slackness of type i on both sides of the key. This gives us the freedom to choose an arbitrary hole in the hole group to fit the key in. We denote the slackness by s, and its type is the same as that of the key adjacent to it.
Variable Segments: each variable v_i will be associated with a variable segment $sK_3sv_i^1 \ldots v_i^m P_\beta$. Here v_i^j denotes the jth subsegment of the variable segment v_i. (Note that this is an abuse of notation, in contradiction to our concatenation notation, under which v_i^j would have been interpreted as a caterpiller v_i concatenated j times. We hope that no confusion will arise because of this notation abuse.)
Variable Subsegment: v_i^j is $P_\gamma sK_1sP_\gamma sK_2s$, if variable v_i appears in clause C_j and $P_\gamma P_{2h_1}P_\gamma sK_2s$ if v_i does not appear in C_j.

A summary of the various parts of the graph appears in Figure 9.

$$\begin{aligned}
\text{SAT} &= C_1 \ldots C_m \\
\text{UNSAT} &= C_m \ldots C_1 \\
M &= (H_3)^{2n} \\
C_j &= B_\gamma (H_1)^2 B_\gamma (H_2)^n \\
v_i &= K_3 v_i^1 \ldots v_i^m P_\beta \\
v_i^j &= P_\gamma P_{2h_1} P_\gamma K_2 && \text{if } v_i \notin C_j \\
&= P_\gamma K_1 P_\gamma K_2 && \text{if } v_i \in C_j
\end{aligned}$$

Figure 9: Every key of type i has a slackness of the same type on both sides of the key. For simplicity of the description, we have omitted the slackness from this figure.

The overall structure of the graph is depicted in Fig 10.

$$\begin{tikzcd}[arrows=-]
P_\alpha - P_\beta - v_1 - \quad \ldots \quad \ldots \quad - v_{2n} - P_\beta - P_\alpha \arrow[d, bend left] \arrow[d, bend right] \\
B_\alpha - C_m - \ldots - C_1 - M - C_1 - \ldots - C_m - B_\alpha
\end{tikzcd}$$

Figure 10: The final graph G_ξ. The length of P_β as well as the variable segments is roughly equal to the length of the SAT and UNSAT segments.

In summary, the graph G_ξ contains the following basic gadgets: Brushes, Paths, Hole Groups, and Keys. There are three kinds of key-hole pairs. The first type (K_1, H_1) is used as a *functional* gadget with H_1 in C_j and K_1 in v_i^j if v_i appears in C_j. There are two holes H_2 in each C_j corresponding to the fact that exactly two variables are set to true in a *yes* instance of the problem. The second type (K_2, H_2) is a *structural* gadget. This one makes sure that exactly n variables lie in the buckets corresponding to SAT (and exactly n for UNSAT), i.e., exactly n variables are set to *true* and exactly n are set to *false*. Also, it makes sure that the v_i^j part contributes to the same buckets as the C_j part. The last type (K_3, H_3) makes sure that after laying $v_{i\{j:1\ldots m\}}$ the P_β is used to span SAT or UNSAT segment to reach the middle segment. This is achieved by putting $2n$ holes H_3 in M and one key K_3 at the beginning of each v_i.

3.4 Layout on a yes instance

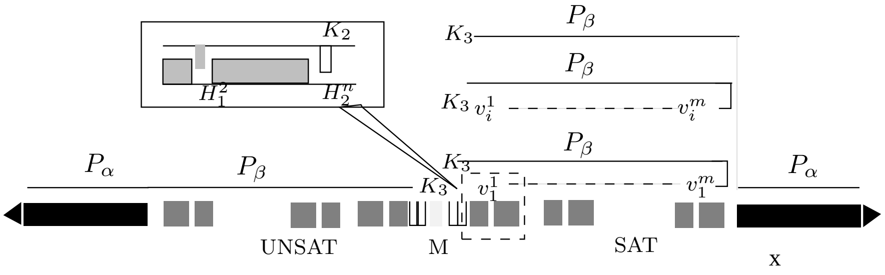

Figure 11: Layout in the *yes* case a. The satellite box shows how the variable subsegment is laid out on top of the corresponding clause subsegment b. The variables set to *true* are laid out to the right and those set to *false* are laid out to the left

We show now that for a *yes* instance of the problem there exists a bucket layout of the graph G_ξ with bucketwidth at most $b + O(n)$. Furthermore, this bucket layout can be converted to a linear arrangement of bandwidth similar to this bucketwidth. The layout is shown in Fig 11. The explanation is as follows:

- We start by putting $B_\alpha\text{UNSAT}M\text{SAT}B_\alpha$ in consecutive buckets at the rate of one backbone vertex per bucket. The strands connected to a particular backbone vertex are put in the same bucket as the backbone.
- We use the P_α to span the brush B_α and use the right P_β to span SAT and the left P_β to span UNSAT.
- We start with the variable v_1. We use the slackness of $2nh_3$ to put the key K_3 in some H_3 out of the $2n$ available ones. The K_3 is followed by another slackness $2nh_3$. This is used to span the rest of M to reach the beginning of SAT if v is set to *true* or span the appropriate part of M to reach the beginning of UNSAT if v_1 is set to false. The slackness is used in a way that does not increase the bucketwidth of any bucket by more than a constant per variable. For example, if say we want to put the key in hole 3 (out of $2n$ available ones) then the slackness $2nh_3$ is used as follows. We start by putting two vertices a bucket beforehand (may involve laying out some vertices on the previous γ brush) such that we spend the P_{2nh_3} before we reach hole 3. We spend the next path of P_{2nh_3} again two vertices a bucket when necessary. After this, for each clause C_j, we use the subsegment v_{1j} to span C_j by fitting the structural keys in the structural hole and functional keys in the functional hole (if available). After spanning C_m by v_{1m}, a path P_β follows. This is used to span SAT or UNSAT to reach the beginning of M.
- Each variable is laid out using exactly the same principles as explained for v_1. It is laid out to the right if the variable is set to *true* and it is laid out to the left if it is set to *false*. Whenever a key needs to placed in a particular hole group, a yet unoccupied hole from that group is chosen for that purpose.

- After laying v_{2n} the right path P_β follows which we already laid out in the first item. The slackness following the key K_3 is used to join v_{2n} to the right path P_β. This completes the layout.

The layout above ensures that every structural key of type K_3 will be placed in a distinct hole of type H_3 in the middle segment. If exactly n variables are set to true (and n variables set to false), this ensures that every structural key of type K_2 will be placed in a distinct hole of type H_2. If every clause is satisfied by exactly two literals this ensures that every functional key of type K_1 is placed in a distinct hole of type H_1. Overall, every bucket contains at most one heavy vertex and at most $O(n)$ light vertices, and the bucketwidth in $b + O(n)$.

The bucket arrangement above can be converted to a linear arrangement as follows. We put all the backbone vertices at the bottom of each bucket and then we do the layout by numbering the vertices in the bucket bottom to top and the buckets themselves left to right. From the discussion above, all the buckets contain at most one backbone vertex from SAT (or UNSAT), n from the paths P_β and at most a constant per variable. So the number of backbone vertices is at most $O(n)$ per bucket. The fact that only backbone vertices are involved in edges between adjacent buckets implies that the bandwidth of the above layout is at most $b + O(n)$. Taking b to be much larger than n/ϵ, the bandwidth is at most $b(1 + O(\epsilon))$.

3.5 Layout on a no instance

We say that a layout is *shallow* if the bucketwidth of the layout is at most $(2 - \epsilon)b$. In this section, we show that a shallow layout of G_ξ exists if and only if it corresponds to a *yes* instance of BM4SAT (in which case the bucketwidth is actually $b + O(n)$, as shown in Section 3.4).

Informally, we say that a bucket-arrangement of G_ξ is *canonical* if the layout follows the general structure depicted in Figure 10. This includes several aspects.

i. The brushes B_α of the two folding gadgets do not overlap. The region between them is then called the *active* region, and contains the formula segments and the middle segments.

ii. The folding gadget is indeed folded. Namely, the variable segments lie in the active region.

iii. The key of type K_3 of every variable segment v_i is spanned by the middle segment M.

iv. The structural key K_2 of v_i^j is spanned by the structural hole group of C_j (either of the SAT segment or of the UNSAT segment).

v. The functional key K_1 (if any) of v_i^j is spanned by the structural hole group of C_j (either of the SAT segment or of the UNSAT segment).

We shall show that any shallow layout has to be canonical, and that a canonical layout of G_ξ can be shallow only if the BM4SAT formula ξ is a *yes* instance.

In a bucket layout, let Y-buckets denote the set of buckets to which the subgraph Y contributes to. In all cases of interest Y will be a connected subgraph, and in this case Y-buckets will be consecutive in every layout.

Lemma 11 *Let D be a connected subgraph which is a concatenation of brushes of length at least λ and of holes, but no paths (expect for the paths of length λ that are parts of holes), and let B be a brush of length at least δ. Then in any shallow layout, B cannot be spanned by D.*

Proof: A brush B_δ occupies at least $\delta/2$ buckets (Lemma 8). Any brush appearing in D can overlap at most ℓ of the B-buckets (Lemma 7), and hence cannot overlap any of the middle $\delta/2 - 2\ell$ buckets of the B-buckets. It follows that these buckets must be spanned by a path. The longest path in D is of length $\lambda \ll \delta$, and is two short to span these buckets. □

The following lemma establishes the structure of the active region in a shallow layout.

Lemma 12 *In a shallow layout*

a. *The brushes B_α of the two folding gadgets do not overlap.*

b. *The segment* UNSATMSAT *lies between the buckets to which the brushes B_α contribute to, and can overlap with each of these brushes by at most δ buckets (rightmost buckets of the left folding gadget and leftmost buckets of the right folding gadget).*

c. *Consider a brush B_γ appearing in* UNSATMSAT*. The subgraph of* UNSATMSAT *to the right of B_γ can overlap at most δ rightmost buckets among B_γ-buckets. Likewise, the subgraph of* UNSATMSAT *to the left of B_γ can overlap at most δ leftmost buckets among B_γ-buckets.*

d. *Consider two consecutive hole groups in* UNSATMSAT*, and recall that they have a brush B_γ between them. There must be at least $\frac{\gamma}{2-\epsilon} - 2\delta$ buckets (occupied by vertices from B_γ) that separate between the two hole groups.*

Proof: The subgraph B_αUNSATMSATB_α is composed only of brushes of length at least δ and holes, and form an example of a graph of type D as defined in Lemma 11. Any δ consecutive backbone vertices from a brush B_α or from a brush B_γ in UNSATMSAT can serve as the brush B in Lemma 11. A moment's reflection shows that Lemma 12 follows in a straightforward way from Lemma 8 and Lemma 11. Details omitted. □

We now concentrate on the layout of the variable segments and how they interact with the formula segments in a shallow layout.

Lemma 13 *In a shallow layout*

a. *Every variable segment v_1, v_2, ..., v_{2n} is spanned by B_α*UNSATMSAT*B_α.*

b. *For every $i \in \{1, 2, 3\}$, every key of type K_i fits a hole of type H_i, and no two keys fit in the same hole.*

Proof: The brushes B_α in each of the two folding gadgets occupy at least $\frac{\alpha}{2-\epsilon}$ buckets (Lemma 8). Every path in a variable segment is of length at most β (+ some low order terms), and $\beta < \epsilon\alpha$. Each variable segment contains also brushes of size $\lambda \gg \ell$ (as part of the keys). Lemma 7 then implies that a variable segment cannot span B_α.

Only one possibility remains, which is to use the paths P_α of the folding gadgets in order to cross over the B_α brushes, and then all variable segments lie in the active region. In this case, every variable segment is spanned by B_αUNSATMSATB_α as desired. Observe that it is not possible to have all variable segments to the left of the left folding gadget (or to the right of the right folding gadget), because then a path of length $\alpha + \beta$ must go over at least $2\frac{\alpha}{2-\epsilon}$ buckets (so as to close the ring).

Now we show part (b) of Lemma 13. The formula segments (ie., UNSATMSAT) is composed of hole groups separated by brushes of size γ. A tooth must lie in a dent (while possibly occupying at most ℓ buckets of any brush, see Lemma 7). Furthermore, the number of keys of type i is exactly equal to the number of holes of type i. As a key K_3 cannot fit either H_2 or H_1, it must fit a hole H_3 (item (b) in Lemma 9). A filled hole H_3 cannot fit another K_3 (item (c) in Lemma 9). Hence, all H_3 holes are filled. Similarly, a key K_2 cannot fit a filled hole H_3 or a hole H_1 (items (c) and (b) respectively in Lemma 9). Hence it must fit a hole H_2. Continuing as above we see that all H_2 holes must be occupied and that the keys of type K_1 must occupy all holes of type H_1. □

Lemma 14 *In a shallow layout*

a. *The structural key K_2 of v_i^j lies in the corresponding structural hole group of C_j in either SAT or UNSAT segment.*

b. *The functional key K_1 of v_i^j (if present) lies in the corresponding functional hole group of C_j in either SAT or UNSAT segment.*

c. *A variable segment fits all its keys to only one of the formula segments (either SAT or UNSAT, but not both).*

Proof: We argue inductively. Each v_i has a key of type K_3 which must fit in one of the holes in M, as shown in Lemma 13.

a. We show that the structural keys K_2 in each v_i^j must lie in the corresponding hole groups $(H_2)^n$. Consider the first structural key K_2 corresponding to C_1. This key either fits a hole group in SAT or UNSAT. Without loss of generality, assume that it fits a hole group in SAT. This structural key is separated by a path of length $2nh_3 + 2\gamma + nh_2$ from K_3. As $\gamma > O(n\delta/\epsilon)$ this path cannot span to the next set of holes of type H_2 (or later) as then a path of length 2γ (+ low order terms) will have to span over $4\frac{\gamma}{2-\epsilon} - 8\delta$ buckets (item (d) of Lemma 12) which is impossible since $\delta \ll \epsilon\gamma$. Hence, the first K_2 must lie in the first set of holes of type H_2. This argument is valid for all v_i. As all holes fit exactly one key of the same type and C_1 has n H_2 holes, exactly n variables are laid out to the right and the rest to the left. Arguing inductively, we assume that all the H_2 holes in $C_{j<k}$ are filled, and the key K_2 of v_i^{k-1} lies in the hole group of C_{k-1}. The key K_2 of v_i^k must lie in the hole group of C_k as the path separating the K_2 keys in v_i^{k-1} and v_i^k cannot span to the next set of H_2 holes (the argument is the same as the argument for the base case).

b. Having established item (b), placing a functional key K_1 of v_i^j in the functional hole group of a clause C_j' with $j \neq j'$ will force the path P_γ connecting K_1 to an adjacent K_2 (or adjacent K_3 if $j = 1$) to cross a distance of at least roughly $3\gamma/2$ (by item (d) of Lemma 12), which is impossible.

c. This item follows from the previous items. A variable once folded to the left or right must span to the next set of holes and cannot span to the previous ones (as they are already filled).

□

Lemmas 12, 13 and 14 established that any shallow layout is canonical. We can now conclude.

Lemma 15 *If G_ξ has a shallow layout then ξ is a* yes *instance of BM4SAT.*

Proof: If ξ is a *no* instance of BM4SAT then it has no satisfying assignment ie., every truth assignment of the variables either sets more than 2 variables in a clause to *true* or more than 2 variables in a clause to *false*. This translates to fitting 3 keys K_1 to fit in 2 holes H_1 or fitting 2 K_1 keys in 1 H_1 hole in either SAT or UNSAT segment. This is a contradiction to item (c) of Lemma 9. □

We can now summarize the proof of Theorem 1.

Proof: We reduce the BM4SAT instance ξ to a ringed caterpillar of hair length at most 1 as in Section 3.3. This graph G_ξ has all the properties listed in Theorem 6. To summarize the graph G_ξ has $poly(n)b$ vertices. If ξ is a *yes* instance of BM4SAT then the bandwidth of G_ξ is $b + O(n)$ (Section 3.4). Furthermore, if ξ is a *no* instance of BM4SAT then there is no shallow layout for G_ξ ie., all layouts have bucketwidth larger than $(2-\epsilon)b$. In this case the bandwidth is larger than $(2-\epsilon)b$ (Lemma 4). □

The prove of Theorem 1 easily generalizes to some other simple families of graphs.

Corollary 16 *The bandwidth problem on clique cycle graphs and unit circular arc graphs is NP-hard to approximate within a factor of $(2-\epsilon)$ for any constant $\epsilon > 0$.*

Proof: The graph G_ξ from Section 3.3 can be converted to a clique cycle graph or a unit circular arc graph preserving the rest of the properties listed in Theorem 6. To convert the ringed caterpillar with strand length 1 to a clique cycle graph, join by an edge every two strand vertices that are connect to the same heavy vertex. The new graph by definition is a clique cycle graph. This conversion has essentially no effect on the proof of hardness. To convert the clique cycle graph to a unit circular arc graph do the following. In each clique, rather than have one clique vertex as the connecting vertex (the original backbone vertex for the ringed caterpillar), have two such vertices – one for the connection in the clockwise direction and one for the connection in the anti-clockwise direction. Again, this conversion has essentially no effect on the proof of hardness. □

3.6 Tightness of the hardness results

In [22] it is shown that the bandwidth of circular arc graphs can be approximated within a factor of 2. The three simple families of graphs that we considered here are all special cases of circular arc graphs. For unit circular arc graphs this is obvious. For ringed caterpillars of hair length 1, the backbone vertices form intervals that overlap only this intervals of the neighboring backbone vertices. For every heavy vertex, the associated strand vertices form small disjoint intervals that are contained in the interval of the backbone vertex. For clique cycle graphs, the clique vertices form small intervals that overlap and are contained in the interval of the respective backbone vertex.

To make our paper self contained, we sketch a simple argument that explains how an approximation ratio of 2 comes out naturally for the special case of clique cycle graph. Let G be a clique cycle graph with cliques $C_1, \ldots, C_n$. The layout of a graph of this kind is shown in the Fig 13 and 12. Each clique is laid out in the following manner. We put the vertices of the clique before the vertex on the cycle.

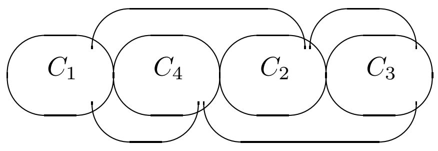

Figure 12: The layout for a even clique cycle graph

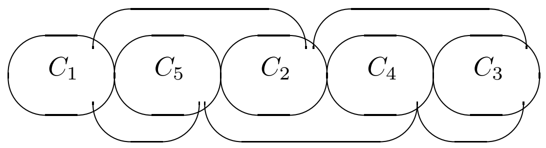

Figure 13: The layout for a odd clique cycle graph

The bandwidth of the graph is at least the maximum clique size of the graph (-1). Also, the bandwidth of the layout given above is at most $2\max_i |C_i|$. Hence, if the layout is denoted as f then $b(f) \le 2b_{opt} + 2$. If the maximum clique size (and hence the bandwidth) is a constant then the bandwidth can be calculated exactly [30]. For non-constant clique size $\frac{2}{b_{opt}}$ is subconstant and hence the approximation factor of this simple algorithm is 2, up to low order terms. As Corollary 16 shows, even this very simple algorithm has a nearly best possible approximation ratio.

4 Caterpillars

We now show the hardness of approximating the bandwidth of caterpillars. We remind the reader that we actually show the hardness of approximating the bucketwidth. As we will be able to make the hardness ratio quite large and the gap between bandwidth and bucketwidth is at most a factor of 2, the hardness for bandwidth remains of the same nature.

4.1 Recursion

To obtain hardness of approximation results within large ratios, we shall use recursion. That is, we shall construct a sequence of caterpillars $G_1, G_2, \ldots$. For every i, the number of vertices in G_{i+1} will be larger than the number of vertices in G_i by a multiplicative factor of $O(n^{O(1)} i^{\Theta(i)})$. Hence for $i < n$, caterpillar G_i will have $n^{O(i^2)}$ vertices. For simplicity we assume that G_1 is a brush of length 1, independent of the input formula.

Our construction will be recursive in the sense that for every i, G_{i+1} will have the same general structure as G_i but on a larger scale, and furthermore, G_{i+1} will contain several copies of G_i embedded within some of its brushes.

Let us explain now the effect of embedding a copy of G_i in a brush. Let g_i denote the length of the backbone of G_i. The total number of vertices in G_i will be roughly $g_i b$. On *yes* instances, there will be a layout of G_i in $g_i(1 - o(1))$ buckets (the loss of $o(1)$ factor is the result of folding of the backbone, and the reason why the loss is only an $o(1)$ factor is that all folding events take place in a small portion of the backbone), and hence the average number of vertices per-bucket will be $b(1 + o(1))$. On *yes* instances, indeed every bucket will have $b(1 + o(1))$ vertices, which is similar to the number of vertices per bucket is a natural layout of a brush. Hence G_i embeds naturally within brushes. On *no* instances, the average number of vertices per bucket will still be roughly b, but it will become unavoidable that some buckets have significantly more vertices ie., $\Omega(bi)$. Hence on *no* instances copies of G_i may be thought of as brushes that contain some irregularities (certain portions correspond to buckets with too many vertices, other portions may correspond to buckets with too few vertices).

To understand how the approximation ratio behaves as a function of the recursion level i we introduce a notion of a k-pile. At this stage the reader may think of a k-pile in a bucket arrangement as a bucket that contains k heavy vertices. (Later, when we introduce gadgets for folding, this definition will be revised.)

On *yes* instances we will maintain the property that throughout all levels of the recursion, every G_i will have a bucket arrangement in which no bucket contains more than one heavy vertex. Hence we will only have 1-piles. On *no* instances, we will maintain the property that in any bucket arrangement, G_i must have an i-pile. To carry on this property inductively, we shall embed copies of G_i in the keys of G_{i+1} (and elsewhere). Now if a key and hole of G_{i+1} are misaligned, it would force a copy of G_i in the key to lie in a region that is a brush in the hole, and hence one more heavy vertex will be added to the i-pile of G_i, making it an $i + 1$ pile.

4.2 A simple folding gadget

We now explain how one can force the backbone of a caterpillar to fold (a *structural* fold). Recall that earlier we introduced the notions of light and heavy backbone vertices. We now introduce a *folding vertex*. This is a backbone vertex with b strands attached to it. Each of these strands is very long (to be specified in Section 4.9).

Assume for simplicity that in the caterpillar G_{i+1} we have a formula segment immediately followed by a variable segment. As explained in Section 2.6, we would like to force the caterpillar to fold so that the formula segment and the variable segment both share the same set of buckets. We can do so by placing a folding vertex in between. In addition, in each one of the segments we embed copies of G_i, which on *no* instances are known to include i-piles.

Assume that the caterpillar is laid out without folding. Then there are i-piles on both sides of the folding vertex. Regardless of which side a long strand is dropped to, it will hit an i-pile. (If the long strands are long enough, then keeping all their vertices in buckets close to the folding vertex is not an option, because then the bucketwidth will be too large. Hence they have to extend a long distance either to the right or to the left.) As there are b long strands and only two sides, some i-pile will receive $b/2$ strand vertices (this will be revised in Section 4.3). If we think of $b/2$ strand vertices as being the equivalent of one heavy vertex, then this i-pile becomes an $(i + 1)$-pile.

Hence to avoid creating an $(i + 1)$-pile, one must fold the caterpillar roughly at the location of the folding vertex. Then one can drop all long strands to one side, and put both segments on the other side.

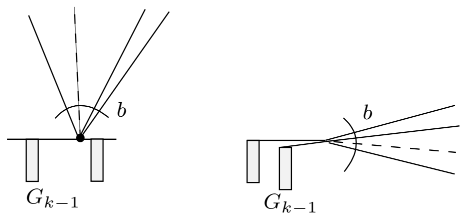

Figure 14: A folding vertex and a possible bucket arrangement with small bucketwidth. Assume that the paths are long.

4.3 Piles

Having introduced the notion of folding vertex, we are now ready to formally define the notion of a pile. A k-pile is a bucket that contains at least k *hits*. A *hit* is one of the following:

1. A *heavy hit*: a heavy backbone vertex.
2. A *folding hit*: at least $b/3$ strand vertices whose strands are all attached to the same folding vertex.

Let us elaborate on the definition above. It is our intention that for *yes* instances the corresponding caterpillar will have a bucket arrangement that has only 1-piles (and 0-piles), but no i-piles for $i \geq 2$, and that for *no* instances, for every $i \geq 1$, the corresponding caterpillar G_i will have i-piles. Our definition of hits implies that if a bucket arrangement contains a k-pile, the bucketwidth is $\Omega(kb)$, which is the conclusion that we wish to reach for *no* instances. However, note that the definition does not imply that if there are only 1-piles, the bucketwidth is $b(1 + o(1))$. For example, in a certain bucket arrangement it might be that with respect to each of k different folding vertices, a bucket contains $b/4$ strand vertices that are connected to the folding vertex. This forces bucketwidth of $\Omega(kb)$ without creating even a single folding hit. Hence the definition by itself does not suffice when treating *yes* instances. However, it will be the case that in the bucket arrangement for *yes* instances, for every bucket and for every folding vertex, a folding hit will correspond to exactly b strand vertices placed in the bucket, and the absence of a folding hit will correspond to 0 strand vertices placed in the bucket. Hence for our particular bucket arrangements for *yes* instances, it will be the case that having only 1-piles will imply a bucketwidth of $(1 + o(1))b$.

4.4 The key lemma

Having introduced the notion of a k-pile, we can state our main lemma.

Lemma 17 *Given an instance of BM4SAT with* $2n$ *variables, for every positive integer* k *one can construct in time* $O(N_k)$ *a caterpillar* G_k *with* $N_k \leq n^{ck^2}$ *vertices, where* c *is some universal constant.*

1. *On* yes *instances, G_k has a bucket arrangement with bucketwidth $b + O(nk)$.*
2. *On* no *instances, every bucket arrangement of G_k has a k-pile.*

Theorem 2 will follow from Lemma 17 by choosing b to be much larger than nk. On *yes* instances the bucketwidth will then be essentially b, up to low order terms. On *no* instances, the existence of a k-pile implies that there is a bucket such that this bucket, possibly together with the two buckets on either side of it, must contain $\Omega(kb)$ vertices. Hence the bucketwidth has to be $\Omega(kb)$. This gives a mutiplicative gap of $\Omega(k)$ between the bucketwidth of *yes* and *no* instances.

Lemma 17 will be proved by induction on k.

4.5 Structural keys and holes

Here we explain how to construct different types of key-hole pairs. The types of key-hole pairs that are presented in this section are the *structural* ones, whose purpose is to force the caterpillar to fold in certain ways, regardless of whether the caterpillar corresponds to a *yes* instance or a *no* instance. The purpose of having several types of key-hole pairs is so that each key can fit a hole of its own type, but cannot fit a hole of a different type. (The notion of *fit* will be made precise later.)

The number of different types of key-hole pairs will depend on the level of the recursion we are in. We now explain how to construct key-hole types for graph G_k. The number of key-hole types will be $O(k)$, and for concreteness let us assume that it is $2k$ (which is more than we need). Each key will start after a very long path, and will contain a sequence of *teeth* (the number of teeth depends on the key type) that are separated by paths, and will be followed by another long path. Each tooth of a key is a sequence of copies of G_{k-1} (the number of copies depends on the key type). The hole that matches a key is a dual of the key in the sense that every path is replaced by a brush of the same length, and every tooth is replaced by a path of the same length (that we call a *dent*). Hence there is a natural bucket arrangement in which the key and hole of the same type are aligned, and every bucket is either a 1-pile, or a pile of the same type that it was in G_{k-1}. We now explain how to construct the pattern of teeth in different key types so that a key does not fit in a hole of a different type.

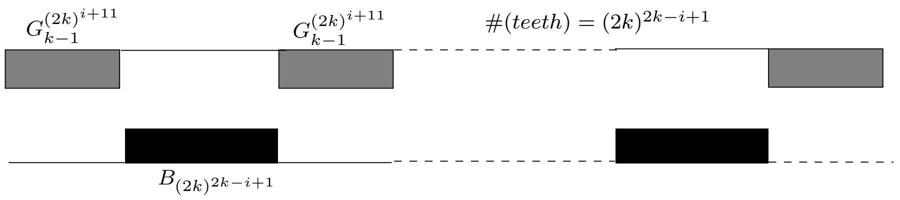

Figure 15: The basics structure of a structural key-hole pair of the same type ie., (K_i, H_i).

In general, the idea is that keys of the lower types will have more teeth, whereas keys of the higher types will have wider teeth. Hence keys of the lower types cannot fit holes of a higher type, because these holes do not have enough dents, and keys of a higher type cannot fit holes of a lower type because the dents are too small. For concreteness, we choose the following parameters. For every $i \in \{1, \ldots, 2k\}$, the number of copies of G_{k-1} in a tooth of a key of type i is $(2k)^{i+11}$. The number of teeth in a key of type i is $(2k)^{2k-i+1}$. The length of a path between two teeth in key of type i is $(2k)^{i+9}g_{k-1}$. Hence the largest key (key of type 1) has less than $k^{ck}g_{k-1}$ backbone vertices, for some constant c.

A key K_i *fits* a hole H_i in the sense that there is a bucket arrangement for K_i (the natural one, placing every new backbone vertex in a new bucket) and a corresponding bucket arrangement for H_i (the natural one, starting at the same bucket as the bucket arrangement for K_i) in which every tooth of K_i is spanned by a dent of H_i. As a result, the level of pileness of this bucket arrangement is the same as that of G_{k-1} (copies of which compose the teeth).

We say that a key K_i *cannot fit* a hole H_j if in any bucket layout in which some bucket contains at least one vertex from one tooth of K_i and at least one vertex from one dent of H_j it must be the case that the level of pileness of the layout increases (compared to the pileness of a bucket layout for G_{k-1}).

To show that a key K_i cannot fit a hole $H_{j \neq i}$ we make the following verifiable statements regarding the construction and its implications.

a. Holes are constructed with simple brushes and paths (dents). The length of a brush in hole H_j is $(2k)^{j+9} g_{k-1}$. This brush lies in at least $2(2k)^{j+8} g_{k-1}$ consecutive buckets or a k-pile is created.

b. The length of every tooth in key K_i is $(2k)^{i+11} g_{k-1}$. As G_{k-1} cannot be spanned by a simple brush, a brush from H_j can have an intersection of at most g_{k-1} buckets with a tooth of K_i. This implies that most part of a tooth (except for g_{k-1} buckets at either end) must be spanned by a dent.

c. The length of paths in key K_i is $(2k)^{i+9} g_{k-1}$. The simple brushes in H_j occupy at least $2(2k)^{j+8} g_{k-1}$ buckets. If $j > i$ then if one tooth of K_i is spanned by a dent of H_j then all teeth corresponding to K_i must be spanned by the same dent of H_j.

Consider now what happens if one vertex of a tooth of a key K_i is in the same bucket as a vertex of a dent of H_j (and $j \neq i$). If $i > j$ then every tooth of K_i is at least a factor of $2k$ larger than every dent of H_j. As implied by (b) above, most part of the tooth must be spanned by a dent. This requires $(2k)^{i+11}$ copies of G_{k-1} to be placed in at most $(2k)^{i+10} g_{k-1} + 2g_{k-1}$ buckets, causing by a local density argument a bucketwidth greater than kb. (Recall that the number of vertices in G_{k-1} is roughly $g_{k-1}b$. See Sections 4.1 and 4.10.)

If $i < j$, then all teeth of K_i must lie in a single dent of H_j (by (c) above). The length of a dent in H_j is $(2k)^{j+11} g_{k-1}$ and the length of all tooth put together in K_i is $(2k)^{2k-i+1}(2k)^{i+11} g_{k-1}$, which is $(2k)^{2k+12} g_{k-1}$. The largest dent (for $j = 2k$) is of size $(2k)^{2k+11} g_{k-1}$. The size of all teeth put together is a factor $2k$ larger than the size of a single dent, again leading to bucketwidth greater than kb.

Summarizing, we have the following proposition.

Proposition 18 *For every k the above construction of $2k$ key-hole pair types has the following properties:*

1. *Every key (or hole) has backbone length at most $k^{ck} g_{k-1}$, where g_{k-1} is the length of the backbone of G_{k-1}.*

2. *On* yes *instances (where G_{k-1} has at most 1-piles), if in a bucket arrangement a key is placed in a hole of the same type, then no t-pile with $t > 1$ is created by this.*

3. *On* no *instances (where G_{k-1} has $(k-1)$-piles), if in a bucket arrangement a vertex of a tooth of a key is placed at the same bucket as a vertex of a dent of a hole of a different type, a k-pile is created.*

4.6 Stacks

Before describing the construction of functional keys and holes, we present here a gadget that will be used in their construction. This gadget is an alternating sequence of G_{k-1} and a simple brush of size Δ. We use X to denote $G_{k-1}B_\Delta$, and then the gadget is X^l_Δ for some $l > 1$. We denote the length of the backbone of G_{k-1} by g_{k-1}. Hence, in any bucket arrangement, the backbone of G_{k-1} can contribute to at most g_{k-1} consecutive buckets. The value of Δ will typically be larger than kg_{k-1}.

To remind the reader, a gadget H *spans* a gadget W in a bucket arrangement if the backbone of H contributes to all buckets to which the backbone of W contributes. A simple observations is that if G_{k-1} has a $(k-1)$-pile and if it is spanned by a simple brush, a k-pile is created.

If two or more copies of G_{k-1} contribute to the same bucket in a linear arrangement then a *stack* is formed. The number of G_{k-1} contributing to this bucket is called the *height* of the stack. See Fig 16 for a pictorial definition.

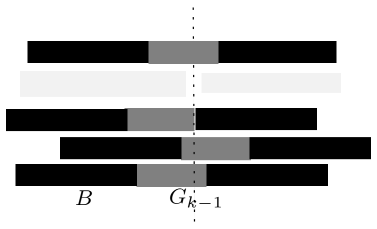

Figure 16: A stack. The dotted line represents the intersecting bucket.

Lemma 19 *If $\Delta \geq 2kg_{k-1}$ and G_{k-1} forms a $k-1$-pile, then in any bucket arrangement with no k-pile, for any two consecutive G_{k-1} in the gadget X_Δ^l defined above one of the following happens: either they share a common bucket forming a stack or they lie at least $\frac{\Delta}{k} - 2g_{k-1}$ buckets apart.*

Proof: A simple brush of length Δ contributes to at least $\frac{\Delta}{k}$ consecutive buckets. G_{k-1} contributes to at most g_{k-1} consecutive buckets. A simple brush cannot span G_{k-1}. Hence G_{k-1} must lie on one side of the buckets in which B_Δ contributes to (with an intersection of at most g_{k-1} buckets on the left or right end). If any two consecutive G_{k-1} do not form a stack then they are a distance $\frac{\Delta}{k} - 2g_{k-1}$ buckets apart. □

Lemma 20 *If the bucketwidth of a bucket arrangement is smaller than kb, then the height of every stack is at most $2k$.*

Proof: This follows from a local density argument. We remind the reader that the number of vertices in G_{k-1} is roughly $g_{k-1}b$ (mentioned in Section 4.1 and proved in Section 4.10). As all G_{k-1} in a stack share at least one common bucket (call it t), they all lie in the buckets numbered $[t - g_{k-1}, t + g_{k-1}]$. This means that the bucketwidth is at least $\frac{2kg_{k-1}b}{2g_{k-1}}$ or kb. □

Lemma 21 *Consider two gadgets $L_1 = X_\Delta^{l_1 \geq 4k}$ and $L_2 = X_{2k\Delta}^{l_2 \geq 4k}$ with $\Delta > 4g_{k-1}$. Then in any bucket layout with no k-pile, neither gadget can span the other.*

Proof: The stacks in L_2 are at least bucket distance $2\Delta - 2g_{k-1}$ apart (from Lemma 19). Also, each gadget forms at least two stacks. The distance between consecutive stacks in L_1 is at most $\Delta + 2g_{k-1}$ (the distance between the consecutive G_{k-1} is an upper bound), while in L_2 it is at least $2\Delta - 2g_{k-1}$. Hence for one of the gadgets to span the other a copy of G_{k-1} in L_1 needs to be spanned by a brush of L_2, creating a k-pile. □

We summarize the results in this section.

Proposition 22 *If G_{k-1} has a $(k-1)$-pile and a bucket arrangement has no k-piles then*

- *A simple brush cannot span G_{k-1}.*
- *All stacks in the bucket arrangement have height $< 2k$.*

- *If $\Delta > 2kg_{k-1}$ then two consecutive G_{k-1} in a gadget $X^l_\Delta = (G_{k-1}B_\Delta)^l$ either form a stack or are at least $\frac{\Delta}{k} - 2g_{k-1}$ buckets apart.*
- *Neither one of two gadgets $X^{l_1 \geq 4k}_{\Delta>0}$ and $X^{l_2>4k}_{2k\Delta}$ can span the other if $\Delta > 4g_{k-1}$.*

4.7 Functional keys

We now describe the key-hole pairs that we call *functional*. The decision of in which hole to place a functional key will roughly correspond to a decision of whether to set a variable in the BM4SAT formula to true or false.

For functional holes (which we will denote by H), we wish to design four different subtypes of functional keys (K^1 up to K^4; note that here, a superscript of {1,2,3,4} does not represent repetition but rather the subtype of functional key) with the following properties:

1. Each of K^1, K^2, K^3, K^4 by itself fits H. This will be immediate from the construction below.
2. Each of K^1, K^2, K^3, K^4 does not fit any structural hole (of Section 4.5). See item c below.
3. No structural key (of Section 4.5) fits in hole H. See item d below.
4. Two keys K^p and K^q for $1 \leq p < q \leq 4$ do not fit together in H. See Lemma 24.

Property 4 is the one that makes this new type of keys special. Without it we could make $K^1 = K^2 = K^3 = K^4 = K$ and the new key-hole pair would be no different than the ones that we introduced in Section 4.5. The reason why we will need four subtypes of keys is because every clause in BM4SAT has four variables.

All four subtypes of keys follow the same high level pattern. They are separated from other keys by long paths on either side. Each subtype of key starts with a *wide tooth* of length $g_{k-1}(2k)^{11}$, followed by a path of length $g_{k-1}(2k)^5$, and then an alternating pattern of $(2k)^{2k+3}$ teeth $(G_{k-1})^l$ (for $l = g_{k-1}(2k)^9$) and paths $P_{g_{k-1}(2k)^4}$. The difference between the four subtypes of keys is in the composition of the wide tooth. The wide tooth of $K^{p\in\{1,2,3,4\}}$ is the gadget $X^l_{\Delta=(2k)^p g_{k-1}}$ with $l = (2k)^{11-p}$ ie., of size $(2k)^{11}g_{k-1}$.

We now describe the hole H as it appears in caterpillar G_k. As always, the hole is separated from other holes by long brushes on either side of it. The hole H starts with a wide dent (path) of length $(2k)^{11}g_{k-1}$. This is followed by a brush of length $(2k)^5 g_{k-1}$. After that comes a pattern of $(2k)^{2k+3}$ alternating paths (dents) and brushes, of size $(2k)^9 g_{k-1}$ and $(2k)^4 g_{k-1}$ respectively.

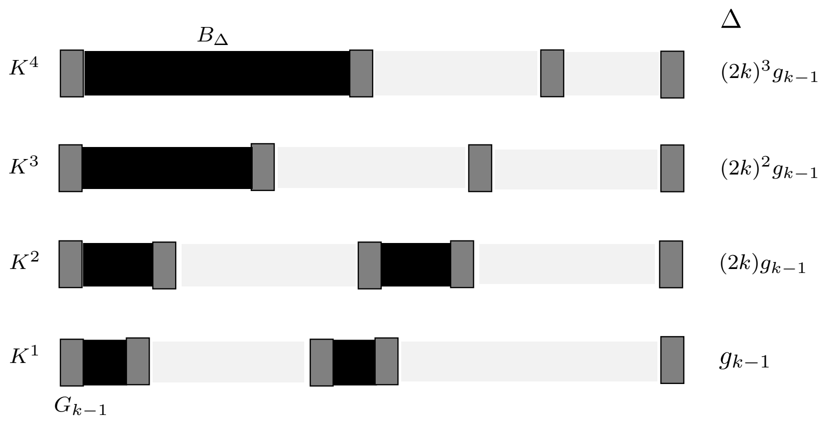

Figure 17: Wide tooth in different types of key K

Proposition 23 *For every k, the above construction of functional keys has the properties (1) to (4) listed above.*

To prove the above proposition, we make observations similar to the ones in Section 4.5.

a. Holes are constructed with simple brushes and paths. The length of a brush in hole H_j is $(2k)^5 g_{k-1}$. This brush lies in at least $2(2k)^4 g_{k-1}$ consecutive buckets or a k-pile is created.

b. Keys contain teeth composed of $(G_{k-1})^l$ and paths. As G_{k-1} cannot be spanned by a simple brush, a brush from H_j (defined in Section 4.5) or a brush from H can have an intersection of at most g_{k-1} buckets with a tooth of K. This implies that most parts of a tooth (except at most g_{k-1} buckets on either side) must be spanned by a dent. This observation is termed as *a tooth must lie in a dent.*

c. We now show that K does not fit any structural holes defined in Section 4.5. There are two types of paths in the key K. The first is a single path of length $(2k)^5 g_{k-1}$ appearing after the wide tooth. The second is $(2k)^{2k+3}$ paths of length $(2k)^4 g_{k-1}$. The brushes in H_j are much longer than both of these paths. Hence all teeth of K must lie in the same dent of H_j for any j. The largest dent corresponding to H_{2k} is $(2k)^{2k+11} g_{k-1}$. On the other hand, all teeth of K put together contribute $g_{k-1}(2k)^9 \times (2k)^{2k+3}$ or $g_{k-1}(2k)^{2k+12}$, which is a factor $2k$ larger. Hence, a k-pile will be created.

d. Every tooth of K_i is much larger than any dent in H. As a tooth must fit a dent, a k-pile will be created. Hence no structural key defined in Section 4.5 fits a hole H.

Lemma 24 *Two functional keys K^p and K^q with $p \neq q$ do not fit in the same hole H.*

Proof: For K^p to fit in the hole H, the wide tooth must lie in the wide dent (and likewise for K^q). The wide tooth of width $(2k)^{11} g_{k-1}$ is followed by a comparatively small path of size $(2k)^5 g_{k-1}$. This implies that either the wide tooth ends close to the end of the wide dent, or also the first tooth of the key is placed in the wide dent. If the first case applies both to K^p and K^q, there will be a large intersection between the bucket layouts of the two wide teeth. But Lemma 21 implies that only small parts of the wide teeth can span each other (as their Δs are at least a factor of $2k$ apart), leading to a contradiction. Hence the second case applies, and then the first tooth of (say) K^p is placed in the wide dent (in addition to the wide teeth). Since the length of path between neighboring teeth is a factor of $2k$ shorter than the brush between the wide dent and the first dent, this implies (if no k-pile is created) that all teeth of K^p must be placed in the wide dent. But their total size is much larger that the size of the wide dent, again forcing bucketwidth greater than bk. $\square$

4.8 Forcing the main structural folds

This section is a modified version of Section 3.2 that applied to ringed caterpillars.

The high level structure of a caterpillar G_k will be as follows. It will have two formula segments, the SAT segment and the UNSAT segment. They will be followed by $2n$ variable segments. In the corresponding bucket arrangement, we call the buckets used by the formula segments the *active region.* We will need to force a fold between the formula segments and the variable segments so that the variable segments will also fall in the active region. In fact, each variable segment will be similar in length to a formula segment, and hence we shall need to enforce additional folds so as to prevent the variable segments from escaping from the active region to either side.

To force folding, we shall place a folding vertex between the UNSAT segment and the first variable segment. To force subsequent variable segments from extending beyond the beginning of the SAT segment, we shall place another folding vertex just before the SAT segment.

Proposition 25 *(see Section 4.11) For* no *instances, either all variable segments of G_k are in the active region or a k-pile is created.*

We need each of the $2n$ variable segments to start in between the SAT and UNSAT segments. It can then be laid out either to the left which corresponds to setting the variable to *false* or to the right which corresponds to setting it to *true*. Recall that in Section 3.2 we introduced a *middle segment* for this purpose, between the SAT and UNSAT segments. It was formed by a sequence of as many holes of a special type as the variables in the BM4SAT instance. Each variable segment starts with a key of this special type. This insured that each variable segment starts at the middle segment.

The caterpillar G_k will also contain a middle section, similar to the ringed caterpillars of Section 3.2. However, we shall in addition introduce another set of special keys (and corresponding holes) that are placed at the end of variable segments (specifically, as part of their mth subsegment), which force the variable segments to stretch all the way until nearly the folding vertex. Having introduced these additional special keys, it is not clear that the middle segment plays a significant role in our reductions, but nevertheless we keep it as part of G_k as it aids intuition.

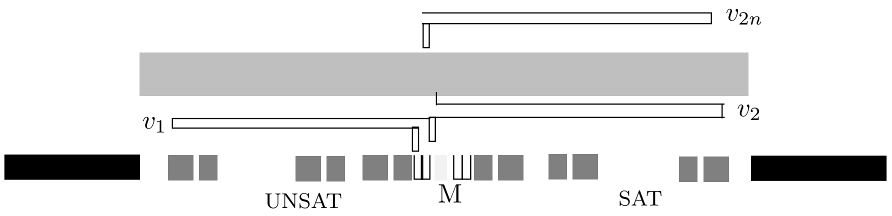

Figure 18: (a) Formula segments lie between folding vertices (the strands attached to the folding vertices are represented in black). (b) Variable segments lie in the active region. (c) M is the middle segment.

4.9 The full reduction

For notational convenience, we define a function $p : \mathrm{N} \to \{0, 1, \ldots, k+5, \infty\}$.

$$p(j) = \begin{cases} j \bmod (k+5) + 1 & \text{if } 0 < j < m \\ \infty & \text{if } j = m \\ 0 & \text{if } j = 0 \end{cases}$$

When j is clear from the context we will denote $p(j)$ by p. The construction of G_k will require $k+7$ types of structural key hole pairs. In Section 4.5 we explained how to construct $2k$ such types, which suffices whenever $k \geq 7$. (The case $k < 7$ can simply be treated as if $k = 7$. Details omitted.) We choose arbitrarily $k+7$ types of structural key hole pairs from Section 4.5, and for notational convenience we rename them as (H_0, K_0), $(H_1, K_1), \ldots, (H_{k+5}, K_{k+5})$, (H_∞, K_∞). Key-hole pairs (H_0, K_0) will be used in the middle segment, and key-hole pairs (H_∞, K_∞) will be used in the m-th clause subsegment. Together they will be used to enforce that each variable segment spans a formula segment (either SAT or UNSAT), starting in the middle segment and ending at the last clause subsegment). The other types of structural key-hole pairs will be used periodically throughout the formula segments. We denote the length of the backbone of H_j by h_j and the length of the backbone of G_{k-1} by g_{k-1}. Let h_{max} be the maximum backbone size of the holes among $H, H_0, \ldots, H_\infty$ (recall that H denotes the functional hole).

We shall use the following two parameters that signify a very large value $\alpha = 4n^2(2k)^{10}h_{max}$ and an extremely large value $\ell = 10kn^2\alpha$. (The exact choices for the values of α and ℓ are somewhat arbitrary, and are meant to convey a tradeoff between being very large but still polynomially bounded in the other parameters.)

Empty Clauses: For notational convenience, we augment the set of clauses by $k+5$ new empty clauses (with no variables). As such, the subgraphs that encode them will have structural holes but no functional holes. We name the empty clauses $C_1, \ldots, C_{k+5}$, and hence will need to add $k+5$ to the index of the original clauses. Though the new number of clauses is $m+k+5$, we shall keep notation simple and rename this number as m. The reason for adding the empty clauses will become clear later (see the paragraph after Lemma 34).

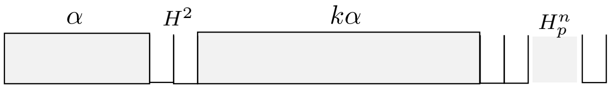

Figure 19: Clause subsegment

Clause Subsegment: The clause subsegment contains n structural holes and two functional holes (or none for the empty clauses). The type of functional hole depends on the index of the clause. For clause C_j with $j > k+5$, the subsegment is $B_\alpha H^2 B_{k\alpha} H_p^n$. See Fig 19. For empty clauses the corresponding clause subsegment is $B_\alpha B_{2h} B_{k\alpha} H_p^n$ (the functional holes are replaced by brushes). The clause subsegment for the last clause C_m is special. Very importantly, its structure is different in SAT and UNSAT segments. The $B_{k\alpha}$ brush in C_m is replaced by $(X_\Delta)^l$ where $l = \frac{k\alpha}{\Delta + g_{k-1}}$ (see Section 4.6 for the notation X). For the subsegment of C_m in the SAT segment we call this construct E_s (E for "end" and s for "sat"), and the value of Δ is $\frac{\alpha}{(2k)^4}$. For UNSAT we call it E_u and the value of Δ is $\frac{\alpha}{(2k)^3}$. We denote the clause subsegment C_m in the SAT segment C_m^s and in the UNSAT segment C_m^u.

Sat: is $C_1 \ldots C_{m-1} C_m^s$.

Unsat: is the mirror image of $C_1 \ldots C_m^u$.

Middle Segment: is H_0^{2n}.

Hole Group: Holes of the same type will come in groups, called hole groups. There are $k+8$ types of hole groups, each corresponding to a different type of hole. They are H^2, H_0^{2n}, H_1^n, $\ldots, H_{k+5}^n$, and H_∞^n.

Slackness: We shall not distinguish between different holes in the same hole group. Hence for example, a key K_i will be allowed to fit any hole in the hole group H_i^n. For this reason we introduce the concept of slackness around each key, where the amount of slackness depends on the type of hole group. The slackness will be of type H, H_0, $\ldots$, H_∞. It is denoted as s, s_0, $\ldots, s_{k+5}$, s_∞ and is paths of lengths $2h, 2nh_0, nh_1, \ldots, nh_{k+5}$, and nh_∞ respectively. Each key appearing in the variable segment has the slackness of the same type before and after the key eg., K is actually sKs and K_{p_j} is actually $s_{p(j)} K_{p(j)} s_{p(j)}$. This gives us the freedom to choose an arbitrary hole in the hole group to fit the key in. For simplicity of notation we will denote sKs as K and so on.

Empty variable: Augment the set of variables by a new variable e called the *empty variable*. This variable appears in no clauses and hence has no functional keys. In fact, it will have only one structural key, of type K_0 (that fits in the middle segment).

Variable Segment: The variable segment for a single variable v starts with a key of type K_0 (intended to be laid in a hole of type H_0 is the middle segment). This is followed by m *forward* subsegments $v^1, \ldots, v^m$. (Note that here superscripts do not mean concatenation notation. Namely v_i^j is not a caterpillar v_i concatenated j times. We hope that no confusion will arise because of this notation abuse.) The variable segment ends by a *backward* path that is used to span either the SAT of UNSAT segment when returning to the middle section. Hence its length is the same as the backbone length of the SAT segment, and hence the backward path will be denoted by P_{SAT}. Altogether, the variable segment for variable v is $K_0 v^1 \ldots v^m P_{SAT}$.

Variable Subsegment: Subsegment v^j of variable v is $P_\alpha Y P_{k\alpha} K_p$. Here Y is a functional key K if v appears in C_j and P_{2h} otherwise. See Fig 20 for a pictorial definition. As there are four types of functional keys (see Section 4.7) and four variables in clause C_j, each of these four variables will

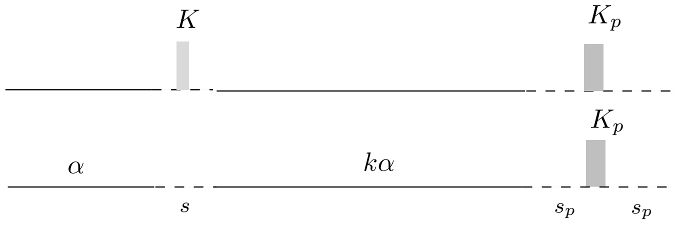

Figure 20: Variable subsegments. Bottom figure applies if v does not appear in C_j. Top figure applies if v appears in C_j. Here s is P_{2h} and s_p is P_{nh_p}, see Slackness

have a different type of functional key in its corresponding subsegment v^j. The key K_p is intended to lie in one of the holes of a hole group H_p^n in a certain clause subsegment. Recall that to make it possible for the key to lie in any of these holes (out of n possible ones), we add slackness: a path of length nh_p on both sides of K_p. For simplicity of notation K_p will mean $s_pK_ps_p$. It is important that the amount nh_p of slackness is a low order term compared to α.

Folding Vertex: is a single vertex F with b strands of length ℓ.

A summary of the various parts appears in Figure 21.

The overall structure of the graph is depicted in Figure 22.

$$\begin{array}{rcll}
G_k & = & P_\ell L\mathrm{FS}RP_{SAT}\mathrm{V}P_\ell & \ell = 10kn^2\alpha \\
\mathrm{FS} & = & \textsc{unsat}M\textsc{sat} & \\
\mathrm{M} & = & (H_0)^{2n} & \\
\mathrm{SAT} & = & C_1C_2\dots C_{m-1}C_m^s & \\
\mathrm{UNSAT} & = & (C_1C_2\dots C_{m-1}C_m^u)^{-1} &
\end{array}$$

$$\begin{array}{rcll}
C_j & = & B_\alpha H^2 B_{k\alpha} H^n_{p(j)} & X_\Delta = G_{k-1}B_\Delta \\
C_m^s & = & B_\alpha H^2 E_m^s H^n_\infty & E_m^s = X_{\Delta_1}^{l_1}, l_1 = k(2k)^4, \Delta_1 = \frac{\alpha}{(2k)^4+g_{k-1}} \\
C_m^u & = & B_\alpha H^2 E_m^u H^n_\infty & E_m^u = X_{\Delta_2}^{l_2}, l_2 = k(2k)^3, \Delta_2 = \frac{\alpha}{(2k)^3+g_{k-1}} \\
\mathrm{V} & = & v_1\dots v_{2n} & \\
v_i & = & K_0v_i^1\dots v_i^m P_{SAT} & \\
e^j & = & P_\alpha P_{2h}P_{k\alpha}K_{p(j)} & \\
v_i^j & = & P_\alpha P_{2h}P_{k\alpha}K_{p(j)} & \text{if } v_i \notin C_j \\
 & = & P_\alpha K P_{k\alpha}K_{p(j)} & \text{if } v_i \in C_j
\end{array}$$

Figure 21: (a) G^{-1} is the mirror image of G. (b) R and L are right and left folding vertices respectively. A folding vertex is a vertex with b strands of length ℓ attached to it. (c) The type of functional key K is chosen from one of four functional key types K^1,K^2,K^3,K^4, where the choices for the four variables that correspond to the same clause C_j are all different. (d) The superscripts in C_m^u, C_m^s, v^{bj} and v^j do not mean concatenation but denote various subsegments.

We conclude this section with the following Proposition concerning the size of the graph G_k.

Proposition 26 *Let g_i denote the number of backbone vertices in G_i. Then:*

1. *The diameter of G_i is g_i.*

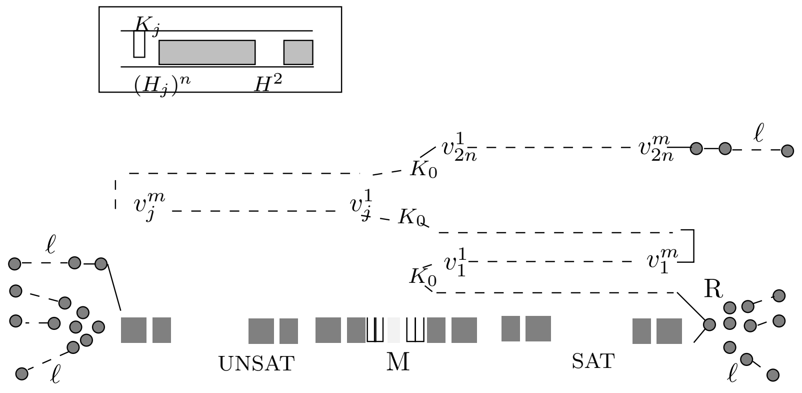

Figure 22: The full graph G_k. Also shows the layout in the *yes* instance.

2. *The number of vertices in G_i is bg_i, up to low order additive terms.*
3. $g_i = g_{i-1} n^{O(1)} i^{\Theta(i)}$.

Proof: The parameter ℓ is chosen to be sufficiently large so that the length of the backbone of G_i is roughly 2ℓ (the contribution of the two paths P_ℓ) plus low order terms. (1) The two most distant vertices of G_i are the endpoints of the backbone. (2) The total number of vertices in G_i is $2b\ell$ (from the strands of the folding vertices) plus low order terms. (3) Comparing ℓ to g_{i-1}, we see that ℓ is a polynomial in n, i times α, that α is a polynomial in n, i times h_{max}, and h_{max} is of the order of $k^{O(k)}$ times g_{i-1}. □

4.10 Layout on yes instances

Proposition 27 *If the BM4SAT instance is satisfiable then the bucketwidth of the graph G_k for all k is at most $b + O(nk)$.*

The proposition is proved by induction. For the base case, G_1 is simply a brush of length 1, and hence has bucketwidth at most b (in fact, $\lceil (b+1)/3 \rceil$). For the inductive step, assume that G_{k-1} has bucketwidth $b + O(n(k-1))$. In this case, we may think of G_{k-1} as being a essentially a brush of length g_{k-1}. (The length is slightly shorter due to folding of the backbone, but not much shorter because the major part of the backbone, the paths P_ℓ, is not folded. The number of hairs attached to each backbone vertex of this imaginary brush should be thought of a slightly larger than b, namely $b + O(n(k-1))$.)

We now describe the bucket layout of the caterpillar G_k, given a satisfying assignment to the BM4SAT formula. We shall assume for convenience that in this satisfying assignment v_{2n} is set to true. (This assumption can be made without loss of generality, as flipping all variables also gives a satisfying assignment.)

- The backbone vertices of the path P_ℓ, folding vertex L, formula segment UNSAT, middle segment M, formula segment SAT and folding vertex R are laid in order.
- The strands attached to the folding vertex R are dropped to the right, at the rate of one vertex per bucket. The strands attached to the folding vertex L are dropped to the left, at the rate of one vertex per bucket.
- The set of buckets occupied by the backbone of UNSATMSAT will be called the *active region.* In the active region, every hair vertex is placed in the same bucket as its connecting backbone

vertex. This applies to brushes, and by analogy, to copies of G_{k-1} (which we said may be thought of as brushes for the purpose of the layout).

- The backbone vertices of P_{SAT}V will all be placed in the active region, in a way to be described next. The same convention of placing hairs in the same buckets as the backbone vertex to which they are attached applies also here.

- The path P_{SAT}, whose length equals that of the backbone of SAT, is used to span SAT, going from R to M.

- V is composed of $2n$ variable segments. Variable segments of variables assigned true will be laid out to the right (over MSAT) and variables assigned false will be laid out to the left (over UNSATM). Since both cases follow exactly the same pattern, we shall describe the layout of only one variable v that is assumed to be assigned true.

- Assume that just before laying out v the bucket arrangement has reached a certain bucket x. Recall that $v = K_0 v^1 \dots v^m P_{SAT}$. Recall also that K_0 has slackness (paths of length $2nh_0$) on both sides. Pick an arbitrary yet unoccupied hole H_0 in M. Denote its starting location by y. The first path of length $2nh_0$ can be used in order to reach y from x. If y it too close, the path can overshoot the location of y and then return to y. Hence reaching y by the path can be done while placing at most two vertices in any single bucket. There after K_0 is placed in H_0. Thereafter, the other slackness path can be used as before (placing at most two vertices per bucket) to reach the first bucket of the SAT segment. Now each $v^j = P_\alpha Y P_{k\alpha} K_p$ (with Y either a functional key or a path) spans the corresponding $C_j = B_\alpha H^2 B_{k\alpha} H_p^n$. Also here, the structural key $K_{p(j)}$ can be placed in and arbitrary unoccupied hole in the structural hole group, using the slackness around $K_{p(j)}$, and likewise for the functional key if it exists. The fact that exactly n variables are set to true implies that an unoccupied structural hole can always be found, and the fact that exactly two variables within a clause are set to true implies that an unoccupied functional hole can be found (if needed). Finally, the backward path P_{SAT} is used in order to return to the middle segment M.

- For the last variable v_{2n}, there is no need to use the backward path P_{SAT} in order to return to M. In fact, this backward path is redundant and can be omitted from the description of G_k. (Alternatively, it can be "wasted" by spanning halfway along SAT and back.) The final path P_ℓ is no used to span over the ℓ buckets that contain the strands of the folding vertex R.

We observe the following for the above layout

- Each variable contributes one structural keys in each clause subsegment in either SAT or UNSAT. As exactly n variable segments are folded to each side, this brings the total to n structural keys on both SAT and UNSAT segments (per clause subsegment).

- Each variable segment contributes one key to the middle segment. This brings the total to $2n$ K_0 keys.

- The assignment is a satisfying one. Hence, each clause subsegment in SAT as well as UNSAT segment gets 2 functional keys.

- The number of keys of each type (functional and structural) equals the number of holes of each type available in the clause subsegment and the middle segment.

- Each path of V contributes at most two vertices to any single bucket. Moreover, every bucket is visited by at most $O(n)$ paths.

It follows from the above that indeed, the bucketwidth of G_k exceeds that of G_{k-1} by at most at additive term of $O(n)$.

4.11 Canonical layout for no instances

A bucket layout of G_k is called *shallow* if the layout has no k-pile and no bucket with bk vertices. We shall show that for *no* instances of BM4SAT, the corresponding caterpillar G_k does not have a shallow layout. The proof is by induction. Recall that G_k is a brush of length 1. Hence it has a heavy vertex and any layout of it has a 1-pile (a bucket with one heavy vertex).

For the inductive step, we assume that every bucket layout of G_{k-1} must contain a $(k-1)$-pile, and we need to prove that every layout of G_k must contain a k-pile. Our proof will use an informal notion of a *canonical layout*, similar in structure to the layout in figure 22). Here are properties a canonical layout must satisfy (up to taking a mirror image). These properties will be made more precise later.

i. Without loss of generality, the folding vertex R is placed to the right of the folding vertex L (or in the same bucket, an option that will disqualified later). The key property for a canonical layout is that the strands of R are folded to the right and the strands of L are folded to the left. The region between L and R will be called the *active region*.

ii. All keys of V (the variable segments) are placed in the active region.

iii. The hole groups of the segments UNSATMSAT are placed in the active region in order, one after the other. (We do not require that the holes within a hole group are also placed in order.)

iv. Keys K_0 of variable segments are placed in the hole group H_0^{2n} in M.

v. For every variable v and subsegment v^j with $j \geq k+5$, the structural key is placed in a hole in the corresponding structural hole group in C_j.

vi. If the structural key K_{j+5} lies in SAT (or UNSAT) for some variable v then all structural and functional keys of v for $j > k+5$ remain in SAT (or UNSAT, respectively).

vii. For every clause C_j and variable v contained in C_j, the functional key of subsegment v^j is placed in the functional hole group H^2 of C_j.

We will show that any shallow bucket layout must be canonical in the sense explained above. Then item (vii) will imply that if the BM4SAT instance is not satisfiable, three functional keys need to be placed in two functional holes. This will create a k-pile (by the results of Section 4.7) contradicting the assumption that a shallow layout exists.

To show that a shallow layout must be canonical, we start by considering the folding vertices L and R. Recall that a bucket suffers a folding hit if it contains at least $b/3$ vertices of strands of a folding vertex.

We call the set of buckets which suffer a folding hit *folding buckets*. We call the ones suffering the hit because of R as right folding buckets and the ones suffering the hit due to L as left folding buckets.

Lemma 28 *In any bucket layout with bucketwidth less than kb, for each of the folding vertices L and R, there is a consecutive set of at least $\ell/6k$ buckets starting at the bucket of the folding vertex that all suffer folding hits.*

Proof: Let F denote the bucket in which the folding vertex lies. We claim that either $b/3$ strands extend $\ell/6k$ to the left, or $b/3$ strands extend $\ell/6k$ to the right. Otherwise, $b/3$ strands are fully contained in $\ell/3k$ buckets, and this the length of each strand is ℓ, some bucket contains kb vertices. Assume then without loss of generality that $b/3$ strands extend $\ell/6k$ to the right. Each of these strands contributes at least one vertex to every bucket that it spans, and hence there are at least $\ell/6k$ consecutive buckets to the right of F that suffer a folding hit. □

We say that the strands of a folding vertex are folded to the right if the $\ell/6k$ buckets to the right of it suffer a folding hit, and to the left otherwise (in which case the $\ell/6k$ buckets to the right of it suffer a folding hit). Observe that under this definition, Lemma 28 implies that the strands of every folding vertex are folded either to the right or to the left (and not both). Hence the notion of direction of folding of strands in item (i) is well defined.

We now prove that the strands of R must be folded to the right and those of L to the left. First, note that one cannot fold the strands of R to the left and at the same time the strands of L to the right. This will create a consecutive block of roughly $\ell/3k$ folding hits with L and R placed near the middle of the block. (We use here the fact that ℓ/k is much larger than the distance in G_k between L and R.) In this case the copies of G_{k-1} that are embedded within other parts of G_k will necessarily be placed in buckets in this block, leading to a k-pile ($k-1$ from G_{k-1} plus one folding hit).

Hence it remains to address the case that the strands of L and R are folded to the same direction, without loss of generality, to the right. Observe that in this case, no copy of G_{k-1} can be placed in buckets to the right of L. This is the place where we use the gadgets E_s and E_u that were placed in the clause subsegments of C_m is SAT and UNSAT.

Lemma 29 *In a shallow layout (in which without loss of generality R is not placed to the left of L) the strands of R are folded to the right and the strands of L are folded to the left.*

Proof: The SAT as well as the UNSAT segment is constructed of hole groups and simple brushes. The brush closest to R in the SAT segment (in clause C_m) was replaced by $E_s = (G_{k-1}B_{\frac{\alpha}{(2k)^4}})^{k(2k)^4}$, and the brush closest to L in the UNSAT segment was replaced by $E_u = (G_{k-1}B_{\frac{\alpha}{(2k)^3}})^{k(2k)^3}$. Both E_s and E_u are separated from their respective folding vertex by a hole group of relatively small size, H_∞^n.

Assume for the sake of contradiction that both the strands of R and the strands of L are folded to the right. In this case, the major part of E_s and of E_u must lie to the left of L (since they contain copies of G_{k-1}, and the $k-1$ piles in these copies cannot lie to the right of L). One of the two (E_s or E_u) extends at least as much as the other to the left. Say that this is E_u. Recall that Proposition 22 shows that if E_u spans E_s then a k-pile is created. The proof of this proposition easily applies also to our setting (in which E_u extends to the left more than E_s does), because the rightmost buckets reached by E_u and E_s are nearly the same (at most nh_∞ to the left of L, at most $\frac{\alpha}{(2k)^3}$ to the right of L, whereas the total backbone length of E_s or E_u is much large, $k\alpha$). This will lead to the creation of a k-pile. $\Box$

This completes the proof of item (i). We may no assume without loss of generality that in fact all strands of a given folding vertex are folded to the same direction (to the right for R and to the left for L). Hence the active region is assumed not to suffer any folding hits.

Having used E_u and E_s for the proof of item (i), we no longer need their special structure, and it will now be convenient for us to think of them simply as brushes $B_{k\alpha}$. This is not a true description of them, because in addition to brushes they contain copies of G_{k-1}. Replacing E_s and E_u by brushes simplifies the proofs that follow, because then we do not need to deal with E_u and E_s as separate special cases in many of the arguments that we make. So as to simplify the presentation we use the following convention.

Convention: For all other purposes except for proving item (i), E_s and E_u can be replaced by brushes $b_{k\alpha}$.

All the claims that follow can be proved also without making the convention. However, the proofs become much longer. The key insight of why the convention can be made is that even though E_s and E_u contain copies of G_{k-1}, each such copy has much longer brushes on both sides.

Given item (i), the proof of item (ii) is straightforward. All buckets (within reachable distance) except for those in the active region suffer folding hits. Every key contains copies of G_{k-1}, and they contain $k-1$ piles. Hence all keys (meaning, or more exactly, all the $k-1$ piles that they contain) lie in the active region, or else a k-pile is created.

We now turn to prove item (iii).

We say that a key *intersects* a hole if at least one vertex of a tooth of the key is placed in the same bucket of at least one vertex of a dent in the hole. Let us first observe the following lemma.

Lemma 30 *Every key from every variable segment must intersect a hole of the same type as the key.*

Proof: As we have seen, all keys must be placed in the active region. The portion of the caterpillar between L and R is UNSATMSAT and it must span the active region. UNSATMSAT is composed only of brushes and holes. (Recall our convention regarding E_s and E_u. This is one place where it is used. There will be other such places but they will not be mentioned explicitly.) So as not to create a k-pile, a key must intersect some hole (as otherwise, a copy of G_{k-1} is spanned by a brush, creating a k-pile). By the properties of our construction of keys and holes, the hole a key is placed in (or if formally it is placed in more than one hole, then at least one of these holes) has to be of the same type as the key, or otherwise a k-pile is created (see Sections 4.5 and 4.7). $\square$

Keys of the variable segments are in the active region and need to be placed in holes. Hence UNSATMSAT need to provide these holes. However, an important aspect of item (iii) is not just that the holes of UNSATMSAT are placed in the active region, but also that they are placed in order. We now establish the consequences of deviated from the natural order.

We shall use the following notation. An *i-coupling* is a key K_i placed in hole H_i. A hole is *destroyed* (at some point of describing the bucket layout) if no key can be placed in it without creating a k-pile. A *hole group* is a concatenation of holes of the same type (for example, H^2, or $H^n_{p(j)}$).

Lemma 31 *Let H' denote a hole of an arbitrary type T'. If in a bucket arrangement H' is spanned by a sequence of brushes and holes of types different than T', then the hole is destroyed.*

Proof: To begin with, only keys of type T' can fit in H'. So assume that one attempts to place such a key K' in H'. If H' is spanned by a sequence of brushes and holes of other types, then with respect to this sequence, K' does not interest a hole of type T', contradicting Lemma 30. $\square$

Corollary 32 *A hole-key coupling of a certain type cannot be spanned by a concatenation of simple brushes and hole groups of other types, unless a k-pile is created.*

We have not completely proved item (iii) yet, but we have shown that if it does not hold then necessarily some holes are destroyed. Destroying a single hole does not have major consequences, because two keys of the same type may be able to fit one hole of the same type, in the sense that no k-pile is created. Hence as long as a fair fraction of holes within a hole group remain, no k-pile needs to be created. However if all holes in a hole group are destroyed, then (as we shall see), certain keys will not have any holes to be placed in. Hence for the moment, all the reader needs to remember regarding item (iii) is that either all hole groups of are placed in order, or some hole group is destroyed.

The proof of item (iv) is straightforward. Keys of type K_0 can only be placed in holes of type H_0, and only the middle segment has such holes. In particular, this means that the middle segment needs to lie in the active region. The following lemma provides some additional content, as it shows that the middle segment indeed lies between the UNSAT and SAT segments.

Lemma 33 *In a shallow layout, there is no bucket that contains vertices both from the SAT segment and from the UNSAT segment.*

Proof: We have just shown above that there is a 0-coupling in the buckets corresponding to the middle segment M. If SAT and UNSAT share a bucket, then they jointly span the active region, and this 0-coupling is in the active region. This contradicts Corollary 32. $\Box$

We now prove item v. For every variable v, it will be proved by induction on its subsegments v^j. It will be more convenient for us to start the induction at $j = m$ and go down. This is because the functional key in v^m is of type K_∞, and the only places where holes of type H_∞ appear are the clause subsegment C_m (in SAT and in UNSAT). As we have already established (see Lemma 30) that K_∞ must be placed in an H_∞ hole, then indeed the structural key of v^m is placed in the structural hole group of C_m (either in C^s_m or in C^u_m). For concreteness, let us assume that it is placed in C^s_m (in the SAT segment). The inductive step is proved in the following lemma.

Lemma 34 *Consider an arbitrary shallow bucket arrangement for G_{k-1}. Let v be an arbitrary variable for which the structural key K_∞ of v^m is placed in the structural hole group H^n_∞ of C^s_m. Then for every other subsegment v^j with $j \geq k+5$, the structural key of v^j lies in the structural hole group of C_j of the SAT segment.*

Proof: We show this inductively, assuming that the structural key of v^{j+1} is placed in a structural hole of C_{j+1} and showing that the corresponding statement for j. The base case of $j = m$ is given.

The structural key of v^j is of type $K_{p(j)}$. Hence it must be placed in hole group of type $H_{p(j)}$. Such a hole indeed appears in C_j, and our intention is that this is indeed the hole group in which the structural key is placed. However, there are other values j' for which $p(j') = p(j)$, and hence it remains to show that the structural key is not placed in a hole group of such a $C_{j'}$. There are three cases to consider.

Turning backward. This is the case $j' > j$. Note that for $p(j') = p(j)$ it must be that $j' \geq j+k+5$. Also, by induction, there is an i-coupling for all i satisfying $j+1 \leq i \leq j'$. By Corollary 32, none of the corresponding holes can be spanned by brushes and other types of holes, implying that all clauses from C_{j+2} up to $C_{j'-1}$ are laid out between C_{j+1} and $C_{j'}$. This is more than k clauses, and hence $C_{j'}$ is too far to be reached from C_{j+1} (unless all the k-clauses are compressed to a number of buckets comparable to the backbone length of one clause, but then the bucketwidth is kb).

Fast forward. This is the case $j' < j$, which in fact implies $j' \leq j-k-5$. As before, j' is simply to far unless part of one of the clauses in between C_{j+1} and $C_{j'}$ spans either the hole used by v^{j+1} in C_{j+1}, or the hole in $C_{j'}$ intended for use by v^j. Whichever case holds, the corresponding hole is destroyed by Corollary 32, contradicting the assumption that a key is placed in it. (There is another case to consider in which the hole group of $C_{j'}$ is placed in the same buckets as the hole group of C_j. This does not destroy these holes, but then we may think of the key of v^j as being placed in C_j rather than $C_{j'}$, and continue with the inductive proof.)

Jump to UNSAT. This is the case that one tries to place the structural hole of v^j in the UNSAT segment rather than in the SAT segment. Here, the assumption that $j \geq k+5$ again ensures that the distance to be jump is large (more than k clauses) and an argument as above applies. $\Box$

Observe that Lemma 34 proves not only item (v), but also item (vi). In fact, it also implies most of the remaining aspects of item (iii), except for the layout of the clauses $C_1, \ldots, C_{k+5}$. However, these clauses are empty, and hence we shall not care about their layout.

We now prove item (vii). Without loss of generality, this will be done only for variables placed in the SAT segment.

Lemma 35 *In a shallow layout, if a variable v is laid out on the SAT segment and v appears in a clause C_j in the BM4SAT instance, then the functional key corresponding to variable subsegment v^j lies in the functional hole group of C_j.*

Proof: Observe that we can assume that $j > k+5$, because clauses up to $k+5$ are empty. We have seen in Lemma 34 that the structural key of v^{j-1} is placed in the functional hole group of C_{j-1}.

The distance in G_k between the functional key of v^j and the structural key of v^{j-1} is roughly α (plus low order terms). Every functional hole except for the functional hole group of C_j is separated in G_k from the functional hole group of C_{j-1} by at least one brush $B_{k\alpha}$. This corresponds to a distance too far to cross (unless the brush $B_{k\alpha}$ spans only α buckets, but then the bucketwidth is at least $k\alpha$). □

We can now finish off the proof.

Lemma 36 *If G_k corresponds to a* no *instance of BM4SAT then every bucket arrangement of G_k has a k-pile.*

Proof: Assume for the sake of contradiction that there is a shallow layout for G_k. Then as we have seen (proving items (i) up to (vii)) this layout must be canonical. In a canonical layout, there is a natural correspondence between the segment (SAT or UNSAT) in which a variable segment is placed and a true/false assignment to the variable. Consider now the placement of the functional keys in this layout. It there are at most two functional keys in every functional hole group, there must be exactly two functional keys in every functional hole group (the number of functional keys is equal to the number of functional holes). Hence this corresponds to a BM4SAT assignment that satisfies exactly two variables per clause, contradiction the assumption that this is a *no* instance.

It follows that some functional hole group (with two holes) contains at least three functional keys. As each key has to be placed in some hole, two keys are placed in the same hole. These are different types of functional keys, because with every variable associated with the clause we associate a different type of functional key. Lemma 24 now imply that k-pile is created. □

4.12 Summary

We can now complete the proof of Theorem 2.

Proof: Given a BM4SAT formula ξ, pick b to be much larger than kn (for example $b = kn^2$) and construct the graph G_k as described in Section 4.9. If the BM4SAT formula is satisfiable, then the graph has bucketwidth $b + O(kn)$ (by Section 4.10), and hence bandwidth at most $2b + O(kn)$ (by Lemma 4). If the BM4SAT formula is not satisfiable, then any bucket arrangement of G_k creates a k-pile, and hence has bucketwidth $\Omega(kb)$ (see Section 4.11). The same lower bound applies to the bandwidth (by Lemma 4). Hence the gap between *yes* and *no* instances is $\Omega(k)$.

The size of G_k can be computed from Proposition 26. For constant k it is $n^{O(k)}$. This establishes the NP-hardness result. To prove hardness of approximation results within larger ratios, observe that for every k the size of G_k is at most $n^{O(k)}k^{O(k^2)}$. Picking $k \simeq \frac{\log n}{\log\log n}$ gives caterpillars of size $N = k^{O(k^2)}$ and then $k = \Omega(\sqrt{\log N/\log\log N})$. □

We would have liked to get a hardness of approximation ratio of $\Omega(\log n/\log\log n)$ for caterpillars, matching the algorithmic result of [11] up to constant factors. We do not get such a result because the size of the caterpillars G_k grows too quickly as a function of k. The main reason for this fast growth is our construction of structural keys (Section 4.5). Their sizes are proportional to $k^{O(k)}$. If instead we could use structural keys of size proportional to $k^{O(1)}$ (and still make the rest of the reduction work), then the size of G^k would be $N = n^{O(k)}$, and picking $k = n^\delta$ would allow us to obtain an $\Omega(\log N/\log\log N)$ hardness result for caterpillars (unless NP has subexponential time algorithms).

5 Hardness on asteroidal triple free graphs

In this section we prove Proposition 3. As noted in the introduction, the general scheme for such proofs (via a reduction for BBIS) was communicated to us by Shimon Kogan.

Balanced Bipartite Independent Set (BBIS)

Input A bipartite graph $G = (X \cup Y, E)$ with $|X| = |Y| = n/2$.

Problem Find the maximum value of k such that G contains an independent set $I = (I_X \subseteq X) \cup (I_Y \subseteq Y)$ with $|I_X| = |I_Y| = k$.

Theorem 37 *[18] For some $\epsilon > 0$, no polynomial time algorithm can distinguish between bipartite graphs $G(X \cup Y, E)$ for which BBIS has a value $k \geq 4\epsilon n$ (that we call* yes *instances) and bipartite graphs $G(X, Y)$ for which BBIS has a value $k \leq \epsilon n$ (that we call* no *instances), unless NP has subexponential time algorithms.*

Based on Theorem 37, we now prove Proposition 3.

Proof: To reduce a BBIS instance $G(X, Y)$ into an instance G' of bandwidth, we change each of the two sets X and Y into cliques. Observe that G' is at least $n/2$-dense (assuming that G did not have isolated vertices, an assumption that holds in the proof of Theorem 37) and AT-free (because every set of three vertices contains two vertices from the same clique, and hence cannot form an asteroidal triple).

On a *yes* instance $G(X \cup Y, E)$ of BBIS, let I_X and I_Y be the optimal solution. For G', take a linear arrangement in which the vertices of I_X are the first vertices, the vertices of I_Y are the last vertices, and other vertices are placed in arbitrary order in between. As there are no edges between I_X and I_Y, the bandwidth is at most $n - k \leq n - 4\epsilon n$.

On a *no* instance $G(X \cup Y, E)$ of BBIS, let k' be largest such that G' has bandwidth at least $n - 2k'$. In any linear arrangement achieving this bandwidth there are no edges between the first k' vertices (call them I_1) and the last k' vertices (call them I_2). Also, for I_1 and I_2 one of them must lie entirely in X and the other entirely in Y. (Otherwise there will be edges between I_1 and I_2, because X and Y are cliques in G'). Hence the BBIS instance has value at least k'. Being a *no* BBIS instance, we conclude that $k' \leq \epsilon n$.

Theorem 37 now implies that unless NP has subexponential algorithms, it is hard to distinguish between bandwidth at most $(1 - 4\epsilon)n$ and bandwidth at least $(1 - 2\epsilon)n$. The ratio between these two values is at least $1 + \epsilon$ for some fixed $\epsilon > 0$, excluding the possibility of a PTAS. □

Acknowledgements

We thank Shimon Kogan for the simple proof of Proposition 3.

References

[1] S. F. Assmann, G. W. Peck, M. M. Syslo, and J. Zak. The bandwidth of caterpillars with hairs of length 1 and 2. *SIAM Journal on Algebraic and Discrete Methods*, 2(4):387–393, 1981.

[2] G. Blache, M. Karpinski, and J. Wirtgen. On approximation intractability of the bandwidth problem. *Electronic Colloquium on Computational Complexity (ECCC)*, 5(14), 1998.

[3] P. Chinn, J. Chvatalova, A. Dewdney, and N. Gibbs. The bandwidth problem for graphs and matrices - survey. *Journal of Graph Theory*, 6(3):223–254, 1982.

[4] F. R. K. Chung and P. D. Seymour. Graphs with small bandwidth and cutwidth. *Discrete Mathematics*, 75(1-3):113–119, 1989.

[5] D. G. Corneil, S. Olariu, and L. Stewart. Asteroidal triple-free graphs. *SIAM J. Discrete Math.*, 10(3):399–430, 1997.

[6] C. Dubey. On bandwidth approximation of graphs. *Master's Thesis*, Weizmann Institute of Science, 2008.

[7] J. Dunagan and S. Vempala. On euclidean embeddings and bandwidth minimization. In proceedings of APPROX/RANDOM 2001, volume 2129 of Lecture Notes in Computer Science, Springer, 2001, pages 229–240.

[8] U. Feige. Approximating the bandwidth via volume respecting embeddings. *J. Comput. Syst. Sci.*, 60(3):510–539, 2000.

[9] U. Feige. Coping with the NP-hardness of the graph bandwidth problem. In proceedings of SWAT 2000, 7th Scandinavian Workshop on Algorithm Theory, 2000, volume 1851 of *Lecture Notes in Computer Science*, Springer, 2000, pages 10–19.

[10] U. Feige and S. Kogan. Hardness of Approximation of the Balanced Complete Bipartite Subgraph Problem. *Technical Report*, MCS04-04, Weizmann Institute, 2004.

[11] U. Feige and K. Talwar. Approximating the bandwidth of caterpillars. In *APPROX-RANDOM*, pages 62–73, 2005.

[12] Y. Filmus. Bandwidth approximation of a restricted class of trees. *Master's Thesis*, Weizmann Institute of Science, 2002.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Addison-Wesley, New York, 1979.

[14] A. Gupta. Improved bandwidth approximation for trees and chordal graphs. *J. Algorithms*, 40(1):24–36, 2001.

[15] J. Haralambides, F. Makedon, and B. Monien. Bandwidth minimization: An approximation algorithm for caterpillars. *Mathematical Systems Theory*, 85176-CS:169–177, 1991.

[16] M. Karpinski and J. Wirtgen. NP-hardness of the bandwidth problem on dense graphs. *University of Bonn, Technical Report*, 85176-CS, 1997.

[17] M. Karpinski, J. Wirtgen, and A. Zelikovsky. An approximation algorithm for the bandwidth problem on dense graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(17), 1997.

[18] S. Khot. Ruling out ptas for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM J. Comput.*, 36(4):1025–1071, 2006.

[19] D. J. Kleitman and R. Vohra. Computing the bandwidth of interval graphs. *SIAM J. Discrete Math.*, 3(3):373–375, 1990.

[20] T. Kloks, D. Kratsch, Y. L. Borgne, and H. Müller. Bandwidth of split and circular permutation graphs. In proceedings of Graph-Theoretic Concepts in Computer Science, 26th International Workshop, WG 2000, volume 1928 of *Lecture Notes in Computer Science*, Springer, 2000, pages 243–254.

[21] T. Kloks, D. Kratsch, and H. Müller. Approximating the bandwidth for asteroidal triple-free graphs. *J. Algorithms*, 32(1):41–57, 1999.

[22] D. Kratsch and L. Stewart. Approximating Bandwidth by Mixing Layouts of Interval Graphs. *SIAM J. Discrete Math.*, 15(4):435–449, 2002.

[23] R. Krauthgamer, J. R. Lee, M. Mendel, and A. Naor. Measured descent: A new embedding method for finite metrics. In proceedings of 45th Symposium on Foundations of Computer Science (FOCS 2004), IEEE Computer Society, pages 434–443.

[24] J. R. Lee. Volume distortion for subsets of euclidean spaces: extended abstract. In proceedings of the 22nd ACM Symposium on Computational Geometry, 2006, pages 207–216.

[25] L. Lovász. Colorings and coverings of hypergraphs. *Proc. 4th Southeastern Conference on Combinatorics. Graph Theory, and Computing, Utilitas Mathematica Publishing, Winnipeg*, pages 3–12.

[26] R. Mahesh, C. P. Rangan, and A. Srinivasan. On finding the minimum bandwidth of interval graphs. *Inf. Comput.*, 95(2):218–224, 1991.

[27] B. Monien. The bandwidth minimization problem for caterpillars with hair length 3 is NP-complete. *SIAM journal of Algebraic Discrete Methods*, 7(4):505–512, 1986.

[28] C. Papadimitriou. The NP-completeness of bandwidth minimization problem. *Computing*, 16:263–270, 1976.

[29] G. W. Peck and A. Shastri. Bandwidth of theta graphs with short paths. *Discrete Mathematics*, 103(2):177–187, 1992.

[30] J. Saxe. Dynamic programming algorithms for recognizing small bandwidth graphs in polynomial time. *SIAM journal of Algebraic Discrete Mathods*, 1:363–369, 1980.

[31] A. P. Sprague. An O(n log n) algorithm for bandwidth of interval graphs. *SIAM J. Discrete Math.*, 7(2):213–220, 1994.

[32] W. Unger. The complexity of the approximation of the bandwidth problem. In *FOCS*, pages 82–91, 1998.