Online Promise Problems with Online Width Metrics.

Rodney G. Downey[1] and Catherine McCartin[2]

[1] Victoria University, Wellington, New Zealand
Rod.Downey@mcs.vuw.ac.nz
[2] Massey University, Palmerston North, New Zealand
C.M.McCartin@massey.ac.nz

Abstract. In this article we introduce a new program that applies ideas from parameterized complexity, and topological graph theory, to online problems. We focus on parameterized *promise* problems, where we are *promised* that the problem input obeys certain properties, or is presented in a certain fashion. We present some algorithmic ramifications of such problems in the online setting.
We explore the effects of using graph width metrics as restrictions on the input to online problems. It seems natural to suppose that, for graphs having some form of bounded width, good online algorithms may exist for a number of natural problems. In the work presented we concentrate on online graph coloring problems, where we restrict the allowed input to instances having some form of bounded treewidth or pathwidth.
We also consider the effects of restricting the presentation of the input to some form of bounded width decomposition or layout. A consequence of this part of the work is the clarification of a new parameter for graphs, *persistence*, which arises naturally in the online setting, and is of interest in its own right. We present some basic results regarding the general recognition of graphs having *bounded persistence* path decompositions.

1 Introduction

The last 20 years has seen a revolution in the development of graph algorithms. This revolution has been driven by the systematic use of ideas from topological graph theory, with the use of graph width metrics emerging as a fundamental paradigm in such investigations. The role of graph width metrics, such as treewidth, pathwidth, and cliquewidth, is now seen as central in both algorithm design and the delineation of what is algorithmically possible. In turn, these advances cause us to focus upon the "shape", or the "inductive nature", of much real life data. Indeed, for many real life situations, worst case, or even average case, analysis no longer seems appropriate, since the data is known to have a highly regular form, especially when considered from the parameterized point of view.

In this article we introduce a new program that applies ideas from parameterized complexity, and topological graph theory, to online problems. We focus on parameterized *promise* problems, where we are *promised* that the problem input obeys certain properties, or is presented in a certain fashion. We present some algorithmic ramifications of such problems in the online setting.

It is now commonplace, in the offline setting, to find that by restricting some width parameter for the input graphs, a particular graph problem can be solved efficiently. A number of different graph width metrics naturally arise in this context which restrict the inherent complexity of a graph in various senses. The central idea is that a useful width metric should admit efficient algorithms for many (generally) intractable problems on the class of graphs for which the width is small. One of the most successful measures in this context is the notion of *treewidth* which arose from the seminal work of Robertson and Seymour on graph minors and immersions [29]. Treewidth measures, in a precisely defined way, how "tree-like" a graph is. The idea here is that we can lift many results from trees to graphs that are "tree-like". Related to treewidth is the notion of *pathwidth* which measures, in the same way, how "path-like" a graph is.

It turns out, however, that the classic algorithms generated through the use of width metrics usually rely upon dynamic programming, and so are highly unsuited to the focus of our investigation, the online situation.

The usual theoretic model for online problems has the input data presented to the algorithm in small units, one unit per timestep. The algorithm produces a string of outputs: after seeing t units of input, it needs to produce the t-th unit of output. Thus, the algorithm makes a decision based only on partial information about the whole input string, namely the part that has been read so far. How good the decision of the algorithm is at any given step t may depend on the future inputs, inputs that the algorithm has not yet seen.

Real-life online situations often give rise to input patterns that seem to be "long" and "narrow", that is, pathlike. For instance, consider the online scheduling of some large collection of tasks onto a small number of processors. One might reasonably expect a pattern of precedence constraints that gives rise to only small clusters of interdependent tasks, with each cluster presented more or less contiguously. Alternatively, one might expect a pattern of precedence constraints giving rise to just a few long chains of tasks, with only a small number of dependencies between chains, where each chain is presented more or less in order. One could argue that the most compelling reason for attempting to solve a problem online is that the end of the input is "too long coming" according to some criteria that we have. Given such a situation, we attempt to do the best we can with the partial information available at each timestep.

In the work presented here, we consider the effects of using width metrics as restrictions on the input to online problems. As mentioned above, online situations often give rise to input patterns that seem to naturally conform to restricted width metrics, in particular to bounded pathwidth. We might expect to obtain online algorithms having good performance, for various online problems, if we promise that the allowed input instances have some form of bounded treewidth or pathwidth.

We also consider the effects of restricting the *presentation* of the input to some form of bounded width decomposition or layout. The method of presentation of the input structure to an algorithm has a marked effect on performance. Indeed, this observation underpins the study of online algorithms in the first place. A consequence of this part of the work is the clarification of a new, related, parameter for graphs, *persistence*, which arises naturally in the online setting, and is of interest in its own right. We believe that *persistence* truly captures the intuition behind the notion of pathwidth.

We present some basic results regarding the general recognition of graphs having bounded persistence pathwidth, using the framework of *parameterized complexity theory*, introduced by Downey and Fellows [9]. Our results show that deciding whether or not a given graph has bounded persistence pathwidth is a parametrically hard problem, whereas the corresponding problem for traditional treewidth or pathwidth is not. Nevertheless, the input for many problems is *naturally* presented in a reasonable way and hence, in spite of the lack of efficient recognition algorithms (as exhibited by the hardness results that we present), we believe that persistence is an important parameter to be exploited.

To lay the foundations of the program described, we concentrate on *online graph coloring*. There has been a considerable amount of work done on online graph coloring, much of it related to earlier work on bin packing. However, approaches similar to the one that we take are notably absent from the literature.

In the following section we present some necessary preliminaries. After that, we introduce our definition of an online presentation for a graph, which naturally incorporates the notion of persistence. In Sections 4 and 5 we discuss online graph coloring and present our results for parameterized promise problems in the context of online graph coloring. In Section 6 we return to our notion of persistence and present results regarding the complexity of recognizing graphs having bounded persistence path decompositions.

2 Preliminaries

2.1 Parameterized Complexity

Throughout this article, we use the framework of *parameterized complexity theory*, introduced by Downey and Fellows [9]. We remind the reader that a parameterized language L is a subset of $\Sigma^* \times \Sigma^*$. If L is a parameterized language and $\langle \sigma, k \rangle \in L$ then we refer to σ as the *main part* and k as the *parameter*.

The basic notion of tractability is *fixed parameter tractability* (FPT). Intuitively, we say that a parameterized problem is fixed-parameter tractable (FPT) if we can somehow confine the any "bad" complexity behaviour to some limited aspect of the problem, the parameter.

Formally, we say that a parameterized language, L, is fixed-parameter tractable if there is a computable function f, an algorithm A, and a constant c such that for all k, $\langle x, k \rangle \in L$ iff $A(x, k) = 1$, and $A(x, k)$ runs in time $f(k)|x|^c$ (c is independent of k). For instance, k-Vertex Cover is solvable in time $\mathcal{O}(|x|)$. On the other hand, for k-Turing Machine Acceptance, the problem of deciding if a nondeterministic Turing machine with arbitrarily large fanout has a k-step accepting path, the only known algorithm is to try all possibilities, and this takes time $\Omega(|x|^k)$. This situation, akin to NP-completeness, is described by hardness classes, and reductions. A parameterized reduction, L to L', is a transformation which takes $\langle x, k\rangle$ to $\langle x', k'\rangle$, running in time $g(k)|x|^c$, with $k \mapsto k'$ a function purely of k.

Downey and Fellows [9] observed that these reductions gave rise to a hierarchy called the W-hierarchy.

$$FPT \subset W[1] \subseteq W[2] \subseteq \ldots \subseteq W[t] \subseteq \ldots.$$

The core problem for $W[1]$ is k-Turing Machine Acceptance, which is equivalent to the problem Weighted 3Sat. The input for Weighted 3Sat is a 3CNF formula, φ and the problem is to determine whether or not φ has a satisfying assignment of Hamming weight k. $W[2]$ has the same core problem except that φ is in CNF form, with no bound on the clause size. In general, $W[t]$ has as its core problem the weighted satisfiability problem for φ of the form "products of sums of products of ..." of depth t. It is conjectured that the W-hierarchy is proper, and from $W[1]$ onwards, all parametrically intractable.

2.2 Treewidth and Pathwidth

Many generally intractable problems become tractable for the class of graphs that have bounded treewidth or bounded pathwidth. Furthermore, treewidth and pathwidth subsume many graph properties that have been previously mooted, in the sense that tractability for bounded treewidth or bounded pathwidth implies tractability for many other well-studied classes of graphs. For example, planar graphs with radius k have treewidth at most $3k$, series parallel multigraphs have treewidth 2, chordal graphs (graphs having no induced cycles of length 4 or more) with maximum clique size k have treewidth at most $k-1$, graphs with bandwidth at most k have pathwidth at most k.

A graph G has treewidth at most k if we can associate a tree T with G in which each node represents a subgraph of G having at most $k+1$ vertices, such that all vertices and edges of G are represented in at least one of the nodes of T, and for each vertex v in G, the nodes of T where v is represented form a subtree of T. Such a tree is called a *tree decomposition* of G, of *width* k. We give a formal definition here:

Definition 1. *[Tree decomposition and Treewidth]*
Let $G = (V, E)$ be a graph. A tree decomposition of G is a pair $(T, \mathcal{X})$ where $T = (I, F)$ is a tree, and $\mathcal{X} = \{X_i \mid i \in I\}$ is a family of subsets of V, one for each node of T, such that

1. $\bigcup_{i \in I} X_i = V$,
2. *for every edge $\{v, w\} \in E$, there is an $i \in I$ with $v \in X_i$ and $w \in X_i$,*
3. *for all $i, j, k \in I$, if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

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

Definition 2. *[Path decomposition and Pathwidth]*
A path decomposition of a graph G *is a tree decomposition* $(P, \mathcal{X})$ *of* G *where* P *is simply a path (i.e. the nodes of* P *have degree at most two). The pathwidth of* G *is the minimum width over all possible path decompositions of* G.

Any path decomposition of G is also a tree decomposition of G, so the pathwidth of G is at least equal to the treewidth of G. For many graphs, the pathwidth will be somewhat larger than the treewidth. For example, let B_k denote the complete binary tree of height k and order $2^k - 1$, then $tw(B_k) = 1$, but $pw(B_k) = k$.

Graphs of treewidth and pathwidth at most k are also called *partial k-trees* and *partial k-paths*, respectively, as they are exactly the subgraphs of k-trees and k-paths. There are a number of other important variations equivalent to the notions of treewidth and pathwidth (see, e.g., Bodlaender [3]). For algorithmic purposes, the characterizations provided by the definitions given above tend to be the most useful.

2.3 Finding Tree and Path Decompositions

We mention above that many intractable problems become tractable for the class of graphs that have bounded treewidth or bounded pathwidth. A more accurate statement would be to say that many intractable problems become *theoretically* tractable for this class of graphs, in the general case.

The typical method employed to produce efficient algorithms for problems restricted to graphs of bounded treewidth (pathwidth) proceeds in two stages (see [4]).

1. Find a bounded-width tree (path) decomposition of the input graph that exhibits the underlying tree (path) structure.
2. Perform dynamic programming on this decomposition to solve the problem.

In order for this approach to produce practically efficient algorithms, as opposed to proving that problems are theoretically tractable, it is important to be able to produce the necessary decomposition reasonably efficiently.

Many people have worked on the problem of finding progressively better algorithms for recognition of bounded treewidth (pathwidth) graphs, and construction of associated decompositions.

As a first step, Arnborg, Corneil, and Proskurowski [1] showed that if a bound on the treewidth (pathwidth) of the graph is known, then a decomposition that acheives this bound can be found in time $O(n^{k+2})$, where n is the size of the input graph and k is the bound on the treewidth (pathwidth). They also showed that determining the treewidth or pathwidth of a graph in the first place is NP-hard.

Robertson and Seymour [29] gave the first FPT algorithm, $O(n^2)$, for k-Treewidth. Their algorithm, based upon the minor well-quasi-ordering theorem (see [28]), is highly non-constructive, non-elementary, and has huge constants.

The early work of [1, 29] has been improved upon in the work of Lagergren [14], Reed [27], Fellows and Langston [10], Matousek and Thomas [24], Bodlaender [2], and Bodlaender and Kloks [7], among others.

Bodlaender [2] gave the first linear-time FPT algorithms for the constructive versions of both k-Treewidth and k-Pathwidth, although the $f(k)$'s involved mean that treewidth and pathwidth still remain parameters of theoretical interest only, at least in the general case.

Bodlaender's algorithms recursively invoke a linear-time FPT algorithm due to Bodlaender and Kloks [7] which "squeezes" a given width p tree decomposition of a graph G down to a width k tree (path) decomposition of G, if G has treewidth (pathwidth) at most k. A small improvement to the Bodlaender/Kloks algorithm would substantially improve the performance of Bodlaender's algorithms.

Perkovic and Reed [26] have recently improved upon Bodlaender's work, giving a streamlined algorithm for k-Treewidth that recursively invokes the Bodlaender/Kloks algorithm no more than $O(k^2)$ times, while Bodlaender's algorithms may require $O(k^8)$ recursive iterations.

For some graph classes, the optimal treewidth and pathwidth, or good approximations of these, can be found using practically efficient polynomial time algorithms. Examples are chordal bipartite graphs, interval graphs, permutation graphs, circle graphs, and co-graphs.

2.4 Competitiveness

We measure the performance of an online algorithm, or gauge the difficulty of an online problem, using the concept of *competitiveness*, originally defined by Sleator and Tarjan [30] (see also Manasse, McGeoch, and Sleator [23]).

Suppose that P is an online problem, and A is an online algorithm for P. Let $c \geq 1$ be a constant. We say that A is *c-competitive* if, for any instance I of problem P,

$$cost_A(I) \leq c \cdot cost_{opt}(I) + b$$

where *opt* is an optimal offline algorithm that sees all information about the input in advance, and b is a constant independent of I. In other words, A pays at most $c + O(1)$ times the optimal cost, for any given input string.

We say that a given online problem P is c-competitive if there exists a c-competitive algorithm for P, and we say that it is *no better than c-competitive* if there exists no c'-competitive algorithm for P for any $c' \leq c$. An online algorithm A is said to be *optimal* for P if A is c-competitive and P is no better than c-competitive.

Competitive analysis measures algorithm performance relative to what is achievable by an omniscient algorithm, rather than in absolute terms. If an algorithm A is c-competitive, then we say that A has a *performance ratio* of c.

3 Online Presentations

The usual definition of an *online presentation* of a graph G is a structure $G^< = (V, E, <)$ where $<$ is a linear ordering of V. G is presented one vertex per timestep, v_1 at time 1, v_2 at time 2, …and so on. At each step, the edges incident with the newly introduced vertex and those vertices already "visible" are also presented. We use the terms *online presentation* and *online graph* interchangeably.

Let $V_i = \{v_j \mid j \leq i\}$ and $G^<_i = G^<[V_i]$, the online subgraph of $G^<$ induced by V_i. An algorithm that solves some online problem on G will make a decision regarding v_i (and/or edges incident with v_i) using only information about $G^<_i$.

An extension of this model adds the notion of *lookahead*, where the algorithm is shown a limited portion of the remaining graph at the time that it must make each decision. In this case the algorithm will make a decision regarding v_i (and/or edges incident with v_i) using only information about $G^<_{i+l}$, for some $l \geq 1$. Another way to view lookahead is as limited revision, being able to see the next l units of input at the time that we make each decision is essentially the same as being able to revise the last l decisions each time we see the next unit of input.

In this article we introduce a different method of presenting a graphlike structure online. First, fix some arbitrary constant (parameter) k. At each timestep we present one new *active* vertex that may be incident with at most k active vertices previously presented. Once a vertex has been presented we may render some of the current set of active vertices *inactive* in preparation for the introduction of the next new vertex. At no point do we allow more than $k + 1$ active vertices, and we do not allow a new vertex to be incident with any inactive vertex.

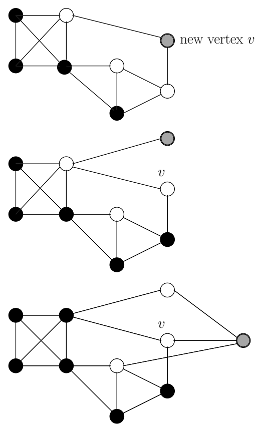

Fig. 1. Online presentation of a pathwidth 3 graph using 4 active vertices. Vertex v remains active for 3 timesteps.

These requirements mean that any graph presented in this fashion must have bounded pathwidth (pathwidth k). We are, in effect, presenting the graph as a path decomposition, one node per timestep. We denote such an online presentation of a graph G as $G^{<\text{ path } k}$.

We can add the further requirement that any vertex may *remain active* for at most l timesteps, for some arbitrary constant (parameter) l. We say that a path decomposition of width k, in which every vertex of the underlying graph belongs to at most l nodes of the path, has width k and *persistence* l, and say that a graph that admits such a decomposition has *bounded persistence pathwidth.*

Bounding the persistence of the presentation further restricts the class of graphs that can be considered, but in a quite natural fashion. We are assuming, in a sense, at most k parallel "channels", where dependencies are limited to short timeframes, e.g. if vertex (event) b appears much later than vertex a, then b can be affected only indirectly by a.

We explore persistence further in Section 6, but we remark here that we believe that this natural notion truly captures the intuition behind the notion of pathwidth.

An online graph that can be presented in the form of a path decomposition with both low width and low persistence is properly pathlike, whereas graphs that have *high* persistence are, in some sense, "unnatural" or pathological. Consider the graph G presented in Figure 2. G is not really path-like, but still has a path decomposition of width only two. The reason for this is reflected in the presence of vertex a in *every* node of the path decomposition.

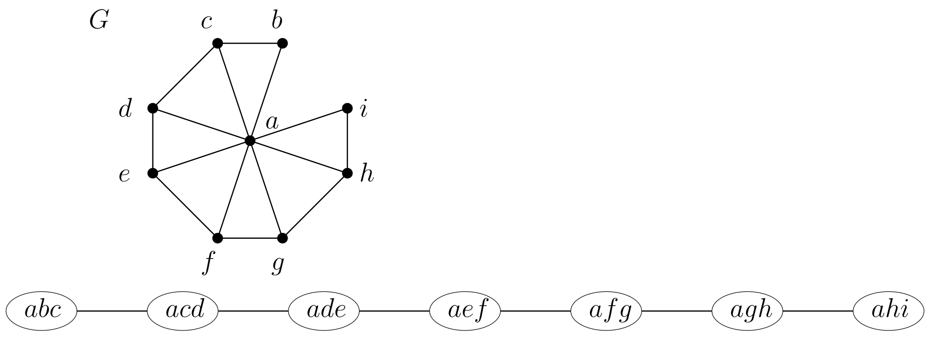

Fig. 2. A graph G having low pathwidth but high persistence.

Persistence appears to be a natural and interesting parameter in both the online setting *and* the offline setting. For many problems where the input is generated as an ordered sequence of small units, it seems natural to expect that the sphere of influence of each unit of input should be localised.

4 Online Coloring

An online algorithm A for coloring an online graph $G^<$ will determine the color of the ith vertex of $G^<$ using only information about $G_i^<$. A colors the vertices of $G^<$ one at a time in the order $v_1 < v_2, \cdots$, and at the time a color is irrevocably assigned to v_i, the algorithm can only see $G_i^<$.

A simple, but important, example of an online algorithm is *First-Fit*, which colors the vertices of $G^<$ with an initial sequence of the colors $\{1, 2, \ldots\}$ by assigning to v_i the least color that has not already been assigned to any vertex in $G_i^<$ that is adjacent to v_i.

Szegedy [31] has shown that, for any online coloring algorithm A and integer k, there is an online graph $G^<$ on at most $k(2^k - 1)$ vertices with chromatic number k on which A will use $2^k - 1$ colors. This yields a lower bound of $\Omega(\frac{n}{(log\, n)^2})$ for the performance ratio of any online coloring algorithm on general graphs. Note that the worst possible performance ratio on general graphs is n. Lovasz, Saks, and Trotter [22] have given an algorithm that achieves a performance ratio $O(\frac{n}{(log\, n)^*})$ on all graphs.

Online coloring of some restricted classes of graphs has been considered. In the bipartite case it can be shown that, for any online coloring algorithm A and integer k, there is an online tree $T^<$ with 2^{t-1} vertices on which A will use at least t colors. Thus, we get a lower bound of $\Omega(log\, n)$ for any online algorithm on bipartite graphs. Lovasz, Saks, and Trotter [22] give an algorithm that colors any bipartite online graph using at most $1 + 2\, log\ n$ colors.

Kierstead and Trotter [19] have given an online coloring algorithm that achieves a performance ratio of 3 on interval graphs, which is also best possible. Kierstead [17] has shown that First-Fit has a constant performance ratio on the class of interval graphs. Gyarfas and Lehel [13] have shown that First-Fit achieves a constant performance ratio on split graphs, complements of bipartite graphs, and complements of chordal graphs.

One approach that is similar in flavour to ours is presented by Irani [15]. Irani introduces the notion of *d-inductive* graphs. A graph G is *d-inductive* if the vertices of G can be ordered in such a way that each vertex is adjacent to at most d higher numbered vertices. Such an ordering on the vertices is called an *inductive order*. As for a path or tree decomposition, an inductive order is not necessarily unique for a graph. An inductive order of a graph G defines an *inductive orientation* of G, obtained by orienting the edges from the higher numbered vertices to the lower numbered vertices. Notice that, in an inductive orientation, the indegree of each vertex is bounded by d. Hence, any d-inductive graph is $d + 1$ colorable.

In [15] it is shown that, if G is a d-inductive graph on n vertices, then First-Fit uses at most $O(d \cdot log\, n)$ colors to color any online presentation $G^<$ of G. Moreover, for any online coloring algorithm A, there exists a d-inductive online graph $G^<$ such that A uses at least $\Omega(d \cdot log\, n)$ colors to color $G^<$.

A connection between graphs of bounded pathwidth or bounded treewidth, and inductive graphs, is given by the following lemma (see [21], or [25]).

Lemma 1. *Any graph G of pathwidth k, or treewidth k, is k-inductive.*

5 Online Coloring of Bounded Width Graphs

We consider two ways in which to formulate parameterized promise problems for online coloring of graphs.

We can simply promise to fix a bound k on the treewidth or pathwidth of any input graph G, and then proceed to present G as a structure $G^{<} = (V, E, <)$ where $<$ is an *arbitrary* linear ordering of V.

Alternatively, we can define a parameterized "presentation" promise, where we fix a bound k on the pathwidth of any input graph G, and then proceed to present G as an implicit path decomposition, in the manner described in Section 3 above. We deal with this formulation first.

5.1 The Presentation Promise

If we undertake to present a graph G in the form of an implicit path decomposition, then we are effectively enforcing the presentation to be, if not best-possible, then at least "very good" for the simple strategy of First-Fit acting on G.

Lemma 2. *If G is a graph of pathwidth k, presented in the form of an implicit path decomposition, then First-Fit will use at most $k+1$ colors to color $G^{<\text{ path } k}$.*

This is easy to see, since, at each step i, $0 \leq i \leq n$, the newly presented vertex v_i will be adjacent to at most k already-colored vertices. This result is best possible in the sense that the chromatic number (and, therefore, the online chromatic number) of the class of graphs of pathwidth k is $k + 1$. However, note that $G^{<\text{path } k}$ may not contain all of the information required to color G *optimally* online, as the following lemma shows.

Lemma 3. *For each $k \geq 0$, there is a tree T of pathwidth k presented as $T^{<\text{ path } k}$ on which First-Fit can be forced to use $k + 1$ colors.*

Proof. Suppose T_0 is a connected tree with pathwidth 0, then T must consist of a single vertex (any graph of pathwidth 0 must consist only of isolated vertices) so First-Fit will color $T_0^{<\text{ path } 0}$ with one color.

Suppose T_1 is a connected tree with pathwidth 1 that has at least two vertices. Each vertex of $T_1^{<\text{ path } 1}$ can be adjacent to at most one active vertex at the time of presentation. Since T_1 is connected, there must be vertices that are adjacent to an active vertex at the time of presentation in any $T_1^{<\text{ path } 1}$. Thus, First-Fit will need to use two colors to color any $T_1^{<\text{ path } 1}$.

Now, suppose that for any $0 \leq t < k$, there is a tree T_t of pathwidth t, and a presentation $T_t^{<\text{ path } t}$, on which First-Fit can be forced to use $t + 1$ colors. We build a connected tree T_k with pathwidth k, and a presentation $T_k^{<\text{ path } k}$, on which First-Fit will be forced to use $k + 1$ colors.

We order the trees T_t, $0 \leq t < k$, and their presentations, in descending order $[T_{k-1}, T_{k-2}, \ldots, T_0]$, and concatenate the presentations together in this order to obtain a new presentation $T^{<}_{con}$. Note that the subsequence $T_t^{<\,\text{path}\,t}$ of $T^{<}_{con}$ will have at most $(t+1) \leq k$ active vertices at any stage.

To obtain $T^{<\,\text{path}\,k}$, we alter $T^{<}_{con}$ as follows. For each t, $0 \leq t < k$, we choose the vertex v_t from $T_t^{<\,\text{path}\,t}$ that is colored with color $t+1$ by First-Fit and allow it to remain active throughout the rest of $T^{<}_{con}$. Every other vertex from $T_t^{<\,\text{path}\,t}$ is rendered inactive at the conclusion of $T_t^{<\,\text{path}\,t}$ in the concatenated presentation. Thus, at any stage of $T^{<}_{con}$, there will be at most $k+1$ active vertices, and at the conclusion of $T^{<}_{con}$ there will be k active vertices, one from each of the $T_t^{<\,\text{path}\,t}$, $0 \leq t < k$. These k active vertices will be colored with colors $1, 2, \ldots, k$ respectively. We now present one new vertex, adjacent to each of the k active vertices, which must be colored with color $k+1$. □

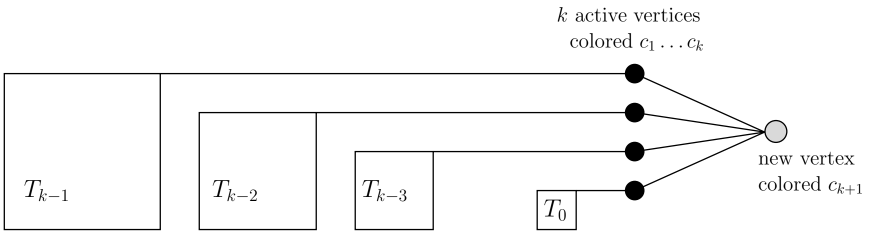

Fig. 3. Schema of $T^{<\,\text{path}\,k}$ on which First-Fit can be forced to use $k+1$ colors.

5.2 Bounded Width Promises

In this section we consider the situation where the promise is simply the topological fact that the input graph has bounded width of some type, and we make no restrictions on how the graph is presented.

As mentioned in Section 4, Kierstead and Trotter [19] have considered online coloring for *interval graphs.* A graph $G = (V, E)$ is an *interval graph* if there is a function ψ which maps each vertex of V to an interval of the real line, such that for each $u, v \in V$ with $u \neq v$, $\psi(u) \cap \psi(v) \neq \emptyset \Leftrightarrow (u, v) \in E$. The function ψ is called an *interval realization* for G. The relation between interval graphs and graphs of bounded pathwidth is captured by the following lemma (see [20], or [25]).

Lemma 4. *A graph G has pathwidth at most k if and only if G is a subgraph of an interval graph G', where G' has maximum clique size at most $k+1$.*

Kierstead and Trotter [19] have given an online algorithm that colors *any* online interval graph $G^{<}$, having maximum clique size at most $k+1$, using

$3k+1$ colors. Thus, any graph of pathwidth k can be colored online using at most $3k+1$ colors.

If we insist upon sticking with the simple strategy of First-Fit we can at least achieve a constant performance ratio on graphs having pathwidth at most k. Kierstead [17] has shown that for every online interval graph $G^<$, with maximum clique size at most $k+1$, First-Fit will use at most $40(k+1)$ colors. In [18] Kierstead and Qin have improved the constant here to 25.72.

Chrobak and Slusarek [8] have shown that there exists an online interval graph $G^<$, and a constant c, where c is the maximum clique size of $G^<$, such that First-Fit will require at least $4.4{\cdot}c$ colors to color $G^<$. Such an interval graph $G^<$ will have pathwidth $c-1$ and chromatic number c. Thus, the performance ratio of First-Fit on graphs of pathwidth k must be at least 4.4. It is open as to what the correct lower bound is here.

In the case where G is a *tree* of pathwidth k we get definitive results for First-Fit.

Lemma 5. *First-Fit will use at most $3k+1$ colors to color any $T^<$ where T is a tree of pathwidth k.*

Proof. Let $k=0$, then T consists only of an isolated vertex, and First-Fit requires only one color to color $T^<$.

Let k be ≥ 1. Suppose that the bound holds for $k-1$: for any tree T of pathwidth at most $k-1$ First-Fit colors any $T^<$ with at most $3k-2$ colors.

We rely on the fact that any tree T of pathwidth k consists of a path P and a collection of subtrees of pathwidth at most $k-1$, each connected to a single vertex on the path P (see [11]).

Let v_i be a vertex appearing in one of the subtrees of T. When v_i is presented in $T^<$, at time i, it will be colored by First-Fit using a color chosen from the first $3k-1$ colors of $\{1,2,\ldots\}$.

Let T^i be the subtree in which v_i appears. Let p_i be the path vertex to which the subtree T^i is connected. Let $V_i = \{v_j \mid j \leq i\}$ and $T_i^< = T^<[V_i]$, the online subgraph of $T^<$ induced by V_i.

Suppose that p_i is not present in $T_i^<$. Then the component of $T_i^<$ containing v_i is a tree of pathwidth at most $k-1$, disjoint from all other components of $T_i^<$. Thus, First-Fit will color v_i using a color chosen from the first $3k-2$ colors of $\{1,2,\ldots\}$.

Suppose that p_i is present in $T_{i-1}^<$. Then, in $T_i^<$, v_i will be adjacent to at most one vertex in the component of $T_{i-1}^<$ containing p_i. Suppose that this is the case and that this vertex has been colored with some color C_i. Consider the other components of $T_{i-1}^<$ to which v_i may become connected. Together with v_i, these form a tree of pathwidth at most $k-1$. First-Fit will require at most $3k-2$ colors to color this tree, but C_i cannot be used to color v_i. If $C_i \notin \{1,2,\ldots,3k-2\}$ then First-Fit will color v_i using a color chosen from $\{1,2,\ldots,3k-2\}$. If $C_i \in \{1,2,\ldots,3k-2\}$ then First-Fit will color v_i using a color chosen from $\{1,2,\ldots,3k-1\} - C_i$. If, in $T_i^<$, v_i is not connected to the component of $T_{i-1}^<$ containing p_i, then First-Fit will color v_i using a color chosen from $\{1,2,\ldots,3k-2\}$.

Let v_i be a vertex appearing in the path of T. When v_i is presented in $T^<$, at time i, it will be colored by First-Fit using a color chosen from the first $3k+1$ colors of $\{1, 2, \ldots\}$.

Let $V_i = \{v_j \mid j \leq i\}$ and $T_i^< = T^<[V_i]$, the online subgraph of $T^<$ induced by V_i.

In $T_i^<$, v_i may be adjacent to single vertices from each of many subtrees. Note that, in $T_{i-1}^<$, each of the subtrees that becomes connected to v_i is disjoint from all other components of $T_{i-1}^<$, so any such subtree will have been colored only with colors from $\{1, 2, \ldots, 3k-2\}$.

The path vertex v_i can also be connected to (at most) two other path vertices already colored. If v_i is not connected to any other path vertex then v_i will be colored by First-Fit using a color chosen from the first $3k-1$ colors of $\{1, 2, \ldots\}$. If v_i is connected to only one other path vertex then v_i will be colored by First-Fit using a color chosen from the first $3k$ colors of $\{1, 2, \ldots\}$. If v_i is connected to two other path vertices then v_i will be colored by First-Fit using a color chosen from the first $3k+1$ colors of $\{1, 2, \ldots\}$. $\square$

Lemma 6. *For each $k \geq 0$, there is an online tree $T^<$, of pathwidth k, such that First-Fit will use $3k+1$ colors to color $T^<$.*

Proof. The proof given for Lemma 5 suggests a way in which to present a tree of pathwidth k that will require $3k+1$ colors.

Let $k = 0$, then T consists only of an isolated vertex, and First-Fit requires $3k+1 = 1$ color to color $T^<$.

Let k be ≥ 1. Suppose that there is an online tree $T^<$, of pathwidth $k-1$, such that First-Fit is forced to use $3(k-1)+1 = 3k-2$ colors to color $T^<$. We build an online tree $T^<$, of pathwidth k, such that First-Fit is forced to use $3k+1$ colors to color $T^<$.

First present four sets of trees having pathwidth $k-1$. Each set contains $3k-2$ disjoint trees, all identical, which are presented one after another in such a way that First-Fit is forced to use $3k-2$ colors on each of them.

Now present a path vertex p_1 and connect p_1 to a single vertex from each of the trees in the first set so that the neighbours of p_1 use each of the colors in $\{1, 2, \ldots, 3k-2\}$. The vertex p_1 must be colored by First-Fit with color $3k-1$.

Now present a path vertex p_2 and connect p_2 to a single vertex from each of the trees in the second set so that the neighbours of p_2 use each of the colors in $\{1, 2, \ldots, 3k-2\}$. The vertex p_2 must be colored by First-Fit with color $3k-1$.

Now present a path vertex p_3 and connect p_3 to a single vertex from each of the trees in the third set, and also to the path vertex p_1, so that the neighbours of p_3 use each of the colors in $\{1, 2, \ldots, 3k-1\}$. The vertex p_3 must be colored by First-Fit with color $3k$.

Now present a path vertex p_4 and connect p_4 to a single vertex from each of the trees in the fourth set, and also to path vertices p_2 and p_3, so that the neighbours of p_4 use each of the colors in $\{1, 2, \ldots, 3k\}$. The vertex p_4 must be colored by First-Fit with color $3k+1$. $\square$

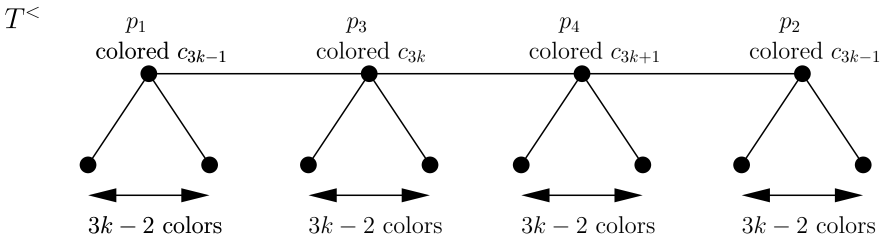

Fig. 4. Schema of online tree $T^<$ of pathwidth k on which First-Fit is forced to use $3k+1$ colors.

We now turn to online coloring of graphs having bounded treewidth. Related is Sandy Irani's [15] notion of a *d-inductive* graph, introduced earlier. Recall that a graph is d-inductive if there is an ordering $v_1, \ldots, v_n$ of its vertices such that for all i, v_i is adjacent to at most d vertices amongst $\{v_{i+1}, \ldots, v_n\}$.

This notion generalizes the notions of bounded treewidth, bounded degree, and planarity. For instance, a planar graph is 5-inductive. To see this, note that any planar graph must have a vertex of degree 5 or less. Call this v_1. Remove this vertex, and all edges adjacent to it. Repeat. Similarly, all graphs of treewidth k are k-inductive (Lemma 1). Irani gives an upper bound on the number of colours needed for First-Fit acting on d-inductive graphs.

Theorem 1 (Irani [15]). *If $G = (V, E)$ is d-inductive, then First-Fit will colour any online presentation of G with at most $O(d \log |V|)$ many colours.*

This bound is tight for the class, as this second theorem from Irani [15] shows.

Theorem 2 (Irani [15]). *For every online graph colouring algorithm A, and for every $d > 0$, there is a family of d-inductive graphs $\mathcal{G}$ such that for every $n > d^3$, there is a $G \in \mathcal{G}$ where G has n vertices and $A(G) = \Omega(d \log n)$.*

We give here a slightly weaker lower bound for First-Fit acting on bounded treewidth graphs.

Theorem 3. *For each $k > 0$, there is a family of graphs $\mathcal{G}$ of treewidth k, such that for every $n > k$, there is a $G \in \mathcal{G}$ where G has n vertices and an online presentation of G on which First-Fit will use $\Omega(\frac{k}{\log(k+1)} \log n)$ many colours.*

Proof. We build $\mathcal{G}$ by describing a single online presentation. Each G_n will be defined by a prefix of this presentation.

We first present $k+1$ vertices, $v_1, \ldots, v_{k+1}$, forming a clique at each step. First-Fit will colour these vertices, in order, with colours $c_1, \ldots, c_{k+1}$. We then present an isolated vertex v^1 which First-Fit will colour using c_1. To force First-Fit to colour the next vertex, v_{k+2}, using colour c_{k+2}, we will ensure that v_{k+2} is adjacent to $v_2, \ldots, v_{k+1}$ and also to v^1.

Note that the graph presented so far, which we denote as G_{k+2}, has treewidth k as required, with vertices $v_2, \ldots, v_{k+1}, v_{k+2}$, coloured using $c_2, \ldots, c_{k+1}, c_{k+2}$, forming a $k+1$ clique in G_{k+2}.

We now proceed inductively. Suppose that we have presented a graph G_m, of treewidth k, on which First-Fit has been forced to use colours c_1 through c_m, and that G_m contains a $k+1$ clique consisting of vertices $v_{m-k}, \ldots, v_m$, coloured using $c_{m-k}, \ldots, c_m$. We form G_{m+1}, having treewidth k, on which First-Fit is forced to use colour c_{m+1} as follows:

We first present a copy of $G_{m-k+1} \backslash v_{m-k+1}$, the graph up to the point where v_{m-k+1} was presented (and coloured with c_{m-k+1}), but with v_{m-k+1} left out. We then present a new vertex v_{m+1} and make it adjacent to $v_{m-k+1}, \ldots, v_m$ and also adjacent to all those vertices in the copy of $G_{m-k+1} \backslash v_{m-k+1}$ which are (copies of) neighbours of v_{m-k+1}. Since First-Fit was forced to use colour c_{m-k+1} on v_{m-k+1} in G_{m-k+1} it must have been the case that the neighbours of v_{m-k+1} in G_{m-k+1} used all the colours c_1 through c_{m-k}. Thus, the neighbours of v_{m+1} use all the colours c_1 through c_m, and so First-Fit will be forced to colour v_{m+1} using c_{m+1}.

We now show that G_{m+1} has treewidth k by constructing a tree decomposition of width k for G_{m+1}. Since $v_{m-k}, \ldots, v_m$ form a clique in G_m, it must be the case that in any tree decomposition of G_m these vertices will appear together in some bag, which we denote by B_m. To create a tree decomposition T_{m+1} of G_{m+1} having width k, we start with a tree decomposition T_m of width k for G_m. We create a new bag containing $v_{m-k+1}, \ldots, v_m, v_{m+1}$, denoted by B_{m+1}. We add an edge from B_m in T_m to B_{m+1}. We then replicate a tree decomposition T_{m-k+1} of width k for G_{m-k+1} and replace every occurrence of v_{m-k+1} in the bags of T_{m-k+1} by v_{m+1}. Finally, we add an edge from some bag in T_{m-k+1} that contains v_{m-k+1} to B_{m+1}. Note that, apart from the vertex v_{m+1}, there are no vertices in common between T_m and our modified T_{m-k+1}, so this completes the construction.

Now, if G_m consists of n_m vertices then G_{m+1} consists of $n_m + n_{m-k+1}$ vertices, G_{m+2} consists of $n_m + n_{m-k+1} + n_{m-k+2}$ vertices, and so on. G_{m+k} consists of $n_m + n_{m-k+1} + n_{m-k+2} + \cdots + n_m$ vertices. Thus, $n_{m+k} \leq (k+1)n_m$, giving $n_{1+k.d} \leq (k+1)^d$, for any $k > 0$, $d \geq 0$.

Rearranging, we get $1 + k.d > \frac{k}{\log(k+1)} \log n_{1+k.d}$, which gives us the required lower bound. $\square$

There are still many unknowns here. For instance, it is unclear how the parameters of treewidth and pathwidth interact on number of colors needed. It is also unclear as to what can be said on average. To make a *bad* online presentation of a graph G having low pathwidth one seems to need to *begin at the outer bags of some path decomposition for G and work in.* This would seem to be a rare event. Recently, Fouhy [12] has run some simulations and found that, in general, for random pathwith k graphs, we only ever seem to need $3k+1$ colours using First-Fit. This is not understood. Along with 0-1 behaviour, it is also suggestive of a more general theorem.

6 Bounded Persistence Pathwidth

In Section 3 we introduced a quite natural online presentation scheme that gives rise to graphs having *bounded persistence pathwidth*. We present the graph as a path decomposition, one node per timestep, where every vertex of the underlying graph belongs to at most l nodes of the path. The underlying idea is that a pathwidth 2 graph should look more like a "long 2-path" than a "fuzzy ball".

Recall that a path decomposition of width k in which every vertex of the underlying graph belongs to at most l nodes of the path has width k and *persistence* l, and that a graph that admits such a decomposition has *bounded persistence pathwidth*.

A related notion is *domino treewidth* introduced by Bodlaender and Engelfriet [5]. A *domino tree decomposition* is a tree decomposition in which every vertex of the underlying graph belongs to at most two nodes of the tree. *Domino pathwidth* is a special case of bounded persistence pathwidth, where $l = 2$.

Note that bounded persistence pathwidth gives us a natural characterization of graphs having both bounded pathwidth and bounded degree. If a graph G admits a path decomposition of width k and persistence l then it must be the case that all vertices in G have degree at most $k \cdot l$. On the other hand, if G has pathwidth k and maximum degree d then the persistence that can be achieved in any path decomposition of G must be "traded off" against the resulting width.

6.1 Complexity of Bounded Persistence Pathwidth

We now present some basic results regarding the complexity of recognizing graphs having bounded persistence path decompositions.

Persistence appears to be an interesting parameter in relation to graph width metrics and associated graph decompositions or layouts. However, in contrast to the case for *pathwidth*, which admits an FPT algorithm, deciding whether or not a given graph has *bounded persistence pathwidth* appears to be a hard problem.

We give some strong evidence for the likely parameterized intractability of both the Bounded Persistence Pathwidth problem and the Domino Pathwidth problem. We show that the Bounded Persistence Pathwidth problem is $W[t]$-hard, for all $t \in \mathbb{N}$, and we show that the Domino Pathwidth is $W[2]$-hard. These results mean that is likely to be impossible to find FPT algorithms for either of these problems, at least in the general case, unless an unlikely collapse occurs in the W-hierarchy.

Note that, even though we will show that the recognition problems are *hard*, in many real life instances we may reasonably expect to "know" that the persistence is relatively low, and, indeed be given such a decomposition.

A related result from [5] is that finding the domino treewidth of a general graph is $W[t]$-hard, for all $t \in \mathbb{N}$. Our first result relies on the following theorem from [6].

Theorem 4. *k-Bandwidth is $W[t]$-hard, for all $t \in \mathbb{N}$.*

k-Bandwidth is defined as follows:

Instance:	A graph $G = (V, E)$.
Parameter:	A positive integer k.
Question:	Is there a bijective *linear layout* of V, $f : V \to \{1, 2, \ldots, \lvert V \rvert\}$, such that, for all $(u, v) \in E$, $\lvert f(u) - f(v) \rvert \leq k$?

Bounded Persistence Pathwidth is defined as follows:

Instance:	A graph $G = (V, E)$.
Parameter:	A pair of positive integers (k, l).
Question:	Is there a path decomposition of G of width at most k and persistence at most l?

Domino Pathwidth is a special case of this problem, where $l = 2$.

Theorem 5. Bounded Persistence Pathwidth *is $W[t]$-hard, for all $t \in \mathcal{N}$*

Proof. We transform from k-Bandwidth.

Let $G = (V, E)$ be a graph and let k be the parameter. We produce $G' = (V', E')$ such that G' has a width $2(k + 1)^2 - 1$, persistence $k + 1$, path decomposition iff G has bandwidth at most k.

To build G' we begin with the original graph G, and alter it as follows:

1. for each vertex v in G, we introduce new vertices and form a clique of size $(k + 1)^2 + 1$ containing these vertices and v, call this C_v.
2. for each neighbour u of v (we can assume at most $2k$ of these, otherwise G cannot have bandwidth at most k), choose a unique vertex, c_{v_u}, from C_v (not v) and attach this vertex to all the vertices in C_u.

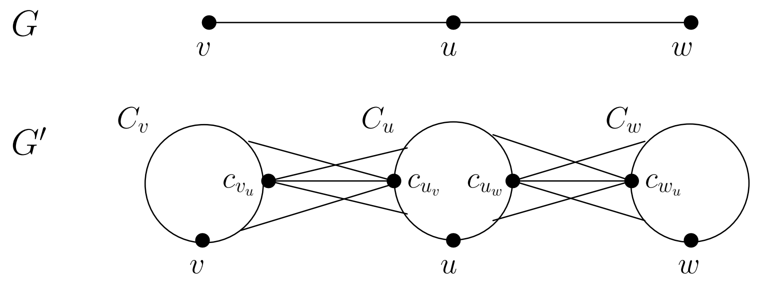

Fig. 5. Bounded persistence pathwidth, transformation from G to G'.

$\Leftarrow$ If G has bandwidth k, then the required decomposition for G' exists.

Let $\{v_0, \ldots, v_n\}$ be a layout of bandwidth k for G. To build the decomposition $(P, \mathcal{X})$ for G' we let the ith node of the decomposition, X_i, contain $\{v_i, \ldots, v_{i+k}\}$ plus C_i, plus each c_{j_r} connected to C_r with $j \leq i$ and $r \geq i$ or $r \leq i$ and $j \geq i$.

This fulfills the requirements for a path decomposition.

1. $\bigcup_{i \in I} X_i = V'$,
2. for every edge $\{v, w\} \in E'$, there is an $i \in I$ with $v \in X_i$ and $w \in X_i$, and
3. for all $i, j, k \in I$, if j is on the path from i to k in P, then $X_i \cap X_k \subseteq X_j$.

Each node contains at most $(k+1) + k(k+1) + (k+1)^2 = 2(k+1)^2$ vertices. Thus, the decomposition has width $2(k+1)^2) - 1$.

Any v_i from G appears in at most $k+1$ nodes, being nodes X_{i-k} up to X_i. Any c_{j_r} appears in at most $k+1$ nodes, being nodes X_j up to X_r, or X_r up to X_j, where $|j - r| \leq k$. Thus, the decomposition has persistence $k+1$.

$\Rightarrow$ If the required decomposition of G' exists, then G has bandwidth k.

At some point in the decomposition, a node $X_{v'}$, containing C_v, will appear for the first time. No other C_u, $u \neq v$ can appear in this node, as there is not room.

Pick a neighbour u of v for which C_u has already appeared. C_u must have appeared for the first time in some node $X_{u'}$, where $v' - u' \leq k$, since some c_{u_v} was present in this node which must appear with C_v at some point, and c_{u_v} cannot appear in more than $k+1$ nodes.

Pick a neighbour w of v for which C_w has not yet appeared. C_w must appear for the first time some node $X_{w'}$ where $w' - v' \leq k$, since there is some c_{v_w} in $X_{v'}$ which must be present with C_w at some point and c_{v_w} cannot appear in more than $k+1$ nodes.

If we lay out the vertices of G in the order in which the corresponding cliques first appear in the decomposition, then we have a layout of G with bandwidth k. □

Theorem 6. Domino Pathwidth *is* $W[2]$*-hard.*

Proof. We transform from k-Dominating Set, a fundamental $W[2]$-complete problem.

k-Dominating Set is defined as follows:

Instance: A graph $G = (V, E)$.
Parameter: A positive integer k.
Question: Does G contain a set of vertices $V' \subseteq V$, of size k, such that, $\forall u \in V$, $\exists v \in V'$ with $uv \in E$?

Let G be a graph and k the parameter. We produce G' such that G' has a width $K - 1$ domino path decomposition, where $K = k^2(k+4) + k(k+3)$, if and only if G has a dominating set of size k.

$G' = (V, E)$ consists of the following components:

- **Two anchors.** Take two cliques, each with K vertices, with vertex sets $A_1 = \{a_i^1 | 1 \leq i \leq K\}$ and $A_2 = \{a_i^2 | 1 \leq i \leq K\}$.
- **The graph thread.** Let $n = |V|$. Take $P = 2n + n^2 + (n+1)$ cliques, each with $Q = k^2(k+4)$ vertices, with vertex sets $C^i = \{c_r^i | 1 \leq r \leq Q\}$, for $1 \leq i \leq P$. Join them into a "thread" by choosing separate vertices in each clique, c_{start}^i and c_{end}^i, and identifying c_{end}^i with c_{start}^{i+1}. Identify c_{start}^1 with a_1^1, and identify c_{end}^P with a_1^2. Now, for each $1 \leq i \leq (P-1)$ cliques C^i and C^{i+1} have a vertex in common, C^1 has a vertex in common with A_1, and C^P has a vertex in common with A_2.
- **The vertex cliques.** For each i of the form $i = 2n + j.n + 1$ for $0 \leq j \leq n-1$ take the clique C^i from the graph thread and add another vertex to each such C^i, to make a clique with $Q+1$ vertices. Each of these n cliques, C_j^i, represents a vertex, v_j, of G.
- **The selector threads.** Take $2n$ cliques of size 2; n^2 cliques of size $k+3$ with vertex sets $S^i = \{s_r^i | 1 \leq r \leq k+3\}$, for $1 \leq i \leq n^2$; and another $2n$ cliques of size 2. Join them into a thread as for the graph thread, so that $2n$ size 2 cliques form the first portion of the thread, and $2n$ size 2 cliques form the last portion of the thread. Now, the first $2n$ size 2 cliques form a simple path having $2n+1$ vertices and $2n$ edges, where the last vertex in this path is also an element of S^1. For each $1 \leq i \leq (n^2 - 1)$ cliques S^i and S^{i+1} have a vertex in common. The last $2n$ size 2 cliques form a simple path having $2n+1$ vertices and $2n$ edges, where the first vertex in this path is also an element of S^{n^2}.

  For $1 \leq i \leq n$, let $\mathcal{S}_i$ denote the ith consecutive set of n cliques of size $k+3$, $\{S^{(i-1)n+1}, \ldots, S^{(i-1)n+n}\}$.

  Remove one (interior) vertex from the ith clique of $\mathcal{S}_i$, $S^{(i-1)n+i}$. If v_i is connected to v_j in G, remove one (interior) vertex from the jth clique of $\mathcal{S}_i$, $S^{(i-1)n+j}$.

  Make k copies of the selector thread component described here, and identify the first vertex of the ith thread with a_{i+1}^1, the last vertex of the ith thread with a_{i+1}^2.

  Each of these threads is used to select a vertex in a dominating set of G, if one exists.

$\Leftarrow$ Suppose G has a dominating set of size k. Then the required domino path decomposition of G', $PD(G')$, exists.

There are $P+2$ nodes in the decomposition, $\{X_0, \ldots, X_{P+1}\}$. We let A_1 be contained in X_0. For $1 \leq i \leq P$, we let X_i contain C^i from the graph thread, and we let the X_{P+1} contain A_2.

Note that X_1 must contain the first (size 2) clique of each of the selector threads, and X_P must contain the last (size 2) clique of each of the selector threads, by domino-ness. Each of these cliques has a vertex in common with the anchors, and this vertex can appear in only two nodes. It must appear in

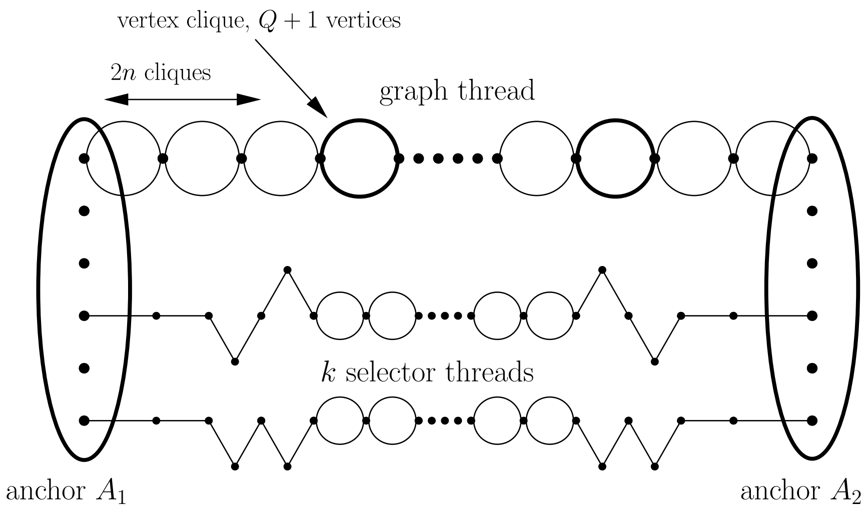

Fig. 6. Gadget for domino pathwidth transformation.

some node with its clique, and this cannot be the same node as the one where it appears with the anchor.

Suppose the dominating set of G is $\{v_{d_1}, v_{d_2}, \ldots, v_{d_k}\}$. We will align the first selector thread so that the v_{d_1}th clique of $\mathcal{S}_1$ in this thread appears in X_{2n+1}, the same node in which C^{2n+1}, the first vertex clique in the graph thread, appears. We will align the second selector thread so that the v_{d_2}th clique of $\mathcal{S}_1$ in this thread appears in X_{2n+1}, and so on. For each selector thread, we place each of the S^i cliques, $1 \leq i \leq n^2$, one per node consecutively, but we "fold" the size 2 cliques at the start of each selector thread, by placing 2 of these at at time into a node (i.e. 3 vertices per node) as many times as necessary, so as to ensure that the v_{d_i}th clique of $\mathcal{S}_1$ in ith thread occurs in the same node as C^{2n+1} from the graph thread. The size 2 cliques at the end of each selector thread are also folded to fit into the nodes remaining before A_2 is reached in X_{P+1}.

Exactly $(n-1)$ folds will be required altogether, for each selector thread. The folding of the size 2 cliques will not breach the width bound, since each fold contributes 3 to the bag size, over at most k threads, and at most $k(k+3)$ is permitted i.e $k+3$ per thread. Allowing $(n-1)$ folds will ensure that any of $\{v_1, \ldots, v_n\}$ can be positioned correctly.

If we align the selector threads this way, then each node containing a vertex clique from the graph thread will also contain a clique from at least one selector thread that is of size only $(k+2)$. Let $C^{2n+j.n+1}$ be a vertex clique representing vertex v_j from G. If v_j is in the dominating set then one of the selector threads will be aligned so that the jth clique of $\mathcal{S}_j$ from that thread appears in $X_{2n+j.n+1}$, and this clique has size only $(k+2)$. If v_j is a neighbour of some vertex v_i in the dominating set then one of the selector threads will be aligned so that the

ith clique of $\mathcal{S}_j$ from that thread appears in $X_{2n+j.n+1}$, and this clique has size only $(k+2)$.

The decomposition described here preserves domino-ness. X_0 and X_{P+1} each contain K vertices. Each interior node in the decomposition contains either a non-vertex clique in the graph thread along with at most $k(k+3)$ other vertices, or a vertex clique in the graph thread along with k cliques of size $(k+3)$ or $(k+2)$, where at least one of these cliques must have size $(k+2)$. Hence, each interior node contains at most K vertices

Thus, we have a domino path decomposition of width at most $K-1$, as required.

$\Rightarrow$ Suppose a domino path decomposition of G' with width K, $PD(G')$, exists, then G must have a dominating set of size k.

- Each of A_1 and A_2 must be the contained in an end node of $PD(G')$, since no threads can pass over these large cliques, and all threads have vertices in common with both of them. Let us assume that A_1 is contained in the first node, X_0.
- Only one clique from the graph thread can be contained in any node of $PD(G')$, since there is not enough room to admit more than one. Each clique of the graph thread must be contained in some node of $PD(G')$. By domino-ness, and a simple induction argument, we must have the same situation described in the first part of this proof. The decomposition $PD(G')$ must consist of $P+2$ nodes, A_1 is contained in the first node, X_0, A_2 is contained in the last node, X_{P+1}, and for $1 \leq i \leq P$, node X_i contains C^i from the graph thread.
- The first vertex clique in the graph thread must appear in a node containing a clique from $\mathcal{S}_1$ for each of the selector threads. The last vertex clique in the graph thread must appear in a node containing a clique from $\mathcal{S}_n$ for each of the the selector threads.
  The first vertex clique in the graph thread appears in node X_{2n+1}. The last vertex clique in the graph thread appears in node $X_{2n+n(n-1)+1}$. There are $2n+1$ nodes that occur before X_{2n+1} in the decomposition and $2n+1$ nodes that occur after $X_{2n+n(n-1)+1}$ in the decomposition. There are only $2n$ vertices occurring in a selector thread before the first clique of $\mathcal{S}_1$ is encountered. These are connected in a path, and by domino-ness, this path cannot stretch over more than $2n$ nodes. Similarly, the path at the end of the selector thread cannot stretch over more than $2n$ nodes.
- Each node in the decomposition, apart from the first and the last, contains a clique from the graph thread and so can contain at most k distinct S^i cliques from the selector threads.
  Each clique from the graph thread contains at least $k^2(k+4)$ vertices and each S^i clique contains at least $(k+2)$ vertices. In any node there is room for only $k(k+3)$ more vertices apart from the graph thread clique. $(k+1)$ distinct S^i cliques will consist of at least $(k+1)(k+2) = k(k+3)+2$ vertices.
- The arguments given here, along with domino-ness, force the following situation.

Every node in the decomposition from X_{2n+1}, which contains the first vertex clique of the graph thread, to $X_{2n+n(n-1)+1}$, which contains the last vertex clique of the graph thread, must contain exactly k S^i cliques, one from each of the selector threads. These must appear in the order in which they occur in the threads.

– For the width bound to be maintained, each node containing a vertex clique $C^{2n+j.n+1}$ from the graph thread must contain at least one S^i clique of size $(k+2)$. Thus, the $\mathcal{S}_1$ cliques that occur in node X_{2n+1} must correspond to k vertices that form a dominating set in G. □

7 Conclusions

We consider the work presented in this article to be a first step in a program to investigate the algorithmic ramifications of parameterized promise problems in the online setting. It seems apparent that graph width metrics, in particular, pathwidth, or metrics that are pathwidth-like, are a natural fit in this context.

The notion of persistence for a path decomposition is introduced. Whilst this natural parameter has a number of very interesting algorithmic implications, we establish that recognition of graphs having bounded persistence pathwidth is parametrically hard.

We see the present article as laying the foundations for our ideas. Future studies will be concerned both with other applications, and with making foundations for online descriptive complexity.

References

1. S. Arnborg, D. G. Corneil, and A. Proskurowski: *Complexity of finding embeddings in a k-tree.* SIAM J. Alg. Disc. Meth. 8, pp 277-284, 1987.
2. H. L. Bodlaender: *A linear time algorithm for finding tree decompositions of small treewidth.* SIAM J. Comput. 25, pp 1305-1317, 1996.
3. H. L. Bodlaender: *A partial k-arboretum of graphs with bounded treewidth.* Technical Report UU-CS-1996-02, Department of Computer Science, Utrecht University, Utrecht, 1996.
4. H. L. Bodlaender: *Treewdith: Algorithmic techniques and results.* Proc. 22nd MFCS, Springer-Verlag LNCS 1295, pp 19-36, 1997.
5. H. L. Bodlaender, J. Engelfreit: *Domino Treewidth.* J. Algorithms 24, pp 94-127, 1997.
6. H. L. Bodlaender, M. R. Fellows, M. T. Hallett: *Beyond NP-completeness for problems of bounded width: Hardness for the W-hierarchy.* Proc. 26th Annual Symposium on Theory of Computing, pp 449-458, ACM Press, New York, 1994.
7. H. L. Bodlaender and T. Kloks: *Efficient and constructive algorithms for the pathwidth and treewdith of graphs.* J. Algorithms 21, pp 358-402, 1996.
8. M. Chrobak, M. Slusarek: *On some packing problems related to dynamic storage allocation.* RAIRO Inform. Theor. Appl. 22, pp 487-499, 1988.
9. R. G. Downey, M. R. Fellows: *Parameterized Complexity* Springer-Verlag, 1999.

10. M. R. Fellows and M. A. Langston: *An analogue of the Myhill-Nerode theorem and its use in computing finite-basis characterizations.* Proceedings of the 30th Annual Symposium on Foundations of Computer Science, IEEE Computer Science Press, Los Alamitos, California, pp 520-525, 1989.
11. B. de Fluiter: *Algorithms for Graphs of Small Treewidth.* ISBN 90-393-1528-0, 1997.
12. J. Fouhy: *Computational Experiments on Graph Width Metrics.* M.Sc. thesis, Victoria University, Wellington, 2003.
13. A. Gyarfas, J. Lehel: *On-line and First Fit Coloring of Graphs.* J. Graph Theory, Vol. 12, No. 2, pp 217-227, 1988.
14. J. Lagergren: *Efficient parallel algorithms for graphs of bounded treewidth.* J. Algorithms 20, pp 20-44, 1996.
15. S. Irani: *Coloring inductive graphs on-line.* Proceedings of the 31st Annual Symposium on Foundations of Computer Science, Vol 2, pp 470-479, 1990.
16. H. A. Kierstead: *Recursive and On-Line Graph Coloring* In Handbook of Recursive Mathematics, Volume 2, pp 1233-1269, Elsevier, 1998.
17. H. A. Kierstead: *The Linearity of First Fit Coloring of Interval Graphs.* SIAM J. on Discrete Math, Vol 1, No. 4, pp 526-530, 1988.
18. H. A. Kierstead, J. Qin: *Coloring interval graphs with First-Fit.* (Special issue: Combinatorics of Ordered Sets, papers from the 4th Oberwolfach Conf., 1991), M. Aigner and R. Wille (eds.), Discrete Math. 144, pp 47-57, 1995.
19. H. A. Kierstead, W. A. Trotter: *An Extremal Problem in Recursive Combinatorics.* Congressus Numeratium 33, pp 143-153, 1981.
20. N. G. Kinnersley: *The Vertex Separation Number of a Graph equals its Path-Width.* Information processing Letters 42(6), pp. 345-350, 1992.
21. L. M. Kirousis, D. M. Thilikos: *The Linkage of a Graph.* SIAM Journal on Computing 25(3), pp. 626-647, 1996.
22. L. Lovasz, M. E. Saks, W. A. Trotter: *An On-Line Graph Coloring Algorithm with Sublinear Performance Ratio.* Bellcore Tech Memorandum, No.TM-ARH-013-014.
23. M. S. Manasse, L. A. McGeoch, D. D. Sleator: *Competitive Algorithms for Online Problems.*
24. J. Matousek and R. Thomas: *Algorithms for finding tree-decompositions of graphs.* J. Algorithms 12, pp 1-22, 1991. Proceedings of the 20th Annual ACM Symposium on Theory of Computing, pp 322-333, 1988.
25. C. M. McCartin: *Contributions to Parameterized Complexity* Ph.D. Thesis, Victoria University, Wellington, 2003.
26. L. Perkovic and B. Reed: *An Improved Algorithm for Finding Tree Decompositions of Small Width.* International Journal of Foundations of Computer Science 11 (3), pp 365-371, 2000.
27. B. Reed: *Finding approximate separators and computing treewdith quickly.* Proceedings of the 24th Annual Symposium on Theory of Computing, ACM Press, New York, pp 221-228, 1992.
28. N. Robertson and P. D. Seymour: *Graph minors - a survey.* Surveys in Combinatorics, I. Anderson (Ed.), Cambridge Univ. Press, pp 153-171, 1985.
29. N. Robertson, P. D. Seymour: *Graph minors II. Algorithmic aspects of tree-width.* J. Algorithms 7, pp 309-322, 1986.
30. D. D. Sleator, R. E. Tarjan: *Amortized Efficiency of List Update and Paging Rules.* Comunication of the ACM 28, pp 202-208, 1985.
31. M. Szegedy: private communication, reported in [16].