Received 23rd April 2021;

Permutation Graph

Nitesh Jain[1] | Abhishek Garg[2] | Pawan Kumar Gupta[3]

[1]nitesh.jain@iiitb.org, M.Tech,IIITB, Madhya Pradesh,India,
[2]abhishek.garg@iiitb.org, M.Tech,IIITB, Uttarakhand,India,
[3]pawan.gupta@iiitb.org, M.Tech,IIITB, New Delhi,India,

Summary

A permutation graph is the graph of inversions in a permutation. This paper broadly covers the definition of permutation graph, computational status of permutation graph such as Independent set, Coloring of permutation graphs and dominating set problem.

KEYWORDS:
permutation graph, intersecting graphs

1 | DEFINITION

An undirected graph G = (V , E) with vertices V = {1,2,..., n} is called a permutation graph if there exists a permutation on N = {1,2,..., n} such that for all i, j $\in N, (i - j)(\pi^{-}1(i) - \pi^{-}1(j)) < 0$
if and only if i and j are joined by an edge in G.

To a permutation $\pi = \pi 1 \cdots \pi$n $\in$ Sn, we associate a graph G as follows. The vertex set of is [n] and the edges are the pairs (i, j) such that i < j and i > j, that is, (i, j) is an inversion of π. Such a graph is called a permutation graph.

A permutation $\pi \in S_n$ is said to be indecomposable if there exists no positive integer k < n such that $\pi 1, ..., \pi k = [k]$.

2 | CONSTRUCTION OF PERMUTATION GRAPH

The permutation graph can be constructed in the following way:

- write numbers 1 to n on a line, then
- write them again on a separate, parallel line according to the order in which they appear in your permutation;
- connect each element from the first line to the same element on the second line (1 to 1, 2 to 2, ..., n to n),
- label each such connection with the numbers that it connects (e.g. connection 2 to 2 receives label 2);
- the resulting permutation graph is obtained by treating each connection as a vertex and connecting two vertices whenever the corresponding connections intersect.

If is a permutation of, the graph has vertices where xy is an edge of if and only if (x, y) or (y, x) is an inversion of. Any graph isomorphic to is called a permutation graph. In 1967 Gallai characterized permutation graphs in terms of forbidden induced subgraphs. In 1971 Pnueli, Lempel, and Even showed that a graph is a permutation graph if and only if both the graph and its complement have transitive orientations. In 2010 Limouzy characterized permutation graphs in terms of forbidden Seidel minors. Hence we characterize permutation graphs in terms of a cohesive order of its vertices. We show that only the caterpillars are permutation graphs among the trees.

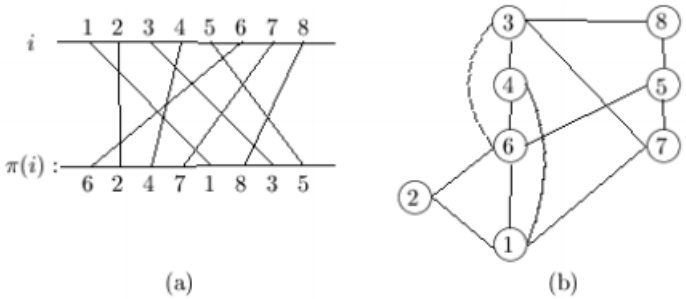

Figure 1 Matching diagram for the permutation (6, 2, 4, 7, 1, 8, 3, 5), below its corresponding permutation graph

3 | CHARACTERISTICS OF PERMUTATION GRAPH

Even permutation graph : Let n >= 3 be a positive integer, and let A_n be the set of even permutations in S_n.Then the graph G = (V, E) is called an even permutation graph whose vertex set is V = S_n and edge set E consisting of an unordered pair (f, g) f g is an edge such that either fg or gf is an even permutation,and it is denoted by G(S_n,A_n).

Odd permutation graph : For each n >= 3 be a positive integer, the graph G(S_n, B_n) is called an odd permutation graph whose vertex set is S_n and for each f,g ∈ S_n,the edge set is treated as E = {(f, g) : either fg or gf ∈ B_n}

Lemma 3.1. The complement of a permutation graph is also a permutation graph.Another property of the graph G[π] is that it is transitively orientable.

Lemma 3.2. The decreasing subsequences of π and the cliques of G[π] are in one-to-one correspondence. The increasing subsequence of π and the stable sets of G[π] are in one-to-one correspondence

Lemma 3.3. Two points p_{ij} , p_{kl} ∈ S(G rs are connected if (a) i < r and j > s or (b) i > r and j < s .A relation between connected points of S(G) and corresponding edges of G is established in the following lemma.

Lemma 3.4. Let p_{ij} , p_{kl} ∈ S(G) be two points corresponding to two vertices i, k ∈ V .The vertices i and k are connected by an edge in G if and only if p_{ij} and p_{kl} are connected.

Lemma 3.5. [104] The vertices corresponding to the points of any path from a leaf to the root of T(G) form a maximal independent set in G .

Theorem 3.1. An undirected graph G is a permutation graph iff G and its complement graph G are comparability graph.

Theorem 3.2. A maximum weight independent set of a permutation graph with n vertices can be computed in $O(n^2)$ time, provided the graph has at most O(n) maximal independent sets.

Theorem 3.3. The maximum weight k -independent set problem can be solved in$O(kn^2)$ time on permutation graph and log(kc$\sqrt{logc}$ +m) time on interval graph, where m and c represent respectively the number of edges and weight of the longest path of graph.

Theorem 3.4. The depth-first tree of a permutation graph can be constructed in O(n) time.

Theorem 3.5. Minimum cardinality 2-neighbourhood covering set of a permutation graph can be computed in O(n + m), where m is the number of edges of the complement graph.

Theorem 3.6. The time complexity to find the next-to-shortest path between any two specified vertices in permutation graph is $O(n^2)$.

Theorem 3.7. The all-pair shortest paths problem can be solved on permutation graphs in $O(n^2)$ time. Also, the average distance can be determined in $O(n^2)$ time.

Theorem 3.8. The minimum cardinality distance k -domination set on permutation graph can be determined in $O(n^2)$ time.

Theorem 3.9. The minimum 2-tuple domination set of a permutation graph can be computed in $O(n^2)$ time.

4 | RECOGNITION OF GRAPH CLASS

A transitive orientation algorithm gives rise to a linear-time bound for recognizing permutation graphs. The algorithm is based on the following characterization of permutation graphs: *An undirected graph G is a permutation graph iff G and its complement G are both comparability graphs.*

When G is a permutation graph, the algorithm finds a topological sort of a transitive orientation D of G and a topological sort of a transitive orientation D' of G. D ∪ D' is a tournament (an orientation of a complete graph) and acyclic. It then finds the unique topological sort of D ∪ D' to yield a linear ordering of V , and the unique topological sort of D^T ∪ D' to give a second linear arrangement of V . By results these two linear arrangements are a permutation model of G. When G is not a permutation graph, the procedure in produces a faulty permutation model of G. Success or failure of an authentication algorithm on the permutation model it produces provides the basis for deciding whether the graph is a permutation graph in that algorithm. The procedure is not a certifying algorithm, since the permutation model could also have been faulty due to a bug in the implementation.The permutation model is therefore a weak certificate.

The algorithm for recognizing permutation graphs uses the transitive orientation algorithm to find linear extensions of orientations D and D' of G and of G'. Since it provides a certificate if G is a permutation graph, we will assume in the remainder of the paper that G is not. In this case, at least one of D and D' has an incompatible pair.We now describe how to find an incompatible pair in D or D' in time linear in the size of G, given G and linear extensions π and τ of D and of D' . This constitutes proof that the implementation of the transitive orientation algorithm failed to produce an orientation of G or of G' that is transitive. However, it is not a certificate that G is not a permutation graph, since the failure could be due to a bug in the implementation of the algorithm.

Lemma 4.1. Let G be a graph, and let D and D' be acyclic orientations of G and G'. Then D ∪ D' and D^T ∪ D' are both acyclic iff D and D' are each transitive.

Lemma 4.2. Let G, D, and D' be as in Lemma 4.1. Given a three-cycle in D ∪ D' or D^T ∪ D', it takes O(1) time to return an incompatible pair in D or in D'.

Lemma 4.3. If for each i ∈ {0, 1,...,n1} there exists x ∈ V such that p(x) = i,then D D' is acyclic.

Lemma 4.4. Let G, D, and D' be as in Lemma 4.1, and let π and τ be topological sorts of D and D', respectively. Given G, π and τ, it takes O(n + m) time to find a three-cycle in D D' or else determine that D D' is acyclic.

Another approach to same problem: Permutation graphs are intersection graphs of segments between two parallel lines. Thereby any permutation graph can be represented by assigning, to each vertex, two integers describing the position of its segment extremities. A permutation graph is a comparability graph the complement of which is also a comparability graph and this is a characterisation. It follows that any permutation graph G and its complement G' can be transitively oriented. The best known permutation graph recognition algorithm makes use of this property. The links between transitive orientation and modules of graphs are well understood since Gallai's work. Indeed, to compute a transitive orientation of a graph, a modular decomposition preliminary step is required.

Linear time permutation graph recognition algorithm :

[1] a preprocessing step first computes the modular decomposition of the input graph G;

[2] then a transitive orientation algorithm is applied to compute a first linear ordering of the vertices, which defines a transitive orientation of G whenever G is comparability;

[3] a second linear ordering is computed, corresponding to a transitive orientation of G if it exists. It turns out that the input graph G is a permutation graph if and only if these two linear orderings define a realiser of G, which can be tested in linear time.

As the three steps described above also require linear time, the whole algorithm is linear. As far as we know, no linear time (non-dynamic) recognition algorithm for permutation graphs avoiding the computation of the modular decomposition has been designed.

5 | VERTEX COVER

Minimum vertex cover problem: given a graph G(V, E) find a subset S $\subseteq$ V with minimum cardinality such that every edge in E has at least one endpoint in S. Consider the following greedy algorithm. Find the highest degree vertex, add it to the vertex cover S and remove it along with all incident edges. Repeat iteratively. Prove that this algorithm has an unbounded approximation factor i.e. for any c there exists an input graph G such that |S| $\geq$ c OPT.

For a given graph G. = (V, E) and integer k, the Independent Set problem is to find whether G contains an Independent Set of size $\geq$ k. Same, for a given graph G = (V, E) and integer k, the Vertex Cover problem is to find whether G contains a vertex Cover of size $\leq$ k. *Independent Set and Vertex Cover problems are very similar and we will show reduction of independent set to vertex cover problem and then prove the independent set problem for permutation graphs.*

In a graph G = {V, E},S is an independent Set $\iff$ V — S) is a Vertex Cover.

1. If S is an Independent Set ,there is no edge e = (u,v) in G, such that both u,v $\in$ S.Hence for any edge e = (u,v), atleast one of u,v mustlie in (V — S).
$\Rightarrow$ (V — S) is a vertex cover in G.

2. If (V — S) is a Vertex Cover, between any pair of vertices (u,v) $\in$ S if there exist an edge e,none of the endpoints of e would exist in (V — S) violating the definition of vertex cover.Hence no pair of vertices in S can be connected by an edge.
$\Rightarrow$) S is an Independent Set in G.

Hence G contains an Independent Set of size k $\iff$ G contains a Vertex Cover of size |V| — k.

6 | INDEPENDENT SET

A subset of the vertices of a graph G = (V, E) is an independent set Z if no two vertices in the subset are adjacent. An independent set is maximal if any vertex not in the set is dominated by at least one vertex in the set. Z(G) is the cardinality of a maximum independent set. We present an O(n log n) algorithm for finding a maximum independent set in a permutation graph. Permutation graphs can be viewed as permutation diagrams or matching diagrams.There is an algorithm for finding a maximum independent set in a permutation graph by using the binary insertion technique. We will generate stacks to store the relation of the positions of numbers in the permutation.

Suppose P = (7, 2, 4, 8, 1, 9, 3, 5, 10, 6). This can be represented as the permutation diagram of Fig. 2. We process the top row of the permutation diagram from left to right. First 1 is placed on the stack 1. The 2 goes in front of 1on the stack 1 because it intersects 1. The 3 cannot go in the stack 1 since 3 does not intersect 2, which is at the front of the stack 1, so put it in the stack 2. The 4 is placed in front of 3 on the stack 2 since 4 does not intersect 2 but does intersect 3. The 5 goes in the stack 3 because 5 does not intersect 4 and 2, which are at the front of the stack 2 and 1, respectively. The 6 goes in stack 4 because 6 does not intersect 5, 4 and 2. The 7 is placed at the front of the stack 1 because 7 intersects 6, 5, 4 and 2. The 8 goes in front of 5 on the stack 3 since 8 does not intersect 4 and 7 but intersects 5. The 9 goes in front 6 on the stack 4 since 9 intersects 6 but does not intersect 8, 4 and 7. The 10 goes in the stack 5 because 10 does not intersect 9, 8, 4 and 7. The generated stacks are shown in Fig. 2. Then we scan the stacks, which have been generated, from stack 5 to stack 1. We select one single number from each stack so that those numbers are in a decreasing sequence. Now we select 10 from the stack 5, 9 from the stack 4, 8 from the stack 3, 4 from the stack 2, and 2 from the stack 1. Then the generated set {10, 9, 8, 4, 2} is a maximum independent set of G(P) and the set (7, 2, l) is a maximal clique of G(P).

During the jth iteration, j is placed on top of the stack i whenever j does not intersect the front entries of the stack 4 but intersects the front entry of the stack r, where 1 <q < i and i f r < m, respectively. A binary search is used to find the corresponding position of j on the top of the stack i. Let T(i) be the number on top of the stack i. Note that P,-' is the position of the number i in the permutation P.

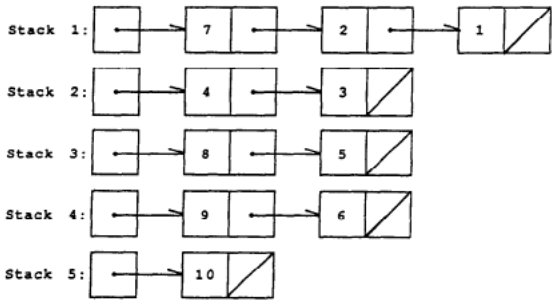

Figure 2 Linked stacks

Ind-Algorithm (Maximum independent set in a permutation):

Input: A permutation P = [P,, P2,. . . , P,] of N=[l,2 ,..., n]

Output: A maximum independent set of G(P).

Method. During the jth iteration, j is placed on top of the stack i whenever j does not intersect the front entries of the stack four but intersects the front entry of the stack r, where 1 <q < i and i f r < m, respectively. A binary search is used to find the corresponding position of j on the top of the stack i. Let T(i) be the number on top of the stack i. Note that P,-' is the position of the number i in the permutation P.The entire algorithm is as follows:

Procedure Maximum_ind();

```
{Find a maximum independent set}
begin
  Push (1, 1)
    {Place 1 on the top of stack l}

  for j = 2 to n do
    {Place j on the top of stack i}
    begin
      Correspond_stack:
      Push ( j, i);
    end:

  Pop( T, max, M );
     {Pop T from the last stack m. T < max}
  Ind:= T;

  for k = i - 1 down to 1 do
    begin
      Pop (7T, k);
      Ind:=Ind+ s;
      T:=s
    end:
end;
  {End of the Maximum_ind}
```

Procedure Correspond_stack;

```
{Find the corresponding stack i }
begin
   high := i: low := 1; found := 0:
  if P_j^{-1} < P_{T(1)}^{-1} {Stack1isfound}
    then k := 1, found := 1;
  if P_j^{-1} < P_{T(m)}^{-1} {Needanotherstack}
    then k := m + 1, found := 1;

  While Not found or high > low do
    begin
      k := (high + low)/2;
      if P_{T(k)}^{-1} > P_{T(k-1)}^{-1}
         then found := 1;
      else
         if P_j^{-1} > P_{T(k-1)}^{-1}
           then low := k + 1
         else high:=k-1;
     end:
  i := k
end;
```

Any stack in the Ind-Algorithm corresponds to a clique. During the jth iteration, the j is placed on top of stack i, and j does not intersect the top of stack (i - 1) but intersects the top of all stacks r with r > i. An independent set is generated from the stacks that have been constructed by the Ind-Algorithm. The Ind-Algorithm generates the partitions of minimum number of cliques.As we can see above, each number is placed on top of an existing stack, if possible. The Ind-Algorithm continues this way until no more numbers are left. Then the Ind-Algorithm generates a minimum number of stacks.The Ind-Algorithm is guaranteed to find a maximum independent set of a permutation graph in O(n log n) time.

We know that the Ind-Algorithm generatesa minimum number of cliques and permutation graphs are perfect and hence the clique number equals the stability number . Since the stability number is always the clique cover number, any algorithm that finds an independent set with size equal to a clique cover number finds the stability number. Therefore, our algorithm is guaranteed to find an independent set, which is maximum. Our algorithm needs O(n log n) time to generate stacks and O(n) time to find a maximum independent set from the stacks that have been constructed. Therefore, finding a maximum independent set in a permutation graph needs O(n log n) time.

7 | COLORING PROBLEM

A coloring of a graph G is an assignment of colors to its vertices so that no two adjacent vertices have the same color. Yu and Chen proposed a technique that transfers the coloring problem into the largest-weight path problem.Their algorithm solves the coloring problem in $O(\log^2 n)$ time with $O(n^3/logn)$ processors on the Crew Pram, or in $O(logn)$ time with $O(n^3)$ processors on the Crew Pram model. They also proposed parallel algorithms that solve the weighted clique problem, the weighted independent set problem, the clique cover problem, and the maximal layers problem with the same time and processor complexity.

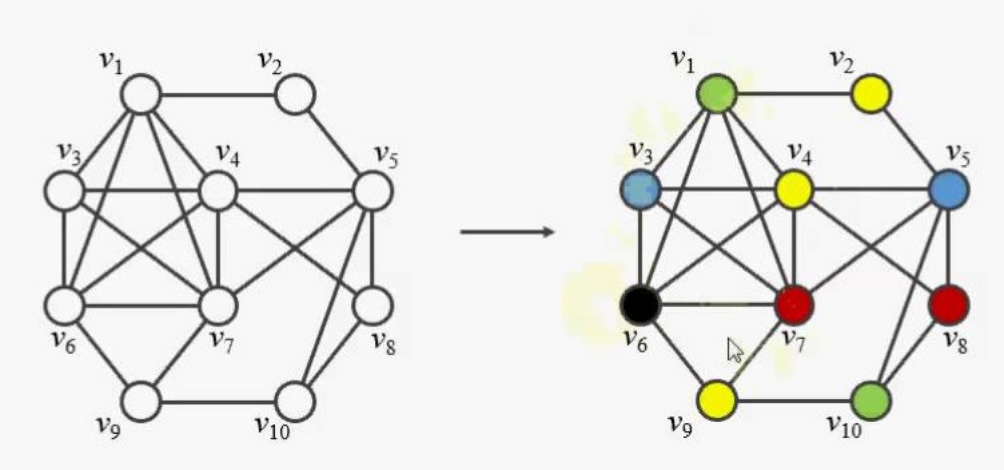

Figure 3 No adjacent vertices are allocated the same color and also the number of colours used is minimised.

Andreou and Nikolopoulos designed a parallel algorithm which solves the problem of coloring a permutation graph of size n in $O(log^2 n)$ time using $O(n^3/log3n)$ processors on the Crew Pram model of computation.Moreover, they showed that the coloring problem on permutation graphs can be solved in $O(lognloglogn)$ time in the average-case with $O(n^2)$ processors. Stavros D.Nikolopoulos presented a parallel algorithm for the problem of coloring a permutation graph, which runs in $O(log2n)$ time with $O(n^2/logn)$ processors on the Crew Pram model.

His algorithm uses a strategy to transform a permutation graph G into a rooted tree T and solves the coloring problem on G by computing the vertex-level function on T. He designed his algorithm using certain combinatorial properties of the lattice representation of a permutation and relationships between permutations, directed acyclic graphs and rooted trees having specific key properties.Specifically, given a permutation π (or its corresponding graph $G[\pi]$), he first constructs a rooted tree $T[\pi]$ by exploiting the inversion and d-inversion relations of the element of π and, then, from the tree $T[\pi]$ he constructs a rooted tree $T[\pi]$ which we call coloring-permutation tree or cp-tree for short. He proves that the problem of coloring a permutation graph $G[\pi]$ is equivalent to the problem of finding the level of each node of the cp-tree $T[\pi]$. We show that the cp-tree of a permutation can be constructed in $O(log^2 n)$ time with $O(n^2/logn)$ processors on the Crew Pram model.Since the level of each vertex of a tree can be computed in $O(logn)$ time with $O(n/logn)$ processors on the Crew Pram model using the well-known Euler-tour technique, it follows that the coloring problem on permutation graphs can be solved in $O(log^2 n)$ time with $O(n^2/logn)$ processors on the Crew Pram model.

8 | DOMINATION SET PROBLEM

A set S of vertices of a graph G = (V, E) is dominating if every vertex in V, S is adjacent to some vertex in S, and is independent if no two vertices in S are adjacent. If the vertices of G are assigned non-negative weights, then the weight of S is the sum of the weights of its elements. We remark that the class of cographs is properly contained in the class of permutation graphs. Hence these results refine the "boundary" between the "hard" and "easy" in- stances of these problems as previously established by the NP-completeness results for comparability graphs, on the one hand, and the polynomial domination algorithm for cographs, on the other.

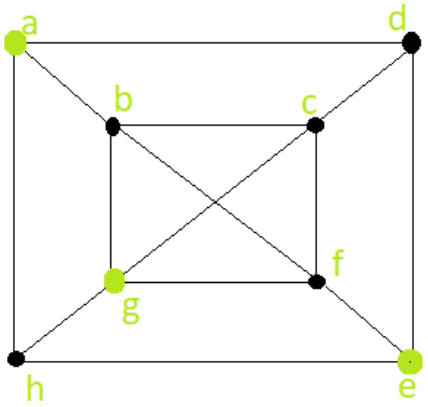

Figure 4 In this graph of 8 vertices and 14 edges.Every black vertex is adjacent to alteast one green vertex,which is then said that the black vertex is being dominated by green vertex.So,dominant set of S is {a,g,e}

Given a graph G = (V, E), it is possible to determine whether G is a permutation graph and, if so, find the bijections L_1 and L_2 of the definition in O(n_2) time, where n = mod(V). Throughout the remainder of this paper we will assume that L_1 and L_2 are known. To simplify the notation we will represent the vertex u by the number L_1(u); and we will denote by s the permutation L_2 o L_1^{-1} With this notation we have ij E E if and only if (i -j)(π(i) - π(j)) < 0.

Each of the algorithms to be presented uses dynamic programming. In the ith stage of each algorithm we cull information about the subgraph induced i and then use this information in the next stage. We will by {1,2.....i} present three algorithms. The first two will be O(n^3) weighted domination and weighted independent domination algorithms. The third will be an O(n^2) cardinality domination algorithm. Unfortunately, we have not been able to find any algorithm for the cardinality independent domination problem that is faster than simply using the weighted independent domina- tion algorithm with unit weights.

Before proceeding to the algorithms we mention a definition and provide some necessary notation. A set S of vertices of G dominates another set T of vertices of G if every vertex in T and S is adjacent to some vertex in S. Let G be a permutation graph on the vertices {1,2,. . . , n } and let π be a defining permutation of G, as described above. For each i, j $\in$ { 1,2, . . . , n }, we let V_{ij} = {1,2,.....i} $\bigcap$ {π^{-1}(1), π^{-1}(2).....π^{-1}(j)} & $V_{i0} = \phi$.

We might find it helpful to visualize two columns, each consisting of the numbers 1,2,. . . , n in their natural order, with a straight line joining k in the left column with π(k) in the right column. The graph G is then the intersection graph of these line segments, and the vertex k is represented by the line segment with k as its left endpoint and π(k) as its right endpoint. V_{ij} consists of those vertices whose corresponding line segments have their left endpoints in the set {1,2,. . . , i} and their right endpoints in the set {1,2.....j}.

The weighted domination algorithm will find, for each i, k $\in$ { 1,2, . . . , n } and j $\in$ {0,1,2,. . ., n }, a set S_{ijk} . This set will be a minimum weight set of vertices in V_{ik} containing π^{-1}(k) and dominating V_{ij}, if such a set exists. Otherwise, S_{ijk} will be a dummy set, N (nonexistent), whose weight is defined to be 1 plus the weight of V. A minimum weight dominating set of G will then be 'a minimum weight member of { S_{nn1}, S_{nn2}, . . . , S_{nnn} }.

The algorithm is as follows:

STAGE 1 : for j=0 to n **do**
for k = 1 to n **do**
S_{1jk} = {1} , if k = π(1)
= N otherwise
end
end

STAGE i (1<i≤ n):
for k=1 to n **do**
S_{i0k} = {π^{-1}(k)} , if π^{-1}(k) ≤ i
= N otherwise
end
for j = 1 TO n **do**
for k = 1 TO n **do**
if j < π(i), then
S_{ijk} = S_{i-1jk} , if k ≠ π(i)
= Any minimum weight member of
{$S_{ij} \bigcup$ {i} : l<k } , if k = π(i)
if j = π(i), then
S_{ijk} = S_{i-1jk} , if k ≥ j and π^{-1}(k) ≤ i
= N otherwise
if j > a(i), then
S_{ijk} = S_{ikk}, if k = $\pi(i)$
= The better of S_{i-1jk} and $Si\pi(i) - 1k \bigcap$ {i}
if k > π(i) and π^{-1}(k) ≤ i(*)
= N otherwise
end
end

STAGE n + 1: D = Any minimum weight member of {S_{nn1},S_{nn2}.....S_{nnn}}

Dominating Set remains NP-complete for restrictions to bipartite and to split graphs.In contrast to this show that Dominating Set together with some variants restricted to permutation graphs is solvable in polynomial time.The problem Dominating Clique is NP-complete. Now we consider the restriction of Dominating Clique to permutation graphs.If the permutation graph G_T has a dominating clique then G_T has also a dominating clique with at most 2 elements.A straightforward algorithm can decide Dominating Clique for permutation graphs in O(n^3) steps.

Independent Dominating Set can be solved in O(n^2) steps for permutation graphs.Now to the problem Dominating Set itself,we have described the principle of an algorithm above which solves this problem in polynomial time for permutation graphs.

Dominating Set can be solved in O(n^4) steps for permutation graphs. Connected Dominating Set is the modification of Dominating Set where it is additionally required that the dominating set V' should also be connected.Connected Dominating Set can be solved in O(n^5) steps for permutation graphs.

Each connected permutation graph has a dominating path (which can be constructed in 0(n^2 steps). It is an open problem whether the construction of a longest dominating path (i.e. the solution of the HAMILTONIAN PATH problem) or a shortest one is possible in polynomial time for permutation graphs.

References

Brandstädt, A., & Kratsch, D. (n.d.). On the restriction of some np-complete graph problems to permutation graphs. *Fundamentals of Computation Theory Lecture Notes in Computer Science*, 53–62. doi: 10.1007/bfb0028791

Farber, M., & Keil, J. M. (1985). Domination in permutation graphs. *Journal of Algorithms*, *6*(3), 309–321. doi: 10.1016/0196-6774(85)90001-x

Gh., B. B. (2015). (g1,g2)-permutation graphs. *Discrete Mathematics, Algorithms and Applications*, *07*(04), 1550051. doi: 10.1142/s1793830915500512

Kim, H. (1990). Finding a maximum independent set in a permutation graph. *Information Processing Letters*, *36*(1), 19–23. doi: 10.1016/0020-0190(90)90180-6

Koh, Y., & Ree, S. (2007). Connected permutation graphs. *Discrete Mathematics*, *307*(21), 2628–2635. doi: 10.1016/j.disc.2006.11.014

Limouzy, V. (2010). Seidel minor, permutation graphs and combinatorial properties. *Algorithms and Computation Lecture Notes in Computer Science*, 194–205. doi: 10.1007/978-3-642-17517-6$_1$9

□