1 Pathwidth: Definition

Definition 16.1. A pathwidth-k (pw-k) graph is constructed as follows.

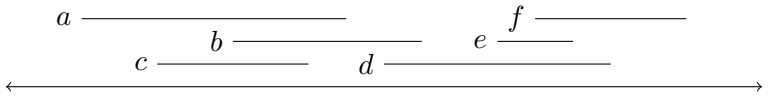


Figure 16.1: Intervals on $\mathbb{R}$

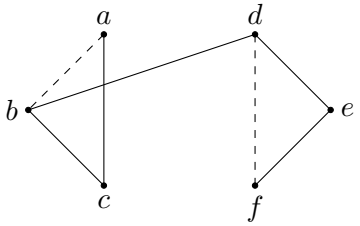


Figure 16.2: One graph corresponding to the intervals above

Every vertex of the graph is an interval in $\mathbb{R}$. We can only connect 2 vertices if their intervals intersect (but we don't have to). For example, from the intervals in Figure 16.1 above, we can obtain the graph in Figure 16.2. Each dotted edge represents an edge that we could have included, but we didn't.

More formally, a pw-k graph satisfies the following three properties:

- Every vertex of the graph is an interval in $\mathbb{R}$
- We can only add edges between vertices whose intervals intersect.
- $\forall x \in \mathbb{R}$, x is in at most $k+1$ intervals.

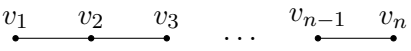
Each pw-k is a collection of graphs. From this we have the following definition.

Definition 16.2. A graph G has pathwidth k (pw k) if k is the smallest integer such that G is in the class of pw-k graphs.

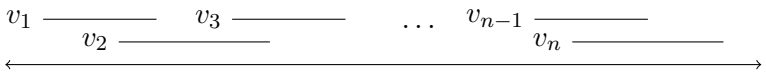
We make the following observations on the pathwidths of simple graphs.

Observation 16.3. A path has pw 1.

Proof. Starting with the path



We can represent it using intervals by



Note that every point is in at most 2 intervals.

We also need to show that a path is not in pw-0. Note that for the interval representation of pw-0, no two intervals can intersect. Hence, only the empty graph can be in pw-0. We conclude that a path has pathwidth 1. □

Note that we define pw-k to be at most $(k+1)$ intervals, precisely because in this way, a path has pathwidth 1.

Observation 16.4. A cycle has pw ≤ 2.

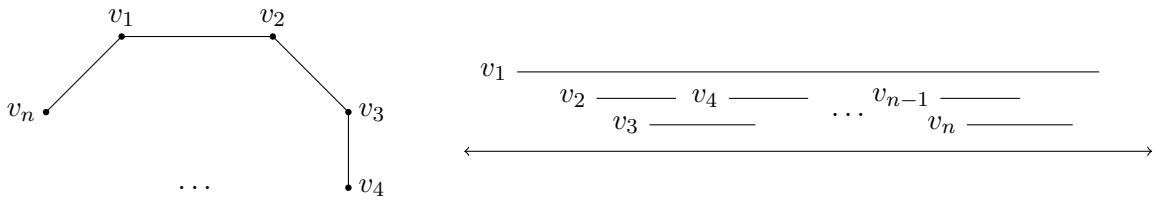


Figure 16.3: The graph and the interval representation of a cycle

Proof. See Figure 16.3. Note that every point is in at most 3 intervals. □

It's possible to show that a cycle is not in pw-1, but we'll skip the proof. In general, lower bounding the pathwidth of a graph can be difficult.

Observation 16.5. A star graph has pw 1.

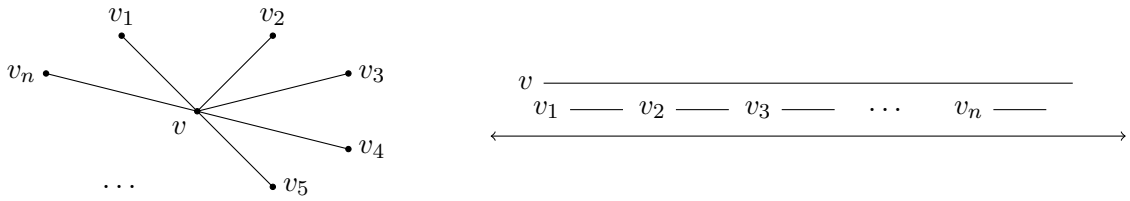


Figure 16.4: The graph and the interval representation of a star graph

Proof. See Figure 16.4. Note that every point is in at most 2 intervals.

The star graph is not in pw-0, since it's not an empty graph. Hence, it has pathwidth 1. □

Observation 16.6. A $(k+2)$-clique is not pw $\leq k$. That is, the class of pw-k graphs does not contain the $(k+2)$-clique.

Proof. Assume for the sake of contradiction that it is in pw-k. We can find $k+2$ intervals in $\mathbb{R}$ such that they intersect pairwise. We claim that there exists a point $x \in \mathbb{R}$ that is included in all $(k+2)$ intervals.

The claim follows from Helly's theorem for $d = 1$. Helly's theorem states that, consider n convex subsets $X_1, \dots, X_n$ in $\mathbb{R}^d$ for $n > d$, and assume that the intersection of every $d+1$ of these sets is nonempty. Then the whole collection has a nonempty intersection. Note that in this case, the convex sets in 1 dimension are just the intervals.

A more elementary proof of the claim is as follows: let the intervals be $[a_i, b_i]$ and suppose they pairwise intersect. Take the interval $[a_i, b_i]$ that maximizes a_i. We claim that the point a_i is in every interval. Suppose not: it's not in interval $[a_j, b_j]$. Then, either $a_i > b_j$ or $a_i < a_j$. The latter case cannot happen by the definition of a_i. So $a_i > b_j$. But then $[a_i, b_i]$ and $[a_j, b_j]$ don't intersect, a contradiction.

Hence, we can find $x \in \mathbb{R}$ that is included in all $(k+2)$ intervals. But pw-k requires that x be in at most $(k+1)$ intervals, a contradiction. □

2 Pathwidth: Discretized Definition

The interval definition of pathwidth makes sense intuitively, but for algorithms, we often prefer a discretized definition.

Consider the intervals from Figure 16.1, and consider discretizing the picture by slicing a vertical plane through the real line from left to right. Each time an "event" happens, i.e. when an interval begins or ends, we record the set of intervals that our plane touches. We obtain the following diagram.

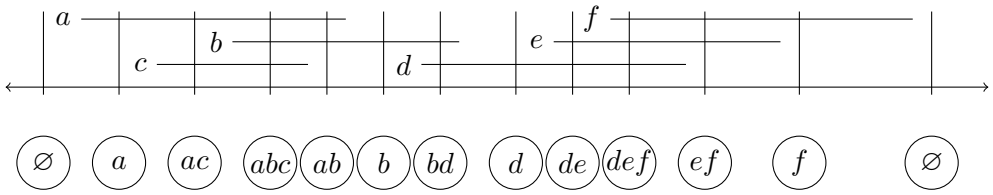


Figure 16.5: Creating a chain of sets of vertices by slicing a plane from left to right

We can now represent the graph using the chain of sets of vertices at the bottom.

More formally, we have the following definition.

Definition 16.7 (Path Decomposition)**.** A path decomposition of G is a sequence $(X_1, \ldots, X_N)$ of bags $X_i \subseteq V$ such that

1. Every vertex belongs to at least one bag.
2. For each edge in G, there is a bag containing its endpoints.
3. For every vertex v, the set of bags containing v forms a connected interval of $(X_1, \ldots, X_N)$.

We note that item 3 is equivalent to the following:

$$v \in X_i,\ v \in X_k \implies \forall j \in [i,k], v \in X_j.$$

We require N to be finite, but note that we can always find a path decomposition with $N = O(n)$: start from the interval representation, we obtain a path decomposition via the slicing of the plane shown in Figure 16.5. Since each interval only generates 2 events (start and finish), we have $2n+1$ events (including the starting $\varnothing$), so at most $O(n)$ bags are needed.

We define the width of a path decomposition as the maximum size of the bags.

Definition 16.8. The pathwidth of G is the smallest $w-1$ such that there exists a path decomposition of G of width w.

Observation 16.9. For a path with vertices $v_1, \ldots, v_n$, the following are two valid path decompositions:

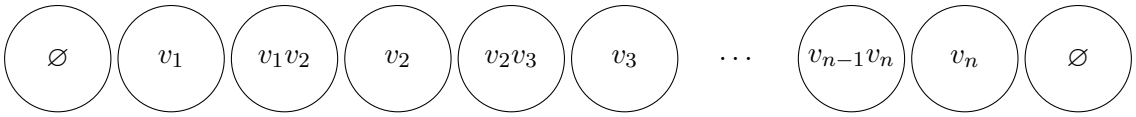


Figure 16.6: One possible path decomposition for a path

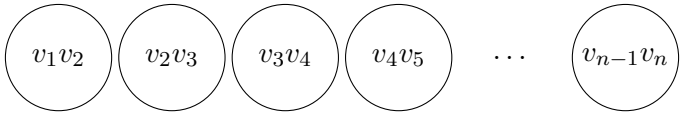


Figure 16.7: Another possible path decomposition for a path

3 Maximum Independent Set Parameterized by Pathwidth

Theorem 16.10 ([BBL13])**.** *The Maximum Independent Set problem can be solved in time* $2^{O(k)}\text{poly}(n)$*, given a pw-k path decomposition of the graph.*

We note that "given a pw-k path decomposition" is crucial, since computing a pw-k path decomposition is non-trivial. But it is known that a pw-k path decomposition can be computed in FPT time, parameterized by k. Hence, the theorem implies the statement "Maximum Independent Set is in FPT, parameterized by the pathwidth."

The algorithm uses dynamic programming, with the following insight: Consider the path decomposition $(X_1, \ldots, X_N)$. Then each X_j separates the left part and the right part of the chain. This idea is captured in the following lemma.

Lemma 16.11. $\forall j \in [N]$*, there is no edge between* $(X_1 \cup \cdots \cup X_{j-1}) \setminus X_j$ *and* $(X_{j+1} \cup \cdots \cup X_N) \setminus X_j$*.*

Proof. Suppose otherwise. Then we can find $i < j$ and $\ell > j$ and an edge (u, v) such that $u \in X_i \setminus X_j$, $v \in X_\ell \setminus X_j$. We will show that either u or v is in X_j, which is a contradiction.

By property 2 of the path decomposition, there exists a bag X' containing u, v.

Now apply property 3 on u, v. If X' is to the left of X_j, property 3 on v tells us that since $v \in X'$, $v \in X_\ell$, we must have $v \in X_j$.

If X' is to the right of X_j, property 3 on u tells us that since $u \in X_i$, $v \in X'$, we must have $u \in X_j$.

If X' is X_j, then both u, v are in X_j. We conclude that in all cases, either $u \in X_j$ or $v \in X_j$, contradicting the definition of u and v. □

We now prove Theorem 16.10 by providing the dynamic programming algorithm.

Proof. Let the subproblems be $DP(X_i, S_i)$, where X_i is a bag in the path decomposition, $S_i \subseteq X_i$, and S_i is an independent set in $G[X_i]$. $DP(X_i, S_i)$ returns the size of the maximum independent set S in $G[X_{\leq i}] = G[X_1 \cup \cdots \cup X_i]$ such that $S \cap X_i = S_i$.

For the recurrence, we have the following observation. For $DP(X_{i+1}, S_{i+1})$, let S^* be the actual maximum independent set in satisfying the conditions. We guess $S^* \cap X_i$ by considering all $\leq 2^k$ possibilities. For each possibility, we look up $DP(X_i, S^* \cap X_i)$, and we attempt to extend it to S_{i+1}.

Note that X_{i+1} cannot contain anything in $X_{\leq i} \setminus X_i$, since otherwise that element will also be in X_i be property 3. So to check if $DP(X_i, S_i)$ is compatible with the current instance $DP(X_{i+1}, S_{i+1})$,

it suffices to check that S_{i+1} and S_i agree on $X_{i+1} \cap X_i$. We obtain the following recurrence.

$$DP(X_{i+1}, S_{i+1}) = \max_{\substack{S_i \text{ s.t.} \\ S_i \cap (X_i \cap X_{i+1}) = S_{i+1} \cap (X_i \cap X_{i+1})}} (DP(X_i, S_i) + |S_{i+1} \setminus X_i|).$$

The final answer can be found by $\max_{S_N \subseteq X_N} DP(X_N, S_N)$.

For the running time, we note that we have at most $2^k \cdot N = 2^k \text{poly}(n)$ subproblems. Each subproblem looks through 2^k subproblems, and it uses at most $O(2^k)$ time. Hence, the total running time is $4^k \text{poly}(n)$. □

4 Improving the Running Time

We note that in the algorithm above, we have about 2^k subproblems, and for each subproblem, we need to search for 2^k previous subproblems. This feels like a lot of repeated work. We would like to reduce the running time to $O(2^k)$ by designing the path decomposition more cleverly.

Definition 16.12 (Nice Path Decomposition)**.** A nice path decomposition $(X_0, \ldots, X_N)$ is a path decomposition such that

- $X_0 = X_N = \varnothing$.
- Every X_i in between is either:

$$\begin{cases} \text{Introduce node:} & X_i = X_{i-1} \cup \{v\} \text{ for some vertex } v \\ \text{Forget node:} & X_i = X_{i-1} \setminus \{v\} \text{ for some vertex } v. \end{cases}$$

Observation 16.13. For a path, Figure 16.6 above is a nice path decomposition, but Figure 16.7 above is not a nice path decomposition.

We observe that using a nice path decomposition, S_i must be very similar to S_{i+1}, so each subproblem (X_{i+1}, S_{i+1}) has only a limited amount of (X_i, S_i) to search for. We obtain the following new recurrences.

- (Leaf) $DP(X_0, \varnothing) = 0$.
- (Introduction) $DP(X_{i+1}, S_{i+1})$, where $X_{i+1} = X_i \cup \{v\}$. It equals $-\infty$ if S_{i+1} is not an independent set of $G[X_{i+1}]$. Else,

$$DP(X_{i+1}, S_{i+1}) = \begin{cases} DP(X_i, S_{i+1}) & \text{if } v \notin S_{i+1} \\ DP(X_i, S_{i+1} \setminus \{v\}) & \text{if } v \in S_{i+1}. \end{cases}$$

- (Forget) $DP(X_{i+1}, S_{i+1})$, where $X_{i+1} = X_i \setminus \{v\}$. It equals $-\infty$ if S_{i+1} is not an independent set of $G[X_{i+1}]$. Else,

$$DP(X_{i+1}, S_{i+1}) = \max \begin{pmatrix} DP(X_i, S_{i+1}), \\ DP(X_i, S_{i+1} \cup \{v\}) \end{pmatrix}.$$

Now for the running time, we only need $O(1)$ time for each subproblem. Hence, the total running time is $2^k \text{poly}(n)$.

Path decomposition + dynamic programming is a general technique for solving graph problems. We note that Maximum Independent Set has a "local property": to check that a candidate solution is an independent set, it suffices to run a check on each vertex. We can replace Maximum Independent Set by Vertex Cover or Coloring, which also have this local property. Then the same idea of pathwidth with slightly different dynamic programming algorithms will give us a FPT algorithms for Vertex Cover and Coloring, parameterized by the pathwidth.

References

[BBL13] Hans L. Bodlaender, Paul Bonsma, and Daniel Lokshtanov. The fine details of fast dynamic programming over tree decompositions. In Gregory Gutin and Stefan Szeider, editors, *Parameterized and Exact Computation*, pages 41–53, Cham, 2013. Springer International Publishing. 16.10