Chapter 9

Lecture 9

Advanced Topics in Graph Algorithms

Spring Semester, 1994

Lecturer : Ron Shamir
Scribe : Muafaq Tannous
Lecture 9 : June 9

9.1 Comparability Graph Recognition

In lecture 6 we discussed characterization of comparability graphs, and we proved a theorem - TRO theorem - that characterizes comparability graphs. We described an algorithm that constructs G-decomposition. By combining the TRO theorem with the G-decomposition construction we obtain an algorithm that recognizes comparability graphs and generates a transitive orientation the graph if it is TRO. The algorithm is shown in figure 9.1

0. initialize $i \longleftarrow 1, E_i \longleftarrow E, F \longleftarrow \emptyset$
1. arbitrarily pick an edge $x_i y_i \in E_i$
2. enumerate the implication class B_i of E_i containing $x_i y_i$
   if $B_i \cap B_i^{-1} = 0$ then add B_i to F
   else stop (G is not TRO)
3. $E_{i+1} \longleftarrow E_i - \hat{B}_i$
4. if $E_{i+1} = 0$ then output F and stop
   else $i \longleftarrow i + 1$ go to 1

Figure 9.1: Code of The TRO Algorithm

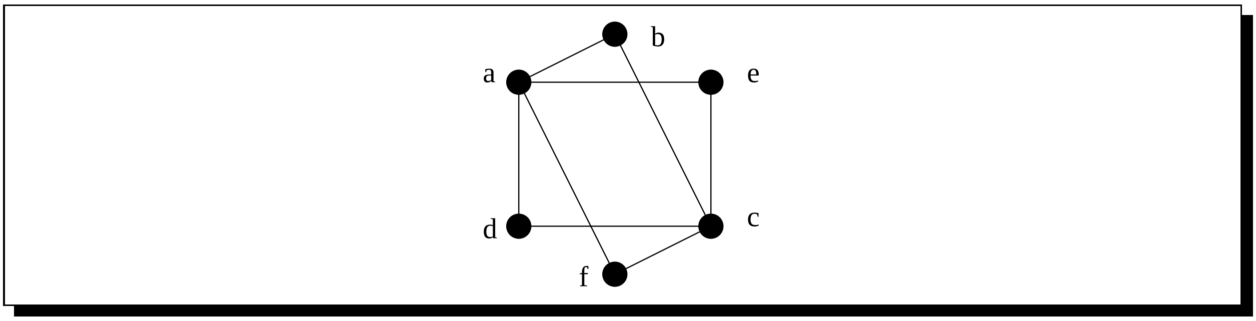

Figure 9.2: Example 1

Example: Let us run the algorithm on the graphs shown in figures 9.2, 9.3 and 9.4. The run of the algorithm on the first graph will be as the following :

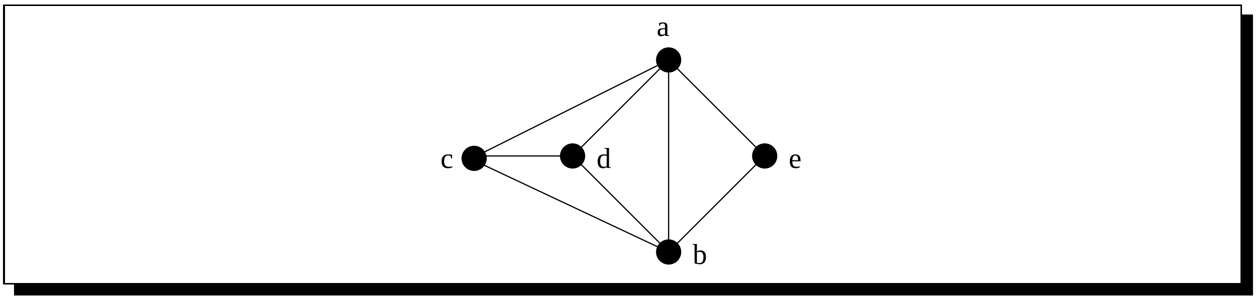

Figure 9.3: Example 2

Initialize : $E_1 = E = \{ab, ba, ae, ea, af, fa, ad, da, bc, cb, ce, ec, cf, fc, ea, ae\}, F = \emptyset$. After the first iteration we have $x_1y_2 = ab, B_1 = \{ab, ad, af, ae, cb, cf, ce, cd\}, F = B_1, E_2 = \emptyset$, so the graph is TRO.

The execution of the algorithm on the second graph will be the following :

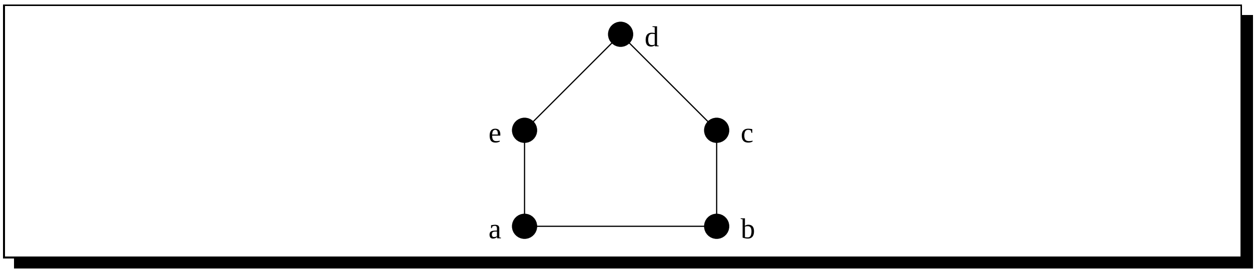

Figure 9.4: Example 3

Initialize :

$$E_1 = \{ac, ca, ae, ea, ad, da, ab, ba, cb, bc, eb, be, cd, dc, db, bd\}, F = \emptyset.$$

After the first iteration we have:

$$x_1y_1 = ac, B_1 = \{ac, ab, ad, ae\}, F = \{ac, ab, ad, ae\}, E_2 = \{dc, cd, ba, ab, be, eb, de, ed\}.$$

After the second iteration

$$x_2y_2 = bc, B_2 = \{bc, bd, ba, be\}, E_2 = \{cd\}, F = \{ac, ab, ad, ae, bc, bd, ba, be\}, E_2 = \{cd\}.$$

After the third iteration

$$x_3y_3 = dc, B_3 = \{dc\}, F = \{ac, ab, ad, ae, bc, bd, ba, be, dc\}, E_2 = \emptyset.$$

Thus the graph is TRO.

The execution of the algorithm on the second graph will be the following :

Initialize : $E_1 = \{ab, ba, bc, cb, cd, dc, de, ed, ea, ae\}, F = \emptyset$. After the first iteration we have $x_1y_1 = ab, B_1 = \{ab, ae, cb, cd, ed, ea, ba, bc, dc, de\}, B_1 \cap B_1^{-1} \neq \emptyset$, so the graph is not TRO.

The edges we choose in step (1) in the algorithm determine which transitive orientation of the set of transitive orientations of the graph will be the output of the algorithm. If we run the algorithm several times, and in every time we choose different edges in step (1) we may get different transitive orientations, but one can prove that the number of iterations of the algorithm in every run will be the same.

9.2 The Complexity of Comparability Graph Recognition

9.2.1 Implementation

We shall show that the complexity of the above algorithm is $O(\delta|E|)$ time and $O(|E| + |V|)$ space, where δ is the maximum degree of a vertex, and we'll give the implementation of this algorithm that gives such complexity. Let $G(V, E)$ be an undirected graph with $V = \{v_1, v_2, \ldots, v_n\}$. The algorithm uses a function called $CLASS(i, j)$, which is defined as the following : $CLASS(i, j) = 0$, if $v_iv_j \notin E$; $CLASS(i, j) = k$, if v_iv_j has been assigned to B_k; $CLASS(i, j) = -k$, if v_jv_i has been assigned to B_k; or undefined, if v_iv_j has not been assigned yet.

Let d_i be the out-degree of v_i after directing the graph G.

The algorithm gets as an input graph $G(V, E)$ such that $V = \{v_1, v_2, \ldots, v_n\}$ and its adjacency sets such that $j \in Adj(i)$ iff $v_iv_j \in E$. The output of the algorithm :

1. a variable $FLAG$ which tells if G is a comparability graph. $FLAG = 0$ iff G is comparability graph.

2. if $FLAG = 0$ then the transitive orientation that was found by the algorithm is the union of all edges having positive $CLASS$.

The algorithm consists of two subroutines, the EXPLORE procedure which is shown in figure 9.5, and the main procedure which is shown in figure 9.6.

The set of edges explored in iteration k is denoted B_k. In iteration k, the algorithm chooses an edge e that has not been assigned to any $B_i, i < k$, and start exploring from e. If $B_k \cap B_k^{-1} \neq \emptyset$ then the value of the variable $FLAG$ will be changed from 0 to 1.

9.2.2 Complexity Analysis

The input graph G is represented by adjacency lists. The adjacency sets are stored as linked lists sorted in increasing order and every element of the list represents a vertex. Each element in the linked list $Adj(i)$ is described by four fields. For every edge $v_i v_j$ in G there are two elements as shown in figure 9.7.

1. j
2. $CLASS(i,j)$
3. pointer to $CLASS(j,i)$
4. pointer to next element on $Adj(i)$

This data structure requires $O(|V| + |E|)$ space. Let us compute the time complexity of the algorithm: In executing the 'for' command in $EXPLORE(i,j)$ we use two pointers: one points to $Adj(i)$ and the other points to $Adj(j)$. These two pointers will scan these two sets in parallel looking for a vertex m which satisfies the condition of the 'for' command and then the value of $CLASS$ will be updated. The loop will take $O(d_i + d_j)$. The complexity of the second loop is similar because it is executed in the same way. The algorithm calls the procedure $EXPLORE$ twice for every edge, hence the time complexity of the algorithm is

$$C \sum_{v_i v_j \in E} (d_i + d_j) \in O(\delta |E|)$$

In summary :

Theorem 9.1 *Comparability graph recognition and finding a transitive orientation can be done in $O(\delta|E|)$ time and $O(|V| + |E|)$ space, where δ is the maximum degree of a vertex.*

```
procedure EXPLORE(i,j):
for each m ∈ Adj(i) such that m ∉ Adj(j) or |CLASS(j,m)| < k do
      begin
         if CLASS(i,m) is undefined then
            begin
               CLASS(i,m) ⟵ k; CLASS(m,i) ⟵ −k;
               EXPLORE(i,m);
            end
         else
            if CLASS(i,m) = −k then
               begin
                  CLASS(i,m) ⟵ k; FLAG ⟵ 1;
                  EXPLORE(i,m);
               end
      end
for each m ∈ Adj(j) such that m ∉ Adj(i) or |CLASS(i,m)| < k do
      begin
         if CLASS(m,j) is undefined then
            begin
               CLASS(m,j) ⟵ k; CLASS(j,m) ⟵ −k;
               EXPLORE(m,j);
            end
         else
            if CLASS(m,j) = −k then
               begin
                  CLASS(m,j) ⟵ k; FLAG ⟵ 1;
                  EXPLORE(m,j);
               end
      end
return
```

Figure 9.5: Procedure EXPLORE

begin
    initialize : $k \longleftarrow 0; FLAG \longleftarrow 0;$
    for each edge $v_i v_j$ in E do
        if $CLASS(i, j)$ is undefined then
            begin
                $k \longleftarrow k + 1;$
                $CLASS(i, j) \longleftarrow k; CLASS(j, i) \longleftarrow -k;$
                EXPLORE(i, j);
            end
end

Figure 9.6: The main procedure of the algorithm

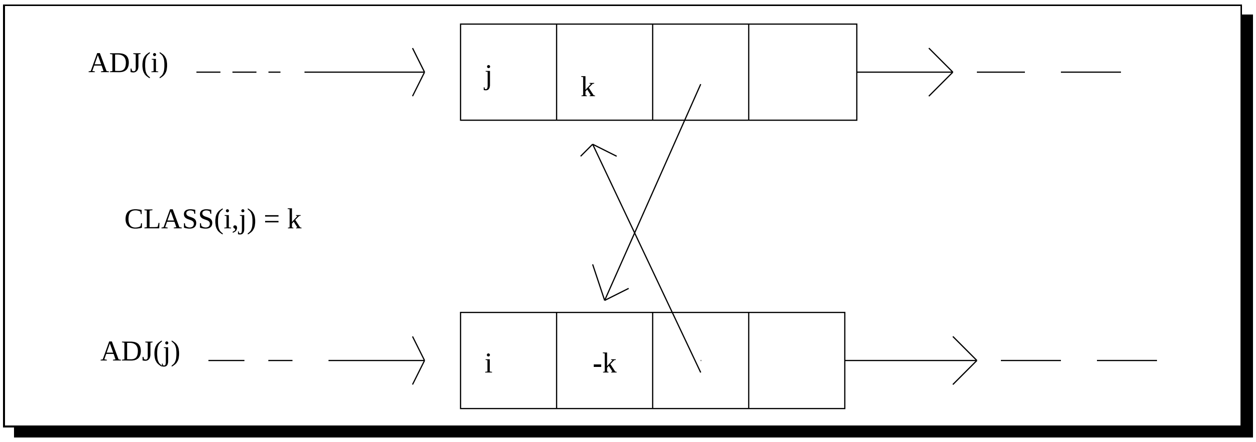

Figure 9.7: Two elements for each Edge-

As we define the procedure $EXPLORE$ it explores the edges in a depth-first search, we can change the algorithm to breadth-first search by storing the edges in queue (instead of stack), then it will run in the same time and space complexity.

There are several different algorithms that solve the TRO recognition problem. Until recently, the time complexity of the best known algorithm was due to Spinrad and has complexity $O(n^{2.5})$, In 1994 McConnel and Spinrad found an algorithm that in $O(m+n\log n)$ constructs an orientation F in the graph G such that if G is a comparability graph then F is transitive. There is an algorithm that decide if F is transitive in $O(n^{2.371\cdots})$.

9.3 Coloring and Other Problems on Comparability Graphs

9.3.1 Comparability Graphs Are Perfect

On a directed and acyclic graph $G(V,F)$, define a strict partial ordering of vertices, namely, $x > y$ iff there is nontrivial path in F from x to y. A height function h can be defined on such graphs : $h(v) = 0$ if out-degree of v is 0, $h(v) = 1 + max\{h(w)|vw \in F\}$. An example of $h(v)$ is shown in figure 9.8. Here in subsequent figures we use *Hasse diagram* of the partially ordered set, i.e. all edges are pointing upwards and transitive edges are omitted.

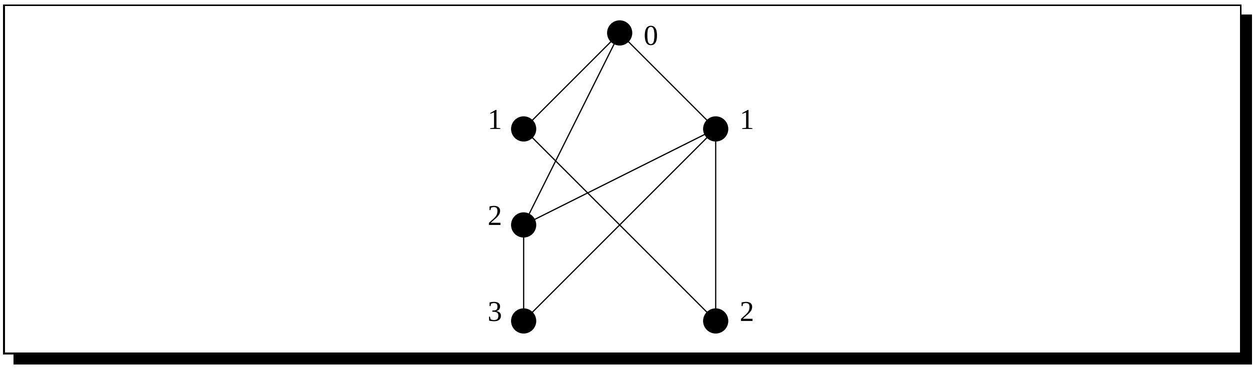

Figure 9.8: Example 4

$h(v)$ is equal to the number of vertices in the longest path from one of G roots to v. $h(v)$ can be computed in linear time using BFS. The function h gives a proper vertex coloring of G, because for every v and u if $vu \in F$ then $h(v) < h(u)$, but it is not necessarily a minimum coloring. For a graph G which is undirected, for any acyclic orientation F, the corresponding h will be a proper vertex coloring of G. We now prove that if G is transitive then h is an optimal coloring.

Theorem 9.2 *Every comparability graph is a perfect graph.*

Proof: Let F be a transitive orientation of a comparability graph $G(V, E)$. Every path in F corresponds to a clique of G, because of transitivity, and every clique corresponds to a path, hence the function h is a proper vertex coloring of G that uses exactly $\omega(G)$ colors, this is an optimal vertex coloring because $\chi \geq \omega$. Because of the hereditary property of comparability graphs we find that $\omega(G_A) = \chi(G_A)$ for all induced subgraph G_A of G, so G is a perfect graph. ∎

Let $(X, \leq)$ be a partially ordered set. A subset of X that is completely ordered is called *chain.* A subset in X that doesn't contain two members comparable by $\geq$ is called *antichain.*

The sets $\{c, e, a\}, \{d, g, b\}, \{f, b\}$ are chains in the partial order shown in figure 9.9, the sets $\{c, d\}, \{e, f, g\}$ are antichains in the same partial order.

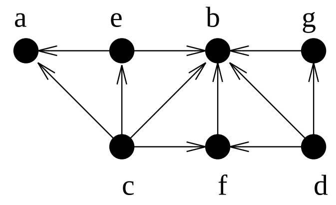

Figure 9.9: Example 5

A *Hasse diagram* is a compact representation of a transitive order. For a partially ordered set (V, F), the Hasse diagram is (V, H) where $H \subseteq F$ and $uv \in H$ iff $uv \in F$ and there is no $m \in V$ s.t. $um \in V, mv \in V$. The edges of Hasse diagram is drawn without the direction arrow, with the convention that all edges are pointing up. For example the Hasse diagram of the graph in figure 9.9 is in figure 9.10.

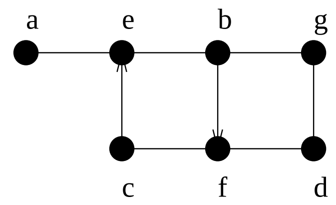

Figure 9.10: Example 6

Theorem 9.3 *(Dilworth 1950) Let $(X, \leq)$ be a partially ordered set, the minimum number of chains needed to partition (cover) X is equal to the maximum cardinality of a antichain in X.*

Proof: $(X, \leq)$ corresponds to comparability graph G which is a perfect graph, hence $k(G) = \alpha(G)$. k is equal to the minimum number of chains needed to partition X, because a vertex set S induces a clique in G iff there is a path in X passing through all vertices in S. α is equal to the maximum cardinality of an antichain. So, the minimum number of chains needed to partition X is equal to the maximum cardinality of antichain. ■

For example consider the partial order that is shown in figure 9.9. The number of chains which partition the partial order is 3, also, the maximum cardinality of antichains is 3.

Corollary 9.4 *In a comparability graph with transitive orientation F, optimal proper coloring and maximum clique can be calculated in $O(|E| + |V|)$ time.*

We shall illustrate this by solving a slightly more general problem.

9.3.2 Maximum Weighted Clique

For a given undirected graph G and an assignment of a weights $w(v)$ to each vertex, the problem is to find a clique in G for which the sum of the weights of its vertices is largest. The algorithm calculates for every vertex v the value $W(v)$ which is the sum of weights of every vertex in the heaviest path from v to one of the graph sinks. A pointer is assigned to v designating its successor on that heaviest path.

Input : a transitive orientation F of a comparability graph $G(V, E)$ and a weight function w defined on V.

Output : a clique K of G whose weight is maximum.

The algorithm uses two subroutine, the EXPLORE procedure which is shown in figure 9.11, and the main procedure which is shown in figure 9.12.

Proving the correctness of the algorithm and displaying an implementation whose complexity is linear are left as exercises.

After running the algorithm on the graph that its Hasse diagram is shown in figure 9.13 we will have $W(a) = 5, W(b) = 6, W(c) = 12, W(d) = 2, W(e) = 11, W(f) = 4, W(g) = 14, POINTER(a) = null, POINTER(b) = a, POINTER(C) = b, POINTER(d) = null$, $POINTER(e) = f, POINTER(F) = null, POINTER(g) = e$ and the maximum weight clique is f, e, g.

9.3.3 Calculating $\alpha(G)$

We shall show a polynomial-time method for finding $\alpha(G)$ for a comparability graph G. The method gets as an input a graph $G(V, E)$ and a transitive orientation F. We transform F into a transportation network by adding two new vertices s and t and edges sx and yt for each source x and sink y. Then we assign a lower bound 1 on the flow through each vertex. We compute a minimum-flow, which is a polynomial problem. The value of such flow will

```
procedure EXPLORE(v):
    if Adj(v) = ∅ then
        W(v) = w(v);
        POINTER(v) ⟵ ∧;
        return;
    else for all x ∈ Adj(v) do
        if x is unexplored then
            EXPLORE(x);
    end for all;
    select y ∈ Adj(v) such that W(y) = max{W(x)|x ∈ Adj(v)};
    W(v) ⟵ w(v) + W(y);
    POINTER(v) ⟵ y;
    return;
end.
```

Figure 9.11: Procedure EXPLORE

```
procedure MAXWEIGHT CLIQUE(V, F):
    for all v ∈ V do
        if v is unexplored then
            EXPLORE(v);
    end for all;
    select y ∈ V such that W(y) = max{W(v)|v ∈ V};
    K ⟵ y;
    y ⟵ POINTER(y);
    while y ≠ ∧ do
        K ⟵ K ∪ {y};
        y ⟵ POINTER(y);
    return;
end.
```

Figure 9.12: The main procedure of the algorithm

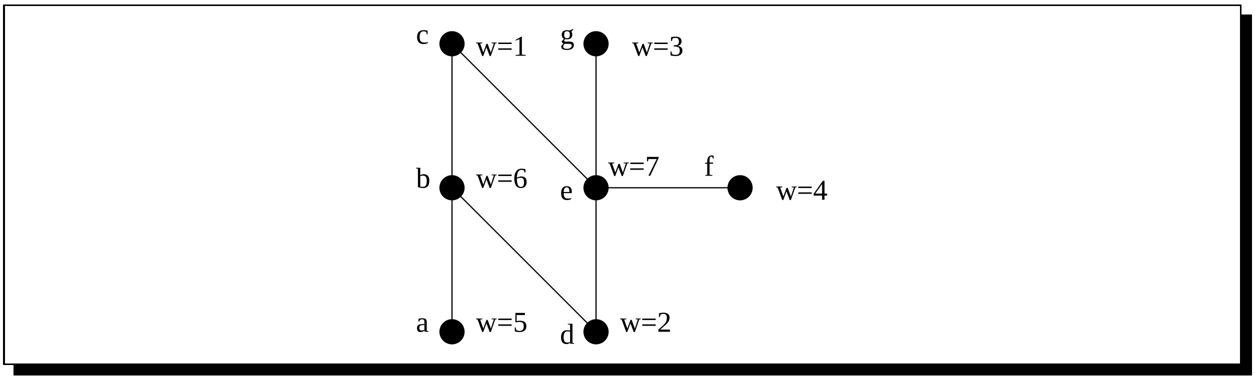

Figure 9.13: Example 7

equal to $k(G)$, the size of the smallest covering of the vertices by cliques. This value is equal to $\alpha(G)$, the maximum cardinality independent set since every comparability graph is perfect.

9.4 The Dimension of Partial Orders

Every partial order (X, P) can be extended to a linear order (also called its topological order). Such an order is called a *linear extension* of P. Let $L(P)$ be the set of linear extensions of P. A subset $L \subseteq L(P)$ that satisfies $\bigcap_{l \in L} l = P$ is called a *realizer* of P, and its size is $|L|$.

Example: The orders $L_1 = (a, b, c, d, e)$; $L_2 = (a, c, b, e, d)$; $L_3 = (a, b, c, e, d)$; are linear extensions of the partial order in figure 9.14. $\{L_1, L_2\}$ is realizer of this partial order.

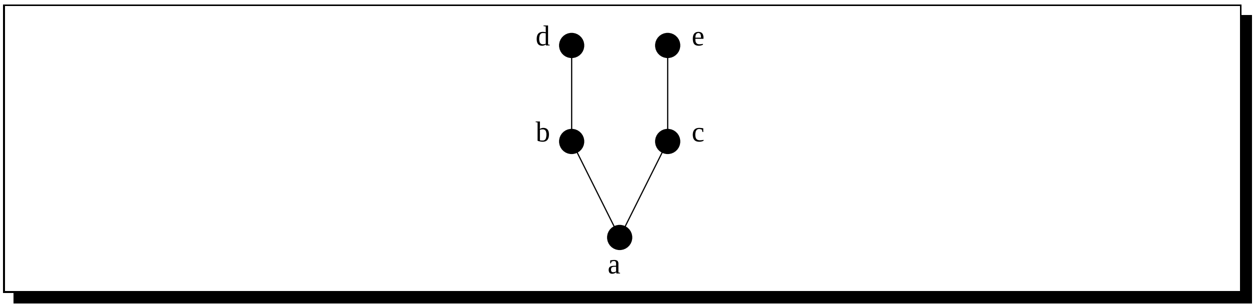

Figure 9.14: Example 8

A realizer of partial order P with the smallest size is called a *minimum realizer.* The *dimension* of partial order P, $dim(P)$ is the size of a minimum realizer of P.

The dimension of the partial order in figure 9.14 is 2.

Remark 9.5 *P is a complete order iff $dim(P) = 1$.*

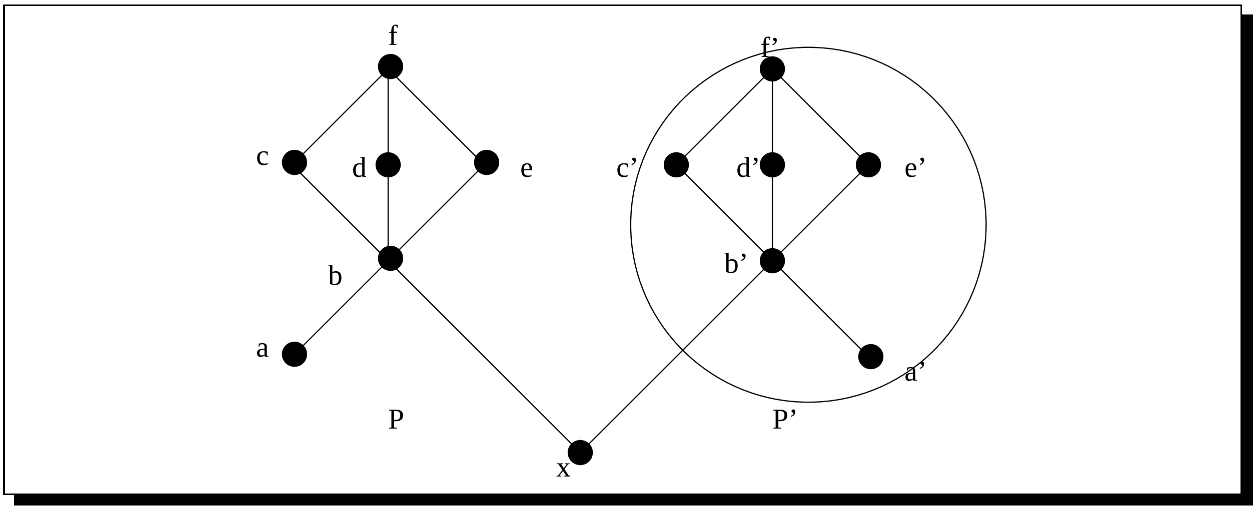

Figure 9.15: Example 9

Example: The orders

$$L_1 = (a, x, b, c, d, e, f, a', b', e', d', c', f'), L_2 = (a', x, b', c', d', e', f', a, b, e, d, c, f)$$

are linear extensions of the partial order P in figure 9.15, $\{L_1, L_2\}$ is the realizer of P, $dim(P) = 2$. And

$$L_1' = (b', e', d', c', f'), L_2' = (b', c', d', e', f')$$

are linear extensions of P' in figure 9.15, $\{L_1', L_2'\}$ is the realizer of P', $dim(P') = 2$.

Example: The orders $L_1 = (a, b, c', c, b', a'), L_2 = (b, c, a', a, c', b'), L_3 = (c, a, b', b, a', c')$ are linear extensions of the partial order P in figure 9.16, $\{L_1, L_2, L_3\}$ is realizer of P, and $dim(P) = 3$. To Show this, let us assume that $dim(P) = 2$, then there are two linear extensions l_1, l_2, that in one of them, say l_1, the elements (a, b, c) will be ordered: a, b, c. Then in l_2 these three elements must be ordered: c, b, a . l_1 must contain the subsequent a, b, c, b', a' and l_2 must contain the subsequent c, b, a, b' so we see that both extensions have b, b', a contradiction.

Lemma 9.6 *Let (X, P) be partial order. For every subset Y of X, $Y \subseteq X$ we have $dim(P_Y) \leq dim(P_X)$*

Proof: Let L be a realizer of P. Restricting L to the elements of Y yields a realizer of P_y. L is a minimum realizer for P so $dim(P_Y) \leq dim(P)$ ∎

Theorem 9.7 *(Hiraguchi 1951) Let $P = P_0[P_1, P_2, \ldots, P_k]$ be a composition of a disjoint partial orders (X_i, P_i) $i = 0, 1, \ldots, k$ then $dim(P) = max\{dim(P_i) | 0 \leq i \leq k\}$*

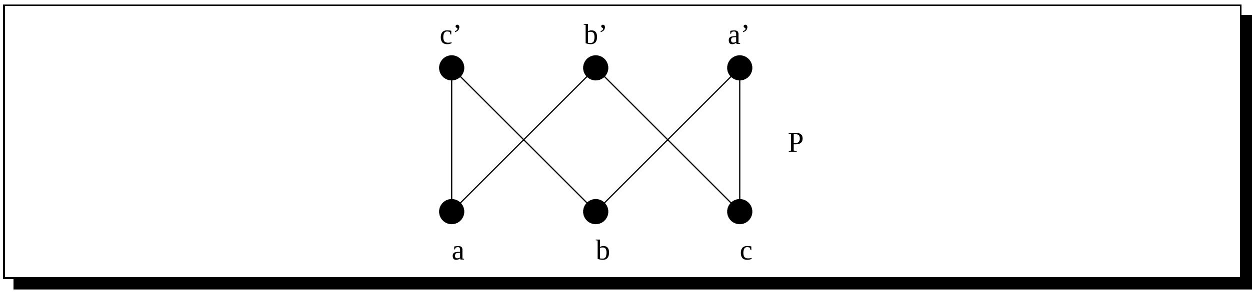

Figure 9.16: Example 10

Proof: Let $m = max\{dim(P_i)|0 \leq i \leq k\}$ and $\{L_{i_1}, L_{i_2}, \ldots, L_{i_m}\}$ be a realizer of P_i. Define $\wedge_i = L_{oj}[L_{1j}, L_{2j}, \ldots, L_{kj}]$, then $\{\wedge_j | j = 1, 2, \ldots, m\}$ is a realizer of P, so $dim(P) \leq m$, every P_i is a subset of P, so $dim(P) \geq max\{dim(P_i), 0 \leq i \leq k\} \longrightarrow dim(P) = max\{dim(P_i), 0 \leq i \leq k\}$ ■

For an example see figure 9.17.

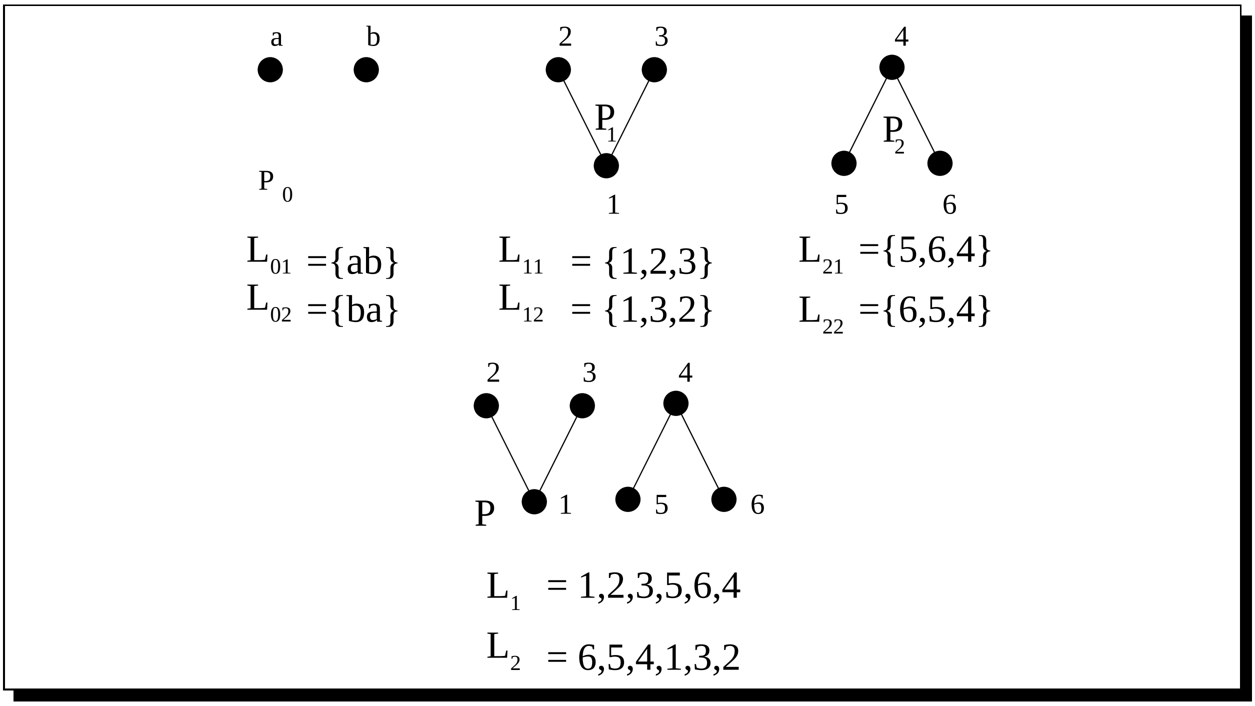

Figure 9.17: Example 11

Theorem 9.8 *(Dushnik-Miller, 1941) Let G be a comparability graph of poset $P, dim(P) \leq 2$ iff $\overline{G}$ the complement graph of G, is a comparability graph.*

Proof:

$\Rightarrow$: Let F be a transitive orientation of $\overline{G}$. Consider the graph $G'(V, E')$ when $E' = P + F$. This graph satisfies :

1. for every $a, b \in V$ either $ab \in E'$ or $ba \in E'$(but not both)
2. G' is transitive

To prove 2, note that if $ab, bc, ca \in E'$, it is obvious that ab, bc, ca cannot be in the same set (all in F or all in P), because F, P are transitive. So, two of these three edges are in the same set (F or P), say $bc, ab, ab \in F$ and by transitivity of F, we see that $ac \in F$ so $ca \notin P$. (1) and (2) imply that G' is a complete order on V. Because of the same reasons we know that $G''(V, E'')$ when $E'' = P + F^{-1}$ is a complete order on V. $L = \{P + F, P + F^{-1}\}$ is a realizer of P because if $ab \in P$ then $ab \in P + F, P + F^{-1}$, and if $ab \notin P$ then either $ab \in F$ and $ab \notin F^{-1}$, or $ab \in F^{-1}$ and $ab \notin F$. Hence $dim(P) <= 2$.

$\Leftarrow$: Assume that $dim(P) \leq 2$. Let $L = \{l_1, l_2\}$ be a realizer of P. Define $F = l_1 - P = (l_2 - P^{-1})$ (the equality holds because L is a realizer. To calculate $l_1 - P$ we look at l_1 as a complete directed graph and remove all the edges from P, see figure 9.18)

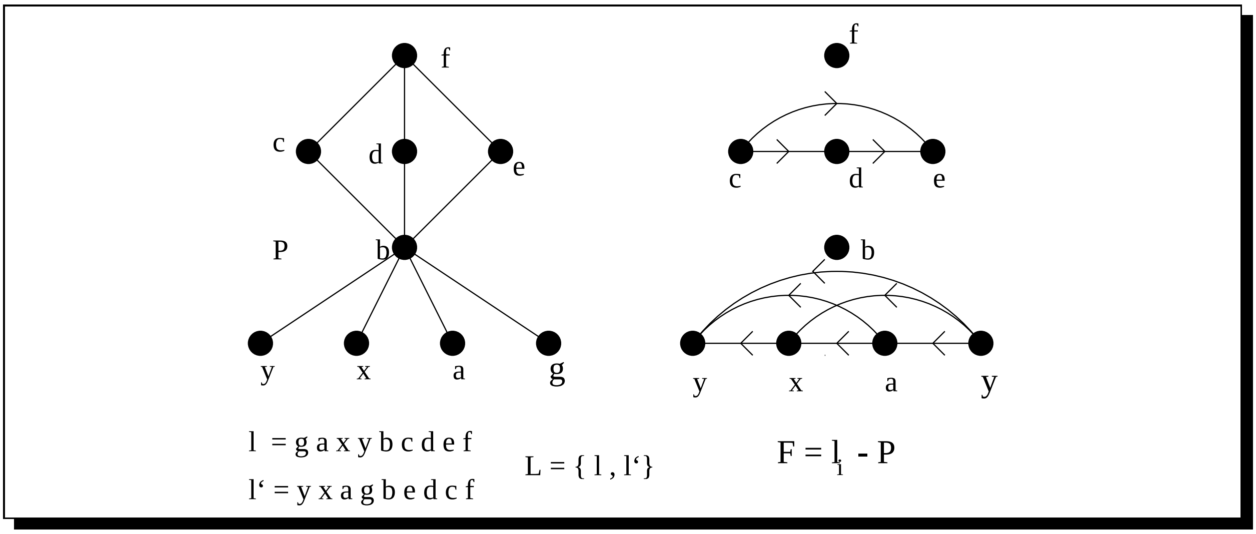

Figure 9.18: Example 12

We will show that F is a transitive orientation of $\overline{G}$, assume that $ab, bc \in F$ and $ac \notin F$. Then $ac \in P$. $ab, bc \in F \longrightarrow ba, cb \in l_2 \longrightarrow ca \in l_2 \longrightarrow ca \in P$ contradiction , hence F is transitive. ∎

Corollary 9.9 *Let G be a comparability graph of poset P. $dim(P) \leq 2$ iff G is a permutation graph.*

Proof: We will give two proofs :

1. $dim(P) \leq 2$ from the previous theorem $\longleftrightarrow \overline{G}$ is a comparability graph. From the theorem of Pnuely-Even-Lempel (1971), $G, \overline{G}$ are comparability graphs $\longleftrightarrow G$ is a permutation graph.

2. let us look at the matching diagram of permutation graph(see figure 9.19).

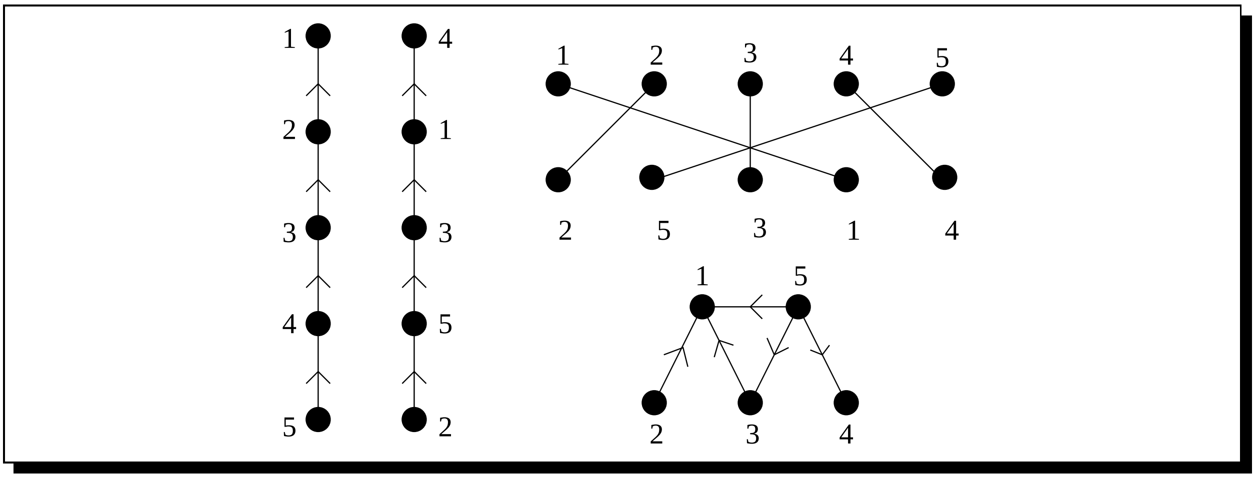

Figure 9.19: Example 13

$(i, j) \in E \longleftrightarrow [\pi^{-1}(i) > \pi^{-1}(j), i < j]$.Hence, the complete orders $l_1 = (n, n-1, \ldots, 1)$ and $l_2 = (\pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(n))$ is a realizer of P, so $dim(P) \leq 2$. Conversely, from the realizer $\{l_1, l_2\}$ we can construct the matching diagram in the same way. ■

The theorem of Dushnik-Miller tells that for two partial orders which have the same comparability graph, the dimension of both ≤ 2 or the dimension of both ≤ 2.

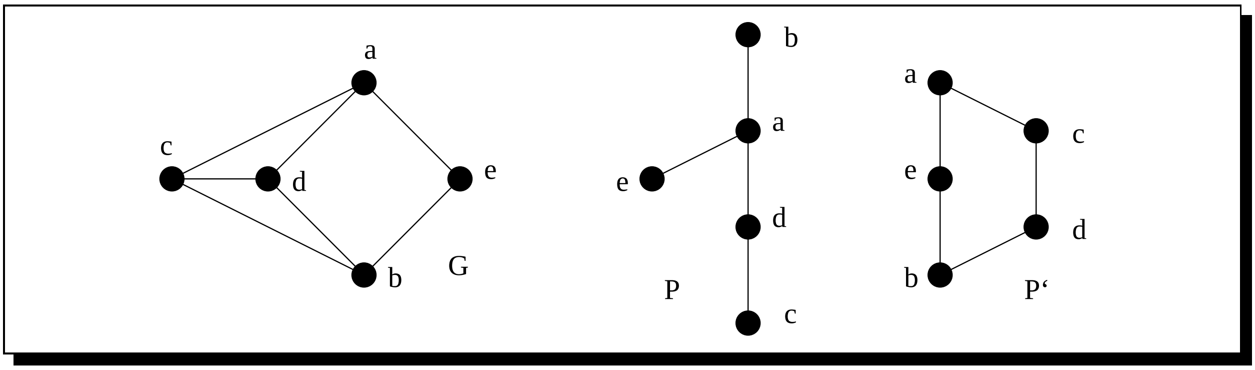

Figure 9.20: Example 14

Theorem 9.10 (Trotter, Moore, Sumner 1976, Gysin 1977) *Two partial orders which have the same comparability graph have the same dimension.*

Figure 9.20 gives an example of two partial orders that have the same comparability graph.

The problem of calculating the dimension of partial order mentioned as a major open problem in the Garey and Johnson book. In 1982, Yannakakis proved that the problem is NPC. In addition he proved deciding if $dim(P) \leq k$, is NPC, for every $k \geq 3$.