Computing pathwidth faster than $2^{n\star}$

Karol Suchan[1,2] and Yngve Villanger[3]

[1] Facultad de Ingeniería y Ciencias, Universidad Adolfo Ibáñez, Santiago, Chile.
[2] WMS, AGH University of Science and Technology, Cracow, Poland.
Email: karol@suchan.info
[3] University of Bergen, N-5020 Bergen, Norway.
Email: yngve.villanger@uib.no

Abstract. Computing the PATHWIDTH of a graph is the problem of finding a tree decomposition of minimum width, where the decomposition tree is a path. It can be easily computed in $\mathcal{O}^*(2^n)$ time by using dynamic programming over all vertex subsets. For some time now there has been an open problem if there exists an algorithm computing PATHWIDTH with running time $\mathcal{O}^*(c^n)$ for $c < 2^{\dagger}$. In this paper we show that such an algorithm with $c = 1.9657$ exists, and that there also exists an approximation algorithm and a constant τ such that an $opt + \tau$ approximation can be obtained in $\mathcal{O}^*(1.89^n)$ time.

1 Introduction

PATHWIDTH is a graph parameter defined in the same way as TREEWIDTH with the exception that the decomposition tree is requested to be a path. Both these parameters where introduced by Robertson and Seymour [17] in their graph minor project, and both parameters have algorithmic applications. Examples of these are algorithms that use dynamic programming over a decomposition. Such algorithms will have running times consisting of a polynomial part, and an exponential part that only depends on the width of the decomposition.

The TREEWIDTH and PATHWIDTH problems have been substantially studied for two decades, and this has resulted in massive literature. For an introduction see [2]. Both problems are NP-hard already in cocomparability graphs [11], and PATHWIDTH is NP-hard even in some restricted subclasses of chordal graphs and in weighted trees [16]. Some examples of efficient algorithms include polynomial time algorithms for CIRCLE and CIRCULAR-ARC graphs [15, 18], and PERMUTATION graphs [6], and fixed parameter tractable algorithms for both problems [3]. There also exist approximation algorithms for both problems, where Feige et al. [8] give the most recent algorithm for treewidth.

When it comes to exponential time algorithms the picture changes. For TREEWIDTH there are several results: Arnborg et al. give an $\mathcal{O}(n^{tw+2})$ time algorithm for treewidth, where tw is the treewidth of the graph[1]; Fomin et al.

⋆ This work is supported by the Research Council of Norway and by the Basal-CMM program of CONICYT, Chile.

† $f(n) = \mathcal{O}^*(g(n))$ if there is a polynomial function $p(n)$ s.t. $f(n) \leq p(n)g(n)$.

give $\mathcal{O}(c^n)$ time algorithms with $c < 2$[9, 10]. These results are based on the property that the treewidth problem decomposes into independent subproblems if one bag or separator of the tree decomposition under construction is known. To be more precise, each of the connected components of the graph remaining when the vertices of the bag are removed defines one independent subproblem. Adapting this approach to the pathwidth problem faces a major difficulty, as the remaining connected components cannot be treated independently, but have to be divided into a left and right set, with the components in a same set being merged together - this reflects the fact that in a tree decomposition there can be an arbitrary number of independent branches whereas in a path decomposition there may be only two of them. Thus, for a star on $n + 1$ vertices there will be 2^n possible partitions.

As a result, Pathwidth is often mentioned in the same breath as Cutwidth, Directed Optimal Linear Arrangement, Directed Feedback Arc Set (for a longer list see [4]), since all three can be expressed as vertex ordering problems, and for each of them the best known exact algorithm solves the problem in $\mathcal{O}^*(2^n)$ time. Furthermore, the bound can be reached for all three problems by dynamic programming over the 2^n different vertex subsets. This technique is often refereed to as the Held and Karp algorithm [13]. In the end of [4] the following citation can be found "*On the more theoretical side, it is interesting to try to improve the time bounds. Some problems appear to be hard, e.g., to improve upon the $\mathcal{O}^*(2^n)$ time for Pathwidth.*"

In this paper we show that the Pathwidth problem can be solved in (asymptotic) time less than 2^n by giving an algorithm that runs over $\mathcal{O}^*(c^n)$ vertex partitions, with $c = 1.9657$. In addition to this, we show that there exits a constant τ such that an $opt + \tau$ approximation can be obtained in $\mathcal{O}^*(1.89^n)$ time.

2 Preliminaries

All considered graphs are simple and undirected. For a graph $G = (V, E)$ we denote by $n = |V|$ the number of vertices and by $m = |E|$ the number of edges. The neighborhood of a vertex v is defined as $N(v) = \{u \in V : \{u, v\} \in E\}$, and the closed neighborhood is defined as $N[v] = N(v) \cup \{v\}$. The cardinality $|N(v)|$ is called the degree of v. For a vertex set W, we define its neighborhood as $N(W) = \bigcup_{v \in W} N(v) \setminus W$, and its closed neighborhood as $N[W] = N(W) \cup W$. A graph $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$ is called a subgraph of G. For a vertex set $W \subseteq V$, the subgraph of G induced by W is denoted $G[W] = (W, E_W)$, where E_W is the restriction of E to edges having both incident vertices in W. A graph G is not connected if it is possible to partition its vertex set into two non-empty subsets, such that no edge of G is incident to vertices in both parts; otherwise, G is connected. A clique is a graph containing an edge for each pair of distinct vertices. A cycle is a connected graph in which all vertices are of degree 2. A path is a connected graph in which all vertices are of degree at most 2 and that contains no cycle as a subgraph. In an intuitive way, we sometimes consider a path to be a permutation of its vertex set. A connected component of is an

inclusion maximal subset of vertices that induces a connected subgraph. For two vertices $a, b \in V$, a set $S \subseteq V$ is an a, b-separator if a and b belong to different connected components of $G[V \setminus S]$. S is a minimal a, b-separator if no proper subset of S is an a, b-separator. In general, S is a minimal separator in G if there exist $a, b \in V$, such that S is a minimal a, b-separator.

The pathwidth of a graph G is defined through *path decompositions*. A *path decomposition* of a graph $G = (V, E)$ is a pair (χ, P) in which $P = (V_P, E_P)$ is a path and $\chi = \{\chi_i \mid i \in V_P\}$ is a family of subsets of V, called *bags*, such that

(i) $\bigcup_{i \in V_P} \chi_i = V$;
(ii) for each edge $\{u, v\} \in E$ there exists an $i \in V_P$ such that both u and v belong to χ_i; and
(iii) for all $v \in V$, the set of nodes $\{i \in V_P \mid v \in \chi_i\}$ induces a connected subgraph of P.

The maximum of $|\chi_i| - 1$, $i \in V_P$, is called the *width* of the path decomposition. The *pathwidth* of a graph G, denoted by $pw(G)$, is the minimum width taken over all path decompositions of G. To help distinguish between the vertices of a graph G and the vertices of its decomposition path P we use the term *node* for the latter.

Treewidth is defined analogously to pathwidth, but is based on *tree decompositions* in place of *path decompositions*. A *tree decomposition* of a graph $G = (V, E)$, is a pair (χ, T) in which $T = (V_T, E_T)$ is a tree and $\chi = \{\chi_i \mid i \in V_T\}$ is a family of subsets of V, called *bags*, such that conditions $(i), (ii), (iii)$ of the path decomposition, with T put in place of P, hold. The treewidth is the minimum width over all possible tree decompositions.

3 Computation of pathwidth

A naive and well known way to compute the pathwidth of a graph, is to enumerate all $n!$ vertex orderings. Fix a permutation α, called a vertex ordering, of the vertex set V; $\alpha(i)$ gives the i^{th} vertex in α and $\alpha^{-1}(v)$ gives the position of v in α. Given a graph $G = (V, E)$ and a vertex ordering α, let $V_j = \bigcup_{i=1}^{j} \alpha(i)$. A path decomposition $(\chi, P)_\alpha$ of G can now be obtained from α as follows. For $i \in \{1, \ldots, n\}$ let χ_i be a bag containing $N[V_i] \setminus V_{i-1}$. In the opposite direction, we can consider a provided path decomposition (χ, P), where $\chi_1, \chi_2, \ldots, \chi_r$ are the bags in the order defined by P; then pick a vertex ordering α_P, such that for all pairs $u, v \in V$ for which all bags containing u have smaller index than those containing v ($u \in \chi_i$ and $v \in \chi_j$ implies that $i < j$) there is $\alpha_P^{-1}(u) < \alpha_P^{-1}(v)$. One simple way to obtain such an ordering is to number vertices in the order they disappear in $\bigcup_{j=i}^{r} \chi_j$ for increasing values of i. Notice that α_P defines a path decomposition of the same width as (χ, P), since by properties of a path decomposition χ_j contains $N(W)$ where $W = \bigcup_{i=1}^{j} \chi_i \setminus \chi_j$. Thus, pathwidth can be computed by enumerating all vertex orderings and returning the minimum obtained width.

This is not an efficient way to compute the pathwidth since $n!$ orderings have to be checked, but it provides the basic idea we will use to compute the pathwidth faster than $\mathcal{O}^*(2^n)$. With some simple arguments this approach can be adapted, so that only 2^n different vertex sets need to be considered. Let us introduce a scheme of dynamic programming of vertex sets of increasing cardinality that achieves this goal.

Let $pw(U, S)$ be defined as the minimum pathwidth of $G[U \cup S]$ where S is contained in the last bag. Recall that we consider a path as a permutation of nodes, hence the ordering. Let $\tilde{U} = V \setminus N[U]$ for a vertex set U. Consider now a vertex ordering α providing a path decomposition of minimum width, and let $V_j = \bigcup_{i=1}^{j} \alpha(i)$. Then $pw(V_1, N(V_1)) = |N[V_1]| - 1$ and $pw(V_i, N(V_i)) \leq max(pw(V_{i-1}, N(V_{i-1})), |(N(\tilde{V}_{i-1}) \cup N[v_i]) \setminus V_{i-1}| - 1)$. Finally, the pathwidth of G is $pw(V_n, \emptyset)$ - which gives us the minimum pathwidth where there are no restrictions on the last bag.

In the general case we do not know the optimal ordering, so for a vertex set U we simply set $pw(U, N(U))$ to be the minimum value obtained by testing every vertex of U as the last vertex added to the set. This is computable since the values for all smaller vertex sets are computed beforehand. One consequence of this is that the value for each of the 2^n vertex sets has to be computed. More formally, for every vertex set $U \subseteq V$ of increasing size we get that

$$pw(U, N(U)) = \min_{u \in U}(\max(pw(U \setminus \{u\}, N(U \setminus \{u\})), |N[U] \setminus (U \setminus \{u\})| - 1)). \quad (1)$$

To break the 2^n barrier we need some more insight into the structure of path decompositions. This comes from the study of minimal interval completions - let us briefly introduce these.

A graph $H = (V, F)$ is an *interval graph* if each vertex of H can be assigned an interval on the real line, such that $\{u, v\} \in F$ if and only if the intervals of u and v intersect. An interval graph H is said to be an *interval completion* of $G = (V, E)$ if H is an interval graph and $E \subseteq F$. In the case where $H' = (V, F')$ is not an interval graph for any edge set F', $E \subseteq F' \subset F$, we say that H is a *minimal interval completion* of G. Any interval graph H can be represented by a structure called a *clique path* (χ, P), where χ is the set of maximal cliques of H, each of which is associated to a node of the path P and, for each vertex $u \in V$, the set of nodes associated to maximal cliques of H containing u induces a subpath of P. Notice that this is also a path decomposition of G. If H is a minimal interval completion of G, we call the path decomposition (χ, P) of G defined by the clique path of H a *minimal path decomposition* of G. Notice that each bag of a minimal path decomposition of G is a maximal clique of H.

As it is enough to consider the set of its *minimal triangulations* in order to compute the *treewidth* of a graph, it is also enough to consider the *minimal interval completions* if we want to compute the *pathwidth* of a graph. Let (χ, P) be a path decomposition of minimum width. By adding edges necessary to make each bag χ_i induce a clique, for each $\chi_i \in \chi$, we obtain an interval completion H of G. Now remove edges from H until a minimal interval completion H' is obtained. Clearly, the minimal path decomposition defined by a clique path of

H' will have the same width as (χ, P). Thus, it is enough to consider minimal path decompositions, and not all possible path decompositions.

4 Faster computation of pathwidth

Minimal path decompositions, as path cliques of minimal interval completions, have some useful properties that we state in the following proposition.

Proposition 1. *Let (χ, P) be a minimal path decomposition of G. Take the bags $\chi_1, \chi_2, \ldots, \chi_r$ in the order defined by P, and let H be the minimal interval completion defined by (χ, P). Then each minimal separator S of H is given by $\chi_j \cap \chi_{j+1}$, for some j (cf. [7, 14]). Moreover, if $S = \chi_j \cap \chi_{j+1}$, then $S = N(\bigcup_{i=1}^{j} \chi_i \setminus S)$ and $S = N(\bigcup_{i=j+1}^{r} \chi_i \setminus S)$ (cf. [12]).*

For a vertex set $U \subset V$, we define $U^* = N[U] \setminus N(\tilde{U})$ as the *full set of* U, that is the set of vertices in $N[U]$ that do not have a neighbor in $V \setminus N[U]$. If $U = U^*$ we will refer to the set U as a *full set*.

Since it is sufficient to consider minimal path decompositions to compute pathwidth, and each prefix of a minimal path decomposition minus the adjacent separator corresponds to a full set $\bigcup_{i=1}^{j} \chi_i \setminus S$, it is sufficient to consider only vertex sets U, where $U = U^*$. But we need to be able to compute each full set, by means of polynomial time computation, directly from full sets of smaller cardinality - to avoid the bottleneck of checking $\Omega(2^n)$ subsets of V. One may notice the resemblance of this approach to the use of certain structures related to minimal triangulations to compute treewidth (cf. [9]).

Consider again the optimal vertex ordering α, and like before let $V_j = \bigcup_{i=1}^{j} \alpha(i)$, and in difference to before, let $U_j = V_j^*$. We now want to show that U_i can be obtained from U_{i-1} and $v_i = \alpha(i)$. Observe first by definition that $V_i \subseteq U_i$ and that $\tilde{V}_i = \tilde{U}_i$. If $v_i \notin N[\tilde{U}_{i-1}]$, then $v_i \in U_{i-1}$ and as a consequence $U_{i-1} = U_i$. Consider now the remaining case when $v_i \notin U_{i-1}$. Let $U = U_{i-1} \cup \{v_i\}$ and out goal is to show that $U^* = U_i$. First of all, we have that $V_i \subseteq U$, which means that $N[V_i] \subseteq N[U]$, and since every vertex in $U_{i-1} \setminus V_{i-1}$ only has neighbors in $N[V_i]$, then we get that $N[V_i] = N[U]$. Thus $V_i^* = U^*$. This means that the algorithm is exactly as before, but only runs over full sets and not over all 2^n vertex subsets.

For a full set U we say that a minimal interval completion H is an *extension* of U if $N(U)$ is a minimal separator, separating U from $\tilde{U}$ in H. In an analogous way, a minimal path decomposition (χ, P) of G is an extension of the full set U if there an integer j such that $N(U) = \chi_j \cap \chi_{j+1}$ and $U = \bigcup_{i=1}^{j} \chi_i \setminus N(U)$.

Observation 1 *For a full set U we have that $pw(U, N(U)) \geq |N(U)|$.*

Proof. Let (χ, P) be a minimal path decomposition of minimum width which is also an extension of the full set U. Let j be the integer such that $N(U) = \chi_j \cap \chi_{j+1}$ and $U = \bigcup_{i=1}^{j} \chi_i \setminus N(U)$. Since (χ, P) is a minimal path decomposition, it follows that $\chi_j \not\subseteq \chi_{j+1}$. Therefore, $|\chi_j| > |\chi_j \cap \chi_{j+1}| = |N(U)|$, and the claim statement follows.

□

Observation 2 *For a full set U, $\tilde{U}$ is also a full set.*

4.1 Bounding the number of base sets

We will now give an algorithm that verifies if the pathwidth of a given graph G is at most k in $\mathcal{O}^*(c^n)$ time for $c < 2$. Part one of this algorithm consist of generating a set of vertex partitions that the algorithm can use as intermediate steps to verify the existence of a path decomposition of width k. We will now define a base $\mathcal{B}$ containing 2 types of pairs of vertex sets:

1. (U, S) where $|U| \leq 0.404n$, and $S = N(U)$,
2. (U, S) where $|S| \leq 0.192n$, $U \subset V \setminus S$, $N(U) \subseteq S$, and each connected component of $G[\tilde{U}]$ contains at least three vertices.

It is important to notice that we only want to take advantage of a pair (U, S) if S turns out to be a minimal separator in a minimal interval completion of minimum width. This makes it safe to contain S in one bag, and by Observation 1 it follows that the pathwidth is at least $|S|$, that is the largest bag contains at least $|S| + 1$ vertices.

Lemma 1. *The number of pairs of vertex sets in $\mathcal{B}$ is $\mathcal{O}(n \cdot 1.9657^n)$.*

Proof. The number of pairs of vertex sets of type 1 is at most the number of vertex sets of size $0.404n$, thus

$$\sum_{i=1}^{0.404n} \binom{n}{i} \leq n \cdot \binom{n}{0.404n} = \mathcal{O}(n \cdot 1.9633^n).$$

For type 2, the number is at most

$$\sum_{i=1}^{0.192n} \binom{n}{i} \cdot 2^{(n-i)/3} \leq n \cdot \binom{n}{0.192n} \cdot 2^{(n-0.192n)/3} = \mathcal{O}(n \cdot 1.9657^n).$$

□

Our goal now will be to compute the pathwidth of G in $\mathcal{O}^*(|\mathcal{B}|)$ time.

4.2 Extending from the base sets

The set of pairs contained in $\mathcal{B}$ at this point will be referred to as *base pairs*, and these will be used to generate at most $n^2|\mathcal{B}|$ new pairs that will be added to the set $\mathcal{F}$. For each pair $(U, S) \in \mathcal{B}$ of type 1, we can compute $pw(U, S)$ by dynamic programming on increasing size over the set of pairs of type 1. The procedure for this is given in Equation 1. Put these pairs as the initial content of the set $\mathcal{F}$. For all of these pairs, U is a full set. Notice that initially we cannot compute the pathwidth of a type 2 pair (U, S). The function $pw(U, S)$ will be computed if pair (U, S) is used as a base for a pair added to the set $\mathcal{F}$.

All remaining pairs added to $\mathcal{F}$ will be created from a pair already in $\mathcal{F}$, by one of three rules. Before describing these rules, the vertices are numbered in any order from 1 to n, where $\ell(v)$ refers to the number assigned to vertex v. The purpose of the numbering is to ensure that a rule creates a unique output from a given pair. Some of the pairs in $\mathcal{F}$ will also be assigned a mark.

Rule 1 *(Monotone Push Rule) Given a pair $(U, S) \in \mathcal{F}$, and a vertex $u \in S$ such that $|N(u) \cap \tilde{U}| = 1$ and $|N(v) \cap \tilde{U}| > 1$ for any vertex $v \in S$ where $\ell(v) < \ell(u)$. Let $W = (U \cup \{u\})^*$ and add the pair $(W, N(W))$ to the set $\mathcal{F}$, and set $pw(W, N(W)) = pw(U, S)$. Mark $(W, N(W))$ if pair (U, S) is marked, and $(W, N(W))$ is not a pair in $\mathcal{B}$.*

Lemma 2. *The Monotone Push Rule is safe for pathwidth.*

Proof. Let us assume that there exists an interval completion H and a minimal path decomposition $\chi_1, \chi_2, \ldots, \chi_r$ of width k, where $\bigcup_{i=1}^{j} \chi_i \setminus S = U$ and $\bigcup_{i=1}^{j} \chi_i \setminus U = S$. Since S is a minimal separator in H, then $|S| \leq k$. Let w be the unique neighbor of u in $\tilde{U}$, and let p be the smallest number such that $w \in \chi_p$. Vertex u is contained in S and thus also in χ_j, and in χ_t for $j \leq t \leq q$ where q is the highest number such that $u \in \chi_q$. Because of the edge uw we have that $p \leq q$. Construct a new path decomposition as follows: Remove u from any bag χ_t where $j < t$, and add w to any χ_t where $j < t < p$, and insert a bag between χ_j and χ_{j+1} containing $S \cup \{w\}$. Clearly this is a legal path decomposition of the same width.

□

Rule 2 *(Component Push Rule) Given a pair $(U, S) \in \mathcal{F}$ where Rule 1 can not be applied, and a vertex $u \in \tilde{U}$ such that $|S| + |C| \leq k + 1$ where C is the connected component of $G[V \setminus (U \cup S)]$ containing u, and for any vertex $v \in \tilde{U}$ where $\ell(v) < \ell(u)$ the connected component C_v of $G[V \setminus (U \cup S)]$ containing v we have that $|S| + |C_v| \geq k + 1$. Let $W = (U \cup C)^*$, and add the pair $(W, N(W))$ to $\mathcal{F}$ and set $pw(W, N(W)) = max(pw(U, S), |S| + |C| - 1)$. Mark $(W, N(W))$ if pair (U, S) is marked, and $(W, N(W))$ is not a pair in $\mathcal{B}$.*

Lemma 3. *The Component Push Rule is safe for pathwidth.*

Proof. The proof is similar as the one for Rule 1. Let us assume that there exists an interval completion H and a minimal path decomposition $\chi_1, \chi_2, \ldots, \chi_r$ of width k, where $\bigcup_{i=1}^{j} \chi_i \setminus S = U$ and $\bigcup_{i=1}^{j} \chi_i \setminus U = S$. Insert a bag between χ_j and χ_{j+1} containing $S \cup C$, and remove vertices of C from χ_t for each $t > j$. Clearly this is a legal path decomposition of width at most k.

□

For each pair in $\mathcal{F}$, at each step of its construction, at most one of Rules 1, 2 can be applied in only one possible way. Therefore these two rules can be applied recursively without producing more than $\mathcal{O}(n)$ new pairs. The final rule is a brute force search rule and applying it recursively would yield an exponential growth in the number of new pairs. Thus, we use the marks to apply it at most once in any sequence of rule applications starting from a base pair.

Rule 3 *(Push Rule) Given an unmarked pair $(U, S) \in \mathcal{F}$, where neither of Rules 1,2 applies, and a vertex $u \in S$ such that $max(pw(U, S), |N[U \cup \{u\}] \setminus U| - 1) \leq k$. Let $W = (U \cup \{u\})^*$ and add the marked pair $(W, N(W))$ to the set $\mathcal{F}$, and set $pw(W, N(W)) = max(pw(U, S), |N[U \cup \{u\}] \setminus U| - 1)$.*

Rule 3 is safe by Equation 1.

4.3 The algorithm

It is now time to populate the set $\mathcal{F}$, and verify if the pathwidth is at most k. First step is to add the seeds, that is type 1 pairs from $\mathcal{B}$. Notice that $pw(U, S) = pw(U^*, N(U^*))$, so for every type 1 pair in $\mathcal{B}$, where $pw(U, S) \leq k$, add an unmarked full set $(U^*, N(U^*))$ to $\mathcal{F}$ and set $pw(U^*, N(U^*)) = pw(U, S)$. From this set of seeds, rules 1,2,3 are applied if possible. Eventually, when no rule can be further applied, the set $\mathcal{F}$ satisfies:

Lemma 4. $|\mathcal{F}| \leq n^2|\mathcal{B}|$.

Proof. Let us assume that $\mathcal{B} \subseteq \mathcal{F}$. All elements of $\mathcal{F} \setminus \mathcal{B}$ are obtained by applying Rule 1, Rule 2, or Rule 3. Thus we can say that a pair $(U, S) \in \mathcal{F} \setminus \mathcal{B}$ is obtained from a base case (W, R) contained in $\mathcal{B}$. Let $\mathcal{F}(W, R)$ be the set of elements in $\mathcal{F} \setminus \mathcal{B}$ originating from the base case (W, R), and where none of the intermediate pairs is a base pair. This gives a partitioning of $\mathcal{F} \setminus \mathcal{B}$ into $|\mathcal{B}|$ sets. Let $\mathcal{F}(W, R)$ be a set of maximum cardinality, and notice that $|\mathcal{F}| \leq |\mathcal{F}(W, R)| \cdot |\mathcal{B}|$. We want to prove that $|\mathcal{F}(W, R)| \leq n^2$. First observation is that Rule 1 and Rule 2 are applied recursively from (W, R) until a pair (W', R') is obtained, where neither of the two rules applies. Since Rule 2 is only applied if Rule 1 does not apply, and the output for both rules is unique due to the labeling, there is a directed path from (W, R) to (W', R'), with no extra leaves. Since the size of the full set increases by at least one at each time a rule is applied, the length of the path is bounded by n. Next, by Rule 3, there might be one marked pair created for each vertex $x \in V \setminus W'$. Thus we obtain at most n marked pairs, each of which might be further developed by Rule 1 and Rule 2 into a path of at most n pairs. But notice that all these pairs are marked, so when Rules 1, 2 cannot be further applied, the development stops. By this construction we obtain a tree, where (W, R) is the root, and with at most $|V \setminus W'|$ leaves and where the distance from the leaves to the root is at most n. Thus, the number of vertices, and thus also pairs in $\mathcal{F}(W, R)$ is at most n^2.

□

Lemma 5. *Consider the set $\mathcal{F}$ computed for a value k, where none of the rules can be applied. Then the pathwidth is at most k if and only if there exists a full set U such that*

- $(U, N(U)) \in \mathcal{F}$,
- $(\tilde{U}, N(U)) \in \mathcal{F}$,
- $pw(U, N(U)) \leq k$, *and*

– $pw(\tilde{U}, N(U)) \leq k$.

Proof. Notice that $pw(U, N(U)) \leq k$ implies that there exists a path decomposition of $G[N[U]]$ of width at most k, where $N(U)$ is contained in the last bag. In the case where $pw(U, N(U)) \leq k$ and $pw(\tilde{U}, N(U)) \leq k$, it is possible to put the two path decompositions of respectively $G[N[U]]$ and $G[V \setminus U]$ next to each other and obtain a path decomposition for G of width k. We can see it as the one extends the other.

For the opposite direction, assume on the contrary that the pathwidth of G is k. Let α be a vertex ordering providing a path decomposition of width at most k. Let $V_j = \bigcup_{i=1}^{j} \alpha(i)$, and let $U_j = V_j^*$. Define $b(\alpha)$ to be the largest integer j such that $(U_i, N(U_i)) \in \mathcal{F}$ and $pw(U_i, N(U_i)) \leq k$ for every $i \in \{1, \ldots, j\}$. Let now α be the vertex ordering with largest $b(\alpha)$ among all orderings providing a path decomposition of width at most k.

Let $j = b(\alpha)$, and consider the pair (U_j, S_j). If $|S_j| < k$, then $G[\tilde{U}_j]$ contains no connected component C where $|C| \leq 2$, since Rule 2 can be applied to these. By the safeness of Rule 2 it follows that there exists an ordering α' which provides a path decomposition of width at most k, where $b(\alpha) < b(\alpha')$. Notice that Rule 2 is applied if possible.

Since Rule 2 cannot be applied, then every connected component of $G[\tilde{U}_j]$ contains at least three vertices. Furthermore, (U_j, S_j) is not of type 1 in $\mathcal{B}$ since this would imply that (U_j, S_j) is unmarked and (U_{i+1}, S_{i+1}) would have been obtained by Rule 3.

As a result $0.404n \leq |U_i|$. By the conditions of the lemma $(\tilde{U}_j, S_j) \notin \mathcal{F}$, and since all pairs in $\mathcal{B}$ of type 1 are added to $\mathcal{F}$ we get that $0.404n \leq |\tilde{U}_j|$ and thus $|S_j| < 0.192n$. Now (U_j, S_j) is of type 2 in $\mathcal{B}$, since $|S_j| < 0.192n$ and $G[\tilde{U}_j]$ only contains connected components of size at least three, which is also a contradiction since (U_j, S_j) would be unmarked and (U_{j+1}, S_{j+1}) would be obtained by Rule 3.

So we know that $|S_j| = k$. Let x be the vertex such that $U_{j+1} = (U_j \cup \{x\})^*$. Observe that there are no connected components in $G(\tilde{U}_j)$ of size 1, since $|S_j| = k$ and Rule 2 can then be applied to these. This implies that x has at least one neighbor in $\tilde{U}_j$. By fixing the next vertex to be x we have also fixed the next bag to be $S_j \cup (N[x] \setminus U_j)$. This implies that $|N(x) \cap \tilde{U}_j| = 1$ and that $x \in S_j$, since otherwise $k + 1 < |S_j \cup (N[x] \setminus U_j)|$. Thus, this is a monotone push, and Rule 1 can be applied, something which is a contradiction to the assumption that α is the vertex ordering with the maximum value of $b(\alpha)$.

□

Theorem 3. *There exists an algorithm that verifies if the pathwidth of a graph G on n vertices is at most k in $\mathcal{O}^*(1.9657^n)$ time.*

Proof. By Lemma 1 the set $\mathcal{B}$ contains at most $\mathcal{O}^*(1.9657^n)$ pairs, and by Lemma 4, the same bound holds for $\mathcal{F}$. Finally by Lemma 5, $\mathcal{F}$ contains a pair (U, S) such that

– $(U, N(U)) \in \mathcal{F}$,

– $(\tilde{U}, N(U)) \in \mathcal{F}$,
– $pw(U, N(U)) \leq k$, and
– $pw(\tilde{U}, N(U)) \leq k$

if and only if the pathwidth of G is at most k.

□

5 Approximation of the pathwidth

This section gives an algorithm that approximates the pathwidth up to $opt + \tau$ for some constant τ, in $\mathcal{O}^*(\binom{n}{tn})$ time, where $0 < t < 1$ is a constant such that $\binom{n}{(1-2t)n} \cdot 2^{2tn/\tau} \leq \binom{n}{tn}$. For instance, setting τ to $2, 20, 200$, and 10.000 gives algorithms with running times $\mathcal{O}^*(1.9427^n), \mathcal{O}^*(1.9047^n), \mathcal{O}^*(1.8921^n), \mathcal{O}^*(1.88997^n)$, respectively. The approximation algorithm can be obtained by a slight adaptation of the exact algorithm for pathwidth. The base set $\mathcal{B}$ is obtained with two types of pairs as before.

1. (U, S) where $|U| \leq tn$, and $S = N(U)$,
2. (U, S) where $|S| \leq (1-2t)n$, $U \subset V \setminus S$, $N(U) \subseteq S$, and each connected component of $G[\tilde{U}]$ contains at least τ vertices.

The second modification is to replace Rule 2 by the rule given below.

Rule 4 *(Component push Rule) Given a pair $(U, S) \in \mathcal{F}$ where Rule 1 cannot be applied, and a vertex $u \in \tilde{U}$ such that $|S| + |C| \leq k + 1 + \tau$, where C is the connected component of $G[V \setminus (U \cup S)]$ containing u, and for any vertex $v \in \tilde{U}$, where $\ell(v) < \ell(u)$, the connected component C_v of $G[V \setminus (U \cup S)]$ containing v we have that $|S| + |C_v| \geq k + 1 + \tau$. Then let $W = U \cup C$, and add the pair (W, S) to $\mathcal{F}$ and set $pw(W, S) = max(pw(U, S), |S| + |C| - 1 - \tau)$. Mark (W, S) if pair (U, S) is marked, and $(W, N(W))$ is not a pair in $\mathcal{B}$.*

In an analogous way as before, we create a set $\mathcal{F}$ using type 1 in $\mathcal{B}$ as the seeds, and then apply Rules 1,4, and 3 if possible. Rule 4 behaves exactly as Rule 2 when it comes to generating new pairs for the set $\mathcal{F}$, thus by Lemma 4 $|\mathcal{F}| \leq n^2|\mathcal{B}|$.

Corollary 1. *Consider the set $\mathcal{F}$ computed for a value k, where no new information can be obtained by applying a rule. Then the pathwidth is at most $k + \tau$ if and only if there exists a full set U such that*

– $(U, N(U)) \in \mathcal{F}$,
– $(\tilde{U}, N(U)) \in \mathcal{F}$,
– $pw(U, N(U)) \leq k$, *and*
– $pw(\tilde{U}, N(U)) \leq k$.

Theorem 4. *There exists an approximation algorithm that verifies if the pathwidth of a graph G on n vertices is at most $k + \tau$ in $\mathcal{O}^*(\binom{n}{(1-2t)n} \cdot 2^{2tn/\tau} + \binom{n}{tn})$ time for some constant $0 \leq t \leq 1$.*

Proof. By Lemma 1 the set $\mathcal{B}$ contains at most $\mathcal{O}^*(\binom{n}{(1-2t)n} \cdot 2^{2tn/\tau} + \binom{n}{tn})$ pairs, and by Lemma 4 the same bound holds for $\mathcal{F}$. Finally by Corollary 1 $\mathcal{F}$ contains a pair (U, S) such that

- $(U, N(U)) \in \mathcal{F}$,
- $(\tilde{U}, N(U)) \in \mathcal{F}$,
- $pw(U, N(U)) \leq k$, and
- $pw(\tilde{U}, N(U)) \leq k$

if and only if the pathwidth of G is at most $k + \tau$.

□

6 Conclusion and open questions

It is well known that Held and Karp's dynamic programming algorithm [13] can be adapted to solve a long sequence of problems which are definable as a vertex ordering problem. Their approach gives running time $\mathcal{O}^*(2^n)$, and is based on doing dynamic programming over all the 2^n vertex subsets of the graph. Some examples of such problems are Treewidth, Pathwidth, Cutwidth, Directed Optimal Linear Arrangement, and Directed Feedback Arc Set. Among these problems, treewidth is the only problem prior to this work where a $\mathcal{O}^*(c^n)$ algorithm for $c < 2$ was known. It would be interesting to know if other problems that are expressible as vertex ordering problems and can be solved by Held and Karp's approach, have an algorithm of running time $\mathcal{O}^*(c^n)$ for $c < 2$.

One of the techniques used to compute the pathwidth is to guess a balanced separator in the cases where the rules do not apply. In [5] the approach of guessing the middle cut or bag was used to give a $\mathcal{O}^*(2.9512^n)$ polynomial space algorithm for computing the treewidth of a graph. It is not clear how or if it is possible to extend this approach to other problems in the list above. Guessing a balanced separator has a natural limitation in $\binom{n}{n/3}$, which is the lower bound obtained for the additive approximation algorithm. Finally the possible improvement of the running time for computing pathwidth exact is left as an open question, and especially if the bound $\binom{n}{n/3}$ can be obtained. Improvement on the running time would probably require some new ideas, since both the Rule's 1 and 2 rely on arriving at a very similar situation. For instance if Rule 1 was extended to have two neighbors instead of one, then the order of applying the rule would make a difference.

References

1. S. Arnborg, D. G. Corneil, and A. Proskurowski, *Complexity of finding embeddings in a k-tree*, SIAM J. Algebraic Discrete Methods, 8 (1987), pp. 277–284.
2. H. L. Bodlaender, *A tourist guide through treewidth*, Acta Cybern., 11 (1993), pp. 1–22.
3. H. L. Bodlaender, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25 (1996), pp. 1305–1317.
4. H. L. Bodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos, *On exact algorithms for treewidth*, Tech. Rep. UU-CS-2006-032, Department of Information and Computing Sciences, Utrecht University, 2006.
5. H. L. Bodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos, *On exact algorithms for treewidth*, in ESA, Y. Azar and T. Erlebach, eds., vol. 4168 of Lecture Notes in Computer Science, Springer, 2006, pp. 672–683.
6. H. L. Bodlaender, T. Kloks, and D. Kratsch, *Treewidth and pathwidth of permutation graphs*, SIAM J. Discrete Math., 8 (1995), pp. 606–616.
7. P. Buneman, *A characterization of rigid circuit graphs*, Discrete Math., 9 (1974), pp. 205–212.
8. U. Feige, M. Hajiaghayi, and J. R. Lee, *Improved approximation algorithms for minimum weight vertex separators*, SIAM J. Comput., 38 (2008), pp. 629–657.
9. F. V. Fomin, D. Kratsch, I. Todinca, and Y. Villanger, *Exact algorithms for treewidth and minimum fill-in*, SIAM J. Comput., 38 (2008), pp. 1058–1079.
10. F. V. Fomin and Y. Villanger, *Treewidth computation and extremal combinatorics*, in ICALP (1), L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, eds., vol. 5125 of Lecture Notes in Computer Science, Springer, 2008, pp. 210–221.
11. M. Habib and R. H. Mhring, *Treewidth of cocomparability graphs and a new order-theoretic parameter*, Order, 11 (1994), pp. 47–60.
12. P. Heggernes, K. Suchan, I. Todinca, and Y. Villanger, *Characterizing minimal interval completions*, in STACS, W. Thomas and P. Weil, eds., vol. 4393 of Lecture Notes in Computer Science, Springer, 2007, pp. 236–247.
13. M. Held and R. M. Karp, *A dynamic programming approach to sequencing problems*, in Proceedings of the 1961 16th ACM national meeting, New York, NY, USA, 1961, ACM, pp. 71.201–71.204.
14. C. W. Ho and R. C. T. Lee, *Counting clique trees and computing perfect elimination schemes in parallel*, Information Processing Letters, 31 (1989), pp. 61–68.
15. T. Kloks, *Treewidth of circle graphs*, Int. J. Found. Comput. Sci., 7 (1996), pp. 111–120.
16. R. Mihai and I. Todinca, *Pathwidth is np-hard for weighted trees*, in FAW, X. Deng, J. E. Hopcroft, and J. Xue, eds., vol. 5598 of Lecture Notes in Computer Science, Springer, 2009, pp. 181–195.
17. N. Robertson and P. D. Seymour, *Graph minors. II. Algorithmic aspects of tree-width*, Journal of Algorithms, 7 (1986), pp. 309–322.
18. K. Suchan and I. Todinca, *Pathwidth of circular-arc graphs*, in WG, A. Brandstädt, D. Kratsch, and H. Müller, eds., vol. 4769 of Lecture Notes in Computer Science, Springer, 2007, pp. 258–269.