Recognition Algorithms for Orders of Small Width and Graphs of Small Dilworth Number

Stefan Felsner[1], Vijay Raghavan[2], and Jeremy Spinrad[2]

[1] Freie Universität Berlin, Fachbereich Mathematik und Informatik,
Takustr. 9, 14195 Berlin, Germany
felsner@inf.fu-berlin.de
[2] Box 1679-B, CS Dept., Vanderbilt University,
Nashville, TN 37235, USA
{raghavan, spin}@vuse.vanderbilt.edu

Abstract. Partially ordered sets of small width and graphs of small Dilworth number have many interesting properties and have been well studied. Here we show that recognition of such orders and graphs can be done more efficiently than by using the well-known algorithms based on bipartite matching and matrix multiplication. In particular, we show that deciding deciding if an order has width k can be done in $O(kn^2)$ time and whether a graph has Dilworth number k can be done in $O(k^2n^2)$ time.

For very small k we have even better results. We show that orders of width at most 3 can be recognized in $O(n)$ time and of width at most 4 in $O(n \log n)$.

Mathematics Subject Classifications (1991). 05C75, 05C85, 06A07.
Key Words. Partial order, width, Dilworth number, recognition algorithms.

1 Introduction

The width of a partially ordered set is the size of its largest antichain. Orders of small width are a particularly attractive class since several optimization problems, known to be intractable for general orders as can be solved in polynomial time when the input is restricted to this class [3, 27, 26, 10]. It is therefore desirable to have fast recognition algorithms for orders of small width. Since the width of an n element order can be obtained using a max–flow computation on a bipartite network with unit capacities there is a $O(n^{5/2})$ recognition algorithm. For special classes of orders much faster algorithms exist. Given a realizer, the width of a 2-dimensional order can be computed in $O(n \log n)$ time [17], a fact which has been discovered many times in different contexts since it corresponds to the natural problem of finding a maximum decreasing subsequence. Using ven Emde Boas' data structure [30], the time complexity can be reduced to $O(n \log \log n)$. The width of an interval order can be determined in $O(n)$ time given an interval representation. 2-dimensional realizers and interval representations can be constructed in linear time [22, 23].

In the next section we show that orders of width at most k can be recognized in $O(kn^2)$ time. The high level idea of the algorithm is to compute the max–flow mentioned above in two phases. In the first phase a greedy chain decomposition is used to obtain a feasible flow which is at most $k \log n$ units less than the maximum. In the second phase the true max–flow is computed using augmenting paths.

The Dilworth number of a graph is the size of the largest subset of its vertices in which no vertex contains the neighborhood of another. Dilworth number 1 graphs correspond to the well-known class of threshold graphs [15, 9, 21] and graphs with Dilworth number at most 2 are the threshold-signed graphs [4]. Both these classes can be recognized in time linear in the number of edges and vertices [8, 9, 5]. Graphs with Dilworth number at most 3 are studied in [19], and shown to be a subset of a class called "good graphs" [19] or "quasitriangulated graphs" [18] defined by a special elimination ordering scheme. Graphs with Dilworth number 4 are shown to be a subset of perfectly orderable graphs in [24].

The known algorithm [19] for computing the Dilworth number of a graph works by constructing a partial order based on neighborhood containment relationships between pairs of vertices and then computing the width of the resultant order. The typical method of constructing the order from the given graph uses matrix multiplication.

In Section 3 we give a non-trivial application of the algorithm for deciding whether an order has small width. We show that graphs with Dilworth number at most k can be recognized in $O(k^2n^2)$ time. The key idea here is an algorithm which either constructs the partial order mentioned above or concludes that the input graph has Dilworth number larger than k. From the point of view of recognizing Dilworth number k graphs, this step effectively finesses away the bottleneck step of using matrix multiplication to compute the partial order.

In Section 4 we come back to orders of bounded width. We show that orders of width at most 3 can be recognized in $O(n)$ time, and orders of width at most 4 can be recognized in $O(n \log n)$ time. These results are achieved by assuming that a linear extension of the order is given and a query about the relation of two elements is answered in $O(1)$ time. More realistically, if the input is given as an adjacency matrix, we show that an initial $O(n \log n)$ preprocessing step can be used to obtain the required linear extension.

2 Orders of Bounded Width

Throughout this paper, $G = (V, E)$ denotes a simple undirected graph and P represents a partially ordered set (or *poset* or *order*). The letter n denotes either the number of vertices in a the graph under consideration or the number of elements in the poset.

We consider an order $P = (X, \leq)$ to be a pair consisting of a ground set V and a reflexive, antisymmetric and transitive binary relation $\leq$ on X. The notations $x \leq y$ in P and $y \geq x$ in P will be used interchangeably. If neither $x \leq y$ in P nor $y \leq x$ in P we say x and y are *incomparable* and write $x||y$. If $x \leq y$ and $x \neq y$, we write $x < y$ or $y > x$. An *antichain* of P is a set of pairwise incomparable elements and a *chain* of P is a set of pairwise comparable elements. The *width* of P is the size of a maximum antichain and the *height* of P is the size of a maximum chain in P.

Chains and antichains are linked by two classical duality theorems stated below in Proposition 1. Interestingly, while the formulation of these theorems is completely symmetric with respect to the exchange of "chain" and "antichain," the first is a folklore result admitting an obvious proof, while the second is a remarkable result known as Dilworth's Theorem [13].

Proposition 1 *For every finite order P.*

(1) *The maximum size of a chain in P equals the minimum number of antichains needed to cover the elements of P.*

(2) *The maximum size of an antichain in P equals the minimum number of chains needed to cover the elements of P.*

Fulkerson [16] gave a proof of Dilworth's Theorem by reducing it to the König-Egerváry duality theorem for bipartite graphs. This theorem states that *in a bipartite graph the size of a minimum vertex cover equals the size of a maximum matching.*

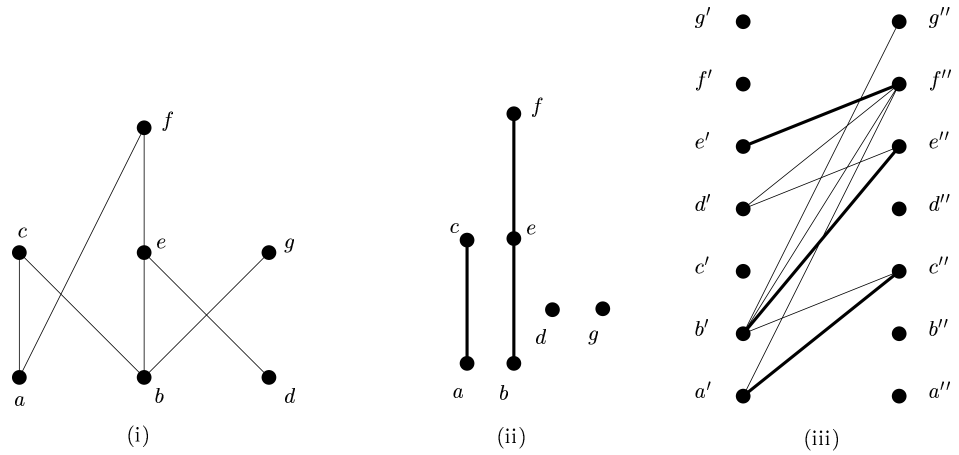

Fig. 1. (i) An order P, (ii) a chain cover for P, and (iii) the corresponding matching in $S(P)$. (The **bold** edges in $S(P)$ are the edges in the matching.)

Fulkerson's proof uses the *split* $S(P)$ of $P = (X, \leq)$, i.e., the bipartite graph having as vertices two copies X', X'' of X and an edge (x', y'') whenever $x < y$ in P. A matching M in $S(P)$ corresponds to a partition of P into $|X| - |M|$ chains. To see this, begin with the partition of P into 1–element chains. For each edge $(x', y'') \in M$ hook the tail of the chain ending with x to the beginning of the chain starting with y thus reducing the number of chains by one. (Figure 1 has an example of an order P, a chain cover for P, and the corresponding matching in $S(P)$.) With a vertex cover U of $S(P)$ associate the antichain $A_U = \{x \in X : x', x'' \notin U\}$. Using the fact that $S(P)$ comes from a transitive order relation it can be shown that $|A_U| = |X| - |U|$ when U is a minimal vertex cover. (In Figure 1 the antichain associated with the minimal vertex cover $U = \{b', d', c'', f''\}$ is the set $A_U = \{a, e, g\}$.) Dilworth's theorem now follows from the König-Egerváry duality theorem, i.e., from $\max |M| = \min |U|$.

By the above considerations the alternating path algorithm for bipartite matching allows one to compute the width of P in $O(n^3)$ time: an augmenting alternating path can be found with a depth-first search in $O(n^2)$ time and at most n augmentations along alternating path are possible. With the Hopcroft and Karp algorithm [20] (see also Tarjan's monograph [29]) this can be improved to $O(n^{5/2})$. A small improvement to $O(n^{5/2}/\sqrt{\log n})$ has been obtained in [2]. If one only wants to compute the width of the poset rather than find the corresponding antichain or chain cover, there is a randomized algorithm which has matrix multiplication rather than matching as its bottleneck step [7].

A *greedy chain decomposition* of an order $P = (X, \leq)$ is a chain cover obtained by the recursive extraction of chains of maximum length from P. More formally, we define $P_0 = P$ and recursively define chain C_i to be a maximum chain in $P_{i-1} = (X_{i-1}, \leq)$ and P_i as the order induced on the set $X_i = X_{i-1} \setminus C_i$.

Lemma 1. *Let $P = (X, \leq)$ be an n-element order of width at most k. The greedy chain decomposition of P can be computed in $O(kn^2)$ time and consists of at most $k \log_e n$ chains.*

Proof. Clearly, any antichain of an induced suborder P' of P is an antichain of P, hence, $\text{width}(P') \leq \text{width}(P)$. By Dilworth's theorem this implies that P_i can be covered by at most k chains and by the pigeon-hole principle the longest chain in P_i has size at least $|X_i|/k$. Therefore, $|X_i| \leq n(1 - \frac{1}{k})^i \leq ne^{-\frac{i}{k}}$ and for $i > k \log_e n$ the set X_i has size less than one and is thus empty. Hence the greedy chain decomposition consists of at most $k \log_e n$ chains.

A maximum chain of an order can be found by a modified depth-first search in $O(n^2)$ time; this is sometimes given in texts, for example [11], as finding a longest path in a directed acyclic graph. The removal of the chain and all comparabilities involving elements of the chain can be done within the same time bound. Using the considerations of the previous paragraph we obtain the recurrence $T(n) \leq T(n(1 - \frac{1}{k})) + c \cdot n^2$ for the time $T(n)$ used by the greedy decomposition algorithm on n-element orders of width at most k. This works out to $T(n) \leq ckn^2$. ∎

Theorem 1. *Deciding whether* $\text{width}(P) \leq k$ *for a given n-element order P and an integer k can be done in $O(kn^2)$ time.*

Proof. The first phase of the decision algorithm consists of a call to the greedy chain decomposition for P. The decomposition algorithm is enhanced by a test for

$|C_i| \geq |X_{i-1}|/k$; if this condition fails the output 'width exceeds k' is returned. By Lemma 1 the first phase takes no more than $\mathrm{O}(kn^2)$ time.

After P has successfully passed the first phase we have a chain cover $C_1, .., C_r$ with $r \leq k \log n$. If $r \leq k$, we are done by Dilworth's theorem. Otherwise, recall that $P_i = (X_i, \leq)$ is the order induced by P on the elements covered by $C_{i+1}, .., C_r$. The order P_{r-k} has width at most k as proved by the chain cover of size k. Corresponding to this chain cover is a matching M_{r-k} in the split $S(P_{r-k})$ with $|M_{r-k}| = |X_{r-k}| - k$.

The work in the second phase is now done by considering the splits $S(P_i)$ backwards, i.e., starting from the split $S(P_{r-k})$ down to the first one $S(P_0)$. Given that P_i is of width at most k we show how to check whether P_{i-1} is of width at most k in $\mathrm{O}(|X_{i-1}|^2)$ time.

To be precise we use a more complex assumption. We assume that a matching M_i in the split $S(P_i)$ of P_i with $|M_i| = |X_i| - k$ has been constructed. Consider the split $S(P_{i-1})$ of P_{i-1}. Since $S(P_i)$ is an induced subgraph of $S(P_{i-1})$ we may view M_i as a matching of $S(P_{i-1})$. A second matching in $S(P_{i-1})$ is the matching N_i corresponding to the chain $C_i = x_1, .., x_j$ which matches each x'_i with x''_{i+1} for $i = 1, .., j-1$. Since C_i and X_{i-1} are disjoint the union $M_i \cup N_{i-1}$ also is a matching in $S(P_{i-1})$; the size of the union is $|M_i \cup N_{i-1}| = |X_i| - k + |C_{i-1}| - 1 = |X_{i-1}| - (k+1)$.

Search for an augmenting path in $S(P_{i-1})$ with respect to the matching $M_i \cup N_{i-1}$. If this search is not successful we conclude from Fulkerson's proof of Dilworth's theorem that the width of P_{i-1} is $k+1$ and return an appropriate statement. Otherwise, there is an alternating augmenting path which is added to the old matching to obtain the matching M_{i-1} of size $|X_{i-1}| - k$ in $S(P_{i-1})$. As mentioned before the search for the augmenting path can be accomplished in $c' \cdot |X_{i-1}|^2$ time.

Using the bound $|X_i| \leq n(1 - \frac{1}{k})^i$ and summing up the time used in each stage of the second phase leads to the recurrence $T'(n) = T'(|X_0|) \leq T'(|X_1|) + c' \cdot n^2$ for the time $T'(|X_i|)$ used until $S(P_{i-1})$ is considered. As in Lemma 1, this works out to $T'(n) \leq c'kn^2$. Since both phases are in $\mathrm{O}(kn^2)$ the whole algorithm has the same complexity. ∎

Although the above algorithm is a decision algorithm, it can be modified to find a maximum antichain of $P = (X, \leq)$ or a minimum chain cover of P in the claimed $\mathrm{O}(kn^2)$ time, if the size of the antichain is at most k. To see this, let M be the maximum matching obtained in the algorithm; using the König Construction Technique [6], find a minimum vertex cover U of $S(P)$ in time linear in the number of edges and vertices. Then, as discussed earlier in Fulkerson's proof of Dilworth's Theorem, $A_U = \{x \in X : x', x'' \notin U\}$ is a maximum antichain of P. To find a minimum chain cover, view P as a directed graph and consider the subgraph obtained by restricting to the edges $\{(x, y) : (x', y'') \in M\}$. Again, as in Fulkerson's proof, it is easy to see that this subgraph yields the desired minimum chain cover.

Until this point, we have assumed that the input is an order, and that we only have to determine the width of the order to solve the recognition problem; that is, we have assumed that the input is transitive. The best known algorithm for verifying that a directed graph is transitive takes time proportional to matrix multiplication. However, given a set of k chains, it is easy to verify transitivity in $O(kn^2)$ time. For each vertex v, find the first vertex c_i on each chain (including the chain containing v) such that v has an edge to c_i; the input is transitive if and only if for every vertex v, $N(v)$ contains $N(c_i)$ for each c_i. Thus, checking transitivity reduces to performing kn neighborhood containment operations each of which can be done in $O(n)$ time, giving a total running time of $O(kn^2)$.

A graph G is a comparability graph if G is the underlying undirected graph of a poset. The algorithm of this section can be combined with results from [22] to recognize comparability graphs with no independent set of size k in $O(kn^2)$ time. [22] contains an algorithm which finds a transitive orientation of a comparability graph in linear time. However, that algorithm will also assign orientations to input graphs which are not comparability graphs. Thus, if we want to test whether G is a comparability graph without any independent set of size $> k$, we run the transitive orientation algorithm to get a directed graph G' and test whether G' is a width k poset. Here we cannot asume that the input is transitive, so we must make the additional test of the previous paragraph.

3 Graphs of Bounded Dilworth Number

The *open neighborhood* of a vertex v in a graph, written $N(v)$, is the set of neighbors of v in the graph, and the *closed neighborhood* of v, written $N[v]$, is the set $N(v) \cup \{v\}$. Vertex v *dominates* vertex w in G if $N(w) \subseteq N[v]$. Two vertices are *twins* if they have the same open neighborhoods. The *Dilworth number* of G is the cardinality of the largest set S of vertices in G such that no vertex in S dominates another vertex in S.

There is a simple relationship between Dilworth number of a graph and poset width. For any graph G, we create a corresponding *neighborhood containment (directed) graph* G' in which x has an edge to y if and only if x dominates y. By definition, the Dilworth number of G is the size of the maximum independent set in G'. Note that twins in G cannot be in an independent set of G'. Therefore, one may as well consider the *reduced* (undirected) graph H of G which has the set of equivalence classes of twins of G as vertices and there is an edge between equivalence classes $[x]$ and $[y]$ if and only if there is an edge between x and y in G. Observe that H', the neighborhood containment graph of H, is a partially ordered set using the relation $x \geq y$ whenever there is an edge from x to y and that the Dilworth number of G is precisely the width of H'. The fact that the neighborhood containment graph H' is a poset seems to have first been noticed in the context of Dilworth number in [19].

Using this observation, the Dilworth number of a graph can be computed in polynomial time. First, the reduced graph can be obtained in linear time given an adjacency list as input. This seems to be a folklore result; one algorithm is given in [25]. Next, the neighborhood containment of the reduced graph can clearly be constructed in $O(n^3)$ time. This can be improved to O(MM) time, where MM is the time to multiply two n by n matrices. The square of the adjacency matrix has the quantity $|N(x) \cap N(y)|$ in the (x, y) entry of the product for all distinct x and y. Hence, x dominates y in G precisely when the degree of y in G is equal to the (x, y) entry in the square of the adjacency matrix of G if x and y are nonadjacent, and when x and y are adjacent x dominates y if the (x, y) entry of the square of the adjacency matrix is equal to the degree of y - 1. The current best time bound for matrix multiplication, due to Coppersmith and Winograd [12], is $O(n^{2.376\cdots})$. For practical purposes, the matrix multiplication bounds of $O(\frac{n^3}{\log n})$ [1] and $O(n^{2.81\cdots})$ [28] given by earlier algorithms may be more appropriate. Finally, the Dilworth number can be computed by finding the width of the neighborhood containment of the reduced graph using network flow techniques or the algorithm given in the previous section.

The main result of this section is a proof that we can do better for graphs of small Dilworth number.

Theorem 2. *Deciding whether the Dilworth number of an n-vertex graph G is at most k can be done in $O(k^2 n^2)$ time.*

Proof. We start by obtaining the reduced graph G' from the input graph G. As mentioned earlier, this can be done in $O(n^2)$. From the discussion above and using the $O(kn^2)$ algorithm of the previous section, it suffices to show how to construct the neighborhood containment poset of G' in $O(k^2 n^2)$ time. Strictly speaking, it suffices to describe a procedure which will either correctly conclude that G' has Dilworth number larger than k or construct the neighborhood containment graph in the required time. Indeed, this is what we now describe.

Our algorithm uses a recursive approach for the calculation of the Dilworth number. Partition the vertices of G into two sets X and Y of nearly equal size, i.e., $|X|$ and $|Y|$ may differ by at most 1. Recursively, decide that the subgraphs induced by either X or Y has Dilworth number greater than k and stop, or conclude that both subgraphs have Dilworth number at most k and obtain both their neighborhood containment posets. In the latter case, use the algorithm of the previous section to compute minimum chain covers C_X and C_Y respectively of each poset. We now show how to use these two chain covers and the corresponding two posets to compute efficiently the neighborhood containment poset of the entire graph G.

First, we find for each vertex v of G and each chain C in $C_X \cup C_Y$, the vertex $p_{v,C}$ on C such that v is a neighbor of every vertex at or above $p_{v,C}$ on C, and is a nonneighbor of everything below. Taking advantage of the total order in each chain, for a given chain C and a given vertex v the vertex $p_{v,C}$ can be found in $O(\log |C|)$ time by binary search.

Next, we compute the relations between elements in two chains $C \in C_X$ and $C' \in C_Y$. This is done in two rounds. First, we determine relations of the form $x < y$ for $x \in C$ and $y \in C'$; in the second round the role of X and Y are exchanged to determine the relations $y < x$. Initially, define for each vertex x on C a vertex $highest(x, C, C')$ of C' as the highest vertex y such that there exists some vertex v for which $y = p_{v,C'}$ and $p_{v,C}$ is at or below x on C. Now if x' is the descendant of x in C, then $highest(x, C, C')$ is the higher of $highest(x', C, C')$ and the highest point $p_{v,C'}$ among vertices v such that $x = p_{v,C}$. Since each vertex v marks exactly one vertex as $p_{v,C}$, we can calculate $highest(x, C, C')$ for all vertices x on C in $O(n)$ time. In other words, the total time to calculate $highest(x, C, C')$ for all vertices x on C over all pairs of chains $C \in C_X$ and $C' \in C_Y$ is $O(k^2 n)$.

Finally, note that for each vertex x on $C \in C_X$, $N(x)$ is contained in the neighborhood of every vertex at or above $highest(x, C, C')$ on $C' \in C_Y$, and is not contained in the neighborhood of any vertex below. Therefore, once the *highest* functions have been computed, one can derive all the neighborhood containment relationships between pairs of vertices $x \in X$ and $y \in Y$.

To analyze the time complexity of this recursive procedure, we need to distinguish between two different variables—n, the number of vertices in the entire input graph, and r, the number of vertices in the subgraph of the current recursive call. Then the time complexity $T(r)$ of any recursive call to the decision procedure for a graph on r vertices is given by the recurrence relation:

$$T(r) = 2T(r/2) + O(kr^2) + O(nk \log r) + O(k^2 n) + O(r^2)$$

Here the term $O(kr^2)$ corresponds to the time for using the algorithm of the previous section to compute the chain covers given the posets, the term $O(nk \log r)$ corresponds to the time for finding $p_{v,C}$ for each vertex v and each chain C, the term $O(k^2 n)$ corresponds to the calculation of the $highest(x, C, C')$ function as explained above, and the term $O(r^2)$ corresponds to the final calculation of all the neighborhood containment relationships given this function.

A routine analysis of this recurrence relation leads to the solution $T(r) = O(kr^2) + O(nkr) + O(k^2 nr)$, whence for an initial value of $r = n$ we get an overall time of $O(k^2 n^2)$ for deciding if an n-vertex graph has Dilworth number k. ■

4 Very Fast Recognition of Orders of Width ≤ 4

In this section of the paper, we develop sublinear algorithms to recognize posets of width k for $k \leq 4$. Note that we must assume that we are given a true poset as input; if we must verify that every edge implied by transitivity is in the input, it takes $\Omega(n^2)$ time to recognize even posets of width 1.

The approach for recognizing orders of width at most k ($k \leq 4$), i.e., orders with no antichain of size $k+1$, presented here makes implicit use of a further important Theorem of R.P. Dilworth.

Proposition 2 (Dilworth 1958) *The antichains of maximum size of P form a distributive sublattice of the lattice of antichains of P.*

The order relation of the lattice of maximum antichains is given by $A \leq B$ for maximum antichains A, B of P iff for all $a \in A$ there is a $b \in B$ with $a \leq b$ in P. We will use the notation $\text{HMA}(P)$ to denote the *highest maximum antichain* of P. In other words, $\text{HMA}(P)$ is the unique maximal element of the lattice of maximum antichains of P.

The idea for the algorithms for recognizing orders of width 2,3 and 4 is to use a linear extension L of P and insert the elements in the order given by L into a data structure that maintains a tower of antichains of decreasing sizes. Let Q be the order induced by the first t elements of L. The invariant structure associated to $Q = Q_0$ is recursively defined by the following rule:

$$\text{Given } Q_i \text{ let } A_i = \text{HMA}(Q_i) \text{ and } Q_{i+1} = \{x \in Q_i : x > a \text{ for some } a \in A_i\}.$$

Since $\text{width}(Q_{i+1}) < \text{width}(Q_i)$ the order Q_k is empty, when k is the width of Q. While inserting the $t+1$st element we have to update the invariant structure. This is done by examining a list of candidate sets for the new highest antichains. For width two this is quite easy. For width 3 and 4 it gets increasingly involved and we will have to introduce some additional bookkeeping. It is quite possible that the approach can be generalized to higher values of $k = \text{width}(P)$. However, the case $k = 5$ already seems to require an unpleasantly involved case analysis.

Theorem 3. *Given an order P with n elements and a linear extension L of P, there is a algorithm that decides $\text{width}(P) \leq k$ for $k = 2, 3$ in $O(n)$ time. For $k = 4$ there is an algorithm for the decision problem running in $O(n \log n)$ time.*

In the next three subsections we describe the algorithms proving the theorem. Notationally it will be convenient to work with a vector $(B_1, B_2, .., B_k)$ of antichains with $B_j = \emptyset$ or $|B_j| = j$ such that the non-empty antichains in this vector are exactly the antichains $A_i = \text{HMA}(Q_i)$ defined above.

The assumption that a linear extension L of P is given as part of the input can be removed if we are willing to spend $O(n \log n)$ time to either find such an extension or show that P has width $> k$, as described below.

Construct a complete binary tree T with root r, which has elements of P as leaves. Maintain at each internal node i the set $S(i)$ of minimal elements of the subposet induced by the elements assigned to the leaves below i; note that $|S(i)| \leq k$ or P has width $> k$. We compute the initial values of $S(i)$ during a postorder traversal of T; this can be done using k^2 comparisons at each internal node, and thus takes $O(k^2 n)$ time. We then choose any element x from $S(r)$ to output next in the linear extension L. Delete x from our tree; this involves repeating comparisons only along the path from x to r in T. Since each element can be deleted in $O(k^2 \log n)$ time, L is found in $O(k^2 n \log n)$ time for a width k poset.

Note that if we are not given a linear extension as part of the input, instead being given access to an adjacency matrix, finding a chain decomposition requires $\Omega(n \log n)$ time, since if the input is a totally ordered set we must produce a sorted sequence using only comparisons between elements.

4.1 Width Two

Let $L = x_1, x_2, \ldots, x_n$ be the linear extension of P which is assumed to be given as part of the input. Let $x_j || x_{j+1}$ be the first incomparable adjacent pair in L. The algorithm is initialized with $B_2 \leftarrow \{x_j, x_{j+1}\}$, $B_1 \leftarrow \emptyset$.

When inserting a new element $x = x_t$ we first compare x to the elements of B_2. If x is incomparable to both we have detected a three element antichain and stop. The remainder of the algorithm is broken up into two cases.

In the first case we assume $B_1 = \{a\} \neq \emptyset$. If $x > a$ then $B_1 \leftarrow \{x\}$ and we are done, otherwise, if $x||a$ we set $B_2 \leftarrow \{a, x\}$ and $B_1 \leftarrow \emptyset$.

The more complex case is when $B_1 = \emptyset$. Let $B_2 = \{b, c\}$ here we have three cases.

(1) If $x > b$ and $x > c$ then $B_1 \leftarrow \{x\}$.

(2) If $x > b$ and $x||c$ then $B_2 \leftarrow \{x, c\}$.

(3) If $x||b$ and $x > c$ then $B_2 \leftarrow \{b, x\}$.

Since, it is easy to extend the algorithm such that a chain covering by two chains is maintained the correctness of the algorithm is obtained by Dilworth's theorem (Proposition 1(2)). We need at most three comparisons to insert a new element, hence, the complexity of the algorithm is $O(n)$.

4.2 Width Three

The initialization of the algorithm for width three is as in the width two case with the addition that $B_3 \leftarrow \emptyset$ has to be initialized. When the new element x is comparable to some element of B_1 or B_2 we reuse the width two algorithm and only have to modify B_1 and/or B_2.

For the remaining discussion assume that x is incomparable to all elements of $B_1 \cup B_2$. We describe the algorithm as a list of cases with the assumption that the algorithm picks the first applicable case.

Case 0. $B_3 \neq \emptyset$ and x is incomparable to all elements of B_3: We have detected a 4-antichain and stop.

Case 1. $B_2 \neq \emptyset$: The update is done by the two assignments $B_3 \leftarrow B_2 + x$ and $B_2 \leftarrow \emptyset$.

For the remainder of the width 3 recognition algorithm, $B_3 = \{y_1, y_2, y_3\}$ and $B_2 = \emptyset$.

Case 2. $B_1 = \emptyset$: Compare x to the elements of B_3. If x is comparable to exactly one element $y \in B_3$ then $B_3 \leftarrow B_3 - y + x$, otherwise, leave B_3 as it is and let $B_1 \leftarrow \{x\}$.

Now suppose that $B_1 = \{a\}$. The set of elements above B_3 is a chain $C = (z_1 < z_2 < .. < z_l)$ with $z_l = a$. Since B_3 is the highest 3-antichain in the order induced by the elements inserted before x the element z_1 is greater then at least two elements of B_3. We assume that $y_1 < z_1$ and $y_2 < z_1$.

Case 3. In this case we consider 3-antichains $\{x, z_i, y_3\}$. The candidate for z_i is the highest element in C which is incomparable to y_3. If $z_1 > y_3$ we proceed with case 4. Otherwise the candidate z_i can be found by comparing the elements of chain C one by one to y_3. This may cause a non-constant cost for the insertion of x, however, all but the last of the elements will be removed from chain C. We will charge the cost of these comparisons to the elements of the chain. When z_i is found we have two subcases.

Subcase 3a. $x||z_i$: Let $B_3 \leftarrow \{x, z_i, y_3\}$ and if $i = l$ also $B_1 \leftarrow \emptyset$.

Subcase 3b. $x > z_i$: Since this implies that x is greater than two elements of B_3 we know that x is not contained in any 3-antichain and we let $B_2 \leftarrow \{x, a\}$ and $B_1 \leftarrow \emptyset$.

Case 4. Element x is greater then exactly one element of B_3: In this case $y_3 < x$ or z_1 is greater than all elements of B_3 and we may rename them so that $x > y_3$. Assign $B_3 \leftarrow \{y_1, y_2, x\}$.

Case 5. In this last case x is not contained in any 3-antichain and we safely let $B_2 \leftarrow \{x, a\}$ and $B_1 \leftarrow \emptyset$.

Taking into account the amortization of cost in Case 3, the running time of the algorithm is obviously in $O(n)$. As consequence of the algorithms for recognition of orders of width 2 and 3 we obtain alternative algorithms for recognizing graphs of Dilworth number 2 and 3 which are as fast as the algorithm given in Section 3. Since our $O(n)$ algorithms only make a linear number of comparisons between elements and an domination-comparison takes $O(n)$ we obtain a time complexity of $O(n^2)$. It only remains to make sure that we can generate a linear extension of the domination order of a graph in $O(n^2)$ time. Here we can avoid an n^2 bottleneck by simply using the ordering of the vertices by degree.

Theorem 4. *Graphs of Dilworth number at most 3 can be recognized in* $O(n^2)$ *time.*

4.3 Width Four

The initialization of the algorithm for width four differs from the width three case only in that $B_4 \leftarrow \emptyset$ has to be initialized. When the new element x is comparable to some element of B_1, B_2 or B_3 we reuse the width three algorithm and only have to modify a subset of B_1, B_2, B_3. However, as an additional feature we have to maintain a chain cover of the width two order on the top. We assume that this cover C_1, C_2 has the property that every element in C_2 is incomparable to some element in C_1; note that such a cover can be produced by the recognition algorithm for width two.

For the remaining discussion assume that x is incomparable to all elements of $B_1 \cup B_2 \cup B_3$. Again, we describe the algorithm as a list of cases such that the algorithm picks the first applicable case.

Case 0. $B_4 \neq \emptyset$ and x is incomparable to all elements of B_4: We have detected a 5-antichain and stop.

Case 1. $B_3 \neq \emptyset$: Set $B_4 \leftarrow B_3 + x$ and $B_3 \leftarrow \emptyset$.

Henceforth $B_4 = \{y_1, y_2, y_3, y_4\}$ and $B_3 = \emptyset$.

Case 2. $B_2 = \emptyset$ and $B_1 = \emptyset$: Compare x to the elements of B_4. If x is comparable to exactly one element $y \in B_4$ then $B_4 \leftarrow B_4 - y + x$, otherwise, leave B_4 as it is and let $B_1 \leftarrow \{x\}$.

Case 3. $B_2 = \emptyset$ and $B_1 = \{a\}$: The set of elements above B_4 is a chain $C = (z_1 < z_2 < .. < z_l)$ with $z_l = a$. Since B_4 is the highest 4-antichain in the order induced by the elements inserted before x the element z_1 is greater then at least two elements of B_4. We assume that $y_1 < z_1$ and $y_2 < z_1$.

Subcase 3a. In this case we consider 4-antichains $\{x, z_i, y_3, y_4\}$. The candidate for z_i is the highest element in C which is incomparable to y_3 and y_4. If z_1 is greater than one of y_3 and y_4 we can skip to the next case. Otherwise, let $h(y_j)$ denote the highest element of C incomparable to y_j for $j = 3, 4$. Let z be the smaller of $h(y_3)$ and $h(y_4)$. If $z||x$ then $B_4 \leftarrow \{x, z, y_3, y_4\}$ and if $i = l$ also $B_1 \leftarrow \emptyset$.

Subcase 3b. Element x is greater then exactly one element y of B_4: Assign $B_4 \leftarrow B_4 + x - y$.

Subcase 3c. In this last case x is not contained in any 4-antichain and we safely let $B_2 \leftarrow \{x, a\}$ and $B_1 \leftarrow \emptyset$.

Case 4. $B_2 = \{a, b\}$:

Let C_1, C_2 be a chain partition of the set of elements above B_4. Recall that we assume that every element in C_2 is incomparable to some element in C_1. For $y \in B_4$ let $h_i(y)$ for $i = 1, 2$ be the highest elements of C_i incomparable to y. If y is below all elements of C_i we define $h_i(y) = y$. Since B_4 is the highest maximum antichain the least element of each of C_1, C_2 is already comparable to at least two of the elements of B_4. We may assume that every element of C_1 is greater than y_1 and y_2 and every element of C_2 is greater than y_3.

Subcase 4a. There is a 4-antichain consisting of x, elements $c_i \in C_i$ for $i = 1, 2$ and y_4. Such a 4-antichain can only exist if y_4 is incomparable to x and $h_1(y_4) \neq y_4$ and $h_2(y_4) \neq y_4$. If x is bigger then either of $h_1(y_4)$ and $h_2(y_4)$ then there is no antichain of this type. Assume that x is incomparable to both.

If $h_1(y_4)||h_2(y_4)$ then $B_4 \leftarrow \{x, h_1(y_4), h_2(y_4), y_4\}$.

If $h_2(y_4) < h_1(y_4)$ let z be the highest element of C_1 incomparable to $h_2(y_4)$. If $z < x$ then there is no antichain of this type. If $z||x$ let $B_4 \leftarrow \{x, z, h_2(y_4), y_4\}$.

If $h_1(y_4) < h_2(y_4)$ the situation is more complex. If there are elements in C_2 incomparable to $h_1(y_4)$ let z be the highest such element. If $z < x$ then there is no antichain of this type. If $z||x$ let $B_4 \leftarrow \{x, h_1(y_4), z, y_4\}$.

If $h_1(y_4) < h_2(y_4)$ and every element of C_2 is comparable to $h_1(y_4)$ let z_2 be the highest element of C_2 below $h_1(y_4)$ and let z_1 be the highest element of C_1 incomparable to z_2. If z_1 and z_2 are both incomparable to x let $B_4 \leftarrow \{x, z_1, z_2, y_4\}$.

Subcase 4b. There is a 4-antichain consisting of x, an element $c \in C_1 \cup C_2$ and two elements from B_4. This requires that x is incomparable to both of y_3 and y_4. If $c \in C_1$ then $h_1(y_3) \neq y_3$ and $h_1(y_4) \neq y_4$. Let $c \leftarrow \min\{h_1(y_3), h_1(y_4)\}$, if $x||c$ then $B_4 \leftarrow \{x, c, y_3, y_4\}$. The case where $c \in C_2$ is identical up to changes in indices.

In the last two subcases we have ignored the update of B_2 and B_1. This update is easily done by considering the possible relations of the elements in the new B_4 and a, b.

Subcase 4c. Element x is greater then exactly one element of B_4: Assign $B_4 \leftarrow B_4 - y + x$.

Subcase 4d. In this last case x is not contained in any 4-antichain and we safely let $B_3 \leftarrow \{x, a, b\}$ and $B_2 \leftarrow \emptyset$.

The algorithm requires a procedure that given a chain C and an element y returns the highest element of C incomparable to y. With binary search this can be accomplished in $O(\log |C|)$ time. During insertion of an element this procedure is only called a constant (<10) number of times. Beside these calls the insertion is in $O(1)$ per element. This gives an overall complexity of $O(n \log n)$.

5 Concluding Remarks

We have shown that it is possible to avoid the bottlenecks of matching and matrix multiplication in recognizing small width partial orders and graphs with small Dilworth number. We have two different techniques (and time complexities) for recognizing small width partial orders—one technique can be used only for partial orders of width at most 4 and the other can be used for larger values of width. It would be interesting to see if these two techniques can somehow be combined to yield a faster algorithm.

References

1. V.L. Arlazarov, E.A. Dinic, M.A. Kronrod and I.A. Faradzev. On economical construction of the transitive closure of a directed graph. *Soviet Math Doklady*, **11**, 1209–1210, 1970.
2. H. Alt, N. Blum, K. Mehlhorn and M. Paul. Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}(m/\log n)^{0.5})$. *Inf. Proc. Letters*, **37**, 237–240, 1991.
3. M.D. Atkinson, H.W. Chang. Linear extensions of posets of bounded width. *Congressus Numerantium*, **52**, 21–35, 1986.
4. C. Benzaken, P.L. Hammer, and D. de Werra. Threshold characterization of graphs with Dilworth number 2. *Journal of Graph Theory*, **9**), 245–267, 1985.
5. C. Benzaken, P.L. Hammer, and D. de Werra. Split graphs of Dilworth number two. *Discrete Math* **55**), 123–128, 1985.
6. J.A. Bondy, U.S.R. Murty. Graph theory with applications. Elsevier, New York/Macmillan, London, 1976.
7. J. Cheriyan. Randomized $O(M(|V|))$ algorithms for problems in matching theory. *SIAM Journal on Computing* **26**, 1635–1655, 1997.
8. V. Chvátal and P.L. Hammer. Set-packing and threshold graphs. *University of Waterloo Research Report CORR73-21*, 1973.
9. V. Chvátal and P.L. Hammer. Aggregation of inequalities in integer programming. *Annals of Discrete Math* **1**, 145–162, 1977.
10. C.J. Colbourn, W.R. Pulleyblank. Minimizing setups in ordered sets of fixed width. *Order* **1**, 225–228, 1985.
11. T.H. Cormen, C.E. Leiserson and R.L. Rivest. Introduction to algorithms. MIT Press, 1990.
12. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *J. of Symb. Comput*, **9**, 251–280, 1990.
13. R.P. Dilworth. A decomposition theorem for partially ordered sets. *Ann. Math.*, **51**, 161–166, 1950.
14. R.P. Dilworth. Some combinatorial problems on partially ordered sets. In *Combinatorial Analysis, Proc. Sympos. appl. Math. 10*, Bellman and Hall eds., 85–90, 1960.
15. S. Földes and P.L. Hammer. The Dilworth number of a graph. *Annals of Discrete Math* **2**, 211–219, 1978.
16. D.R. Fulkerson. Note on Dilworth's embedding theorem for partially ordered sets. *Proc. Amer. Math. Soc.*, **7**, 701–702, 1956.
17. M.C. Golumbic. Algorithmic graph theory and perfect graphs. Academic Press, New York, 1980.
18. I.M. Gorgos. Characterization of quasitriangulated graphs. *Technical Report University of Kishinev*, 1982.
19. C.T. Hoàng and N.V.R. Mahadev. A note on perfect orders. *Discrete Math*, **74**, 77–84, 1989.
20. J.E. Hopcroft and R.M. Karp. A $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, **2**, 225–231, 1973.
21. N.V.R. Mahadev and U.N. Peled. Threshold graphs and related topics. *Annals of Discrete Math*, **56**, 1995.
22. R. McConnell and J. Spinrad. Linear time transitive orientation. *ACM-SIAM Symposium on Discrete Algorithms*, 19–25, 1997.
23. C.H. Pappadimitriou and M. Yannakakis. Scheduling interval ordered tasks. *SIAM Journal on Computing*, **8**, 405–409, 1979.
24. C. Payan. Perfectness and Dilworth number. *Discrete Mathematics*, **44**, 229–230, 1983.
25. V. Raghavan and A. Tripathi. Improved diagnosability algorithms. *IEEE Trans. on Comput.*, **2**, 143–153, 1991.
26. J.B. Sidney and G. Steiner. Optimal sequencing by modular decomposition: polynomial algorithms. *Operations Research*, **34**, 606–612, 1986.
27. G. Steiner. Polynomial algorithms to count linear extensions in certain posets. *Congressus Numerantium*, **75**, 71–90, 1990.
28. V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, **13**, 354–356, 1969.
29. R.E. Tarjan. Data structures and network algorithms. CBMS Series 44, SIAM, Philadelphia, 1983.
30. P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, **6**, 80–82, 1977.