CHAPTER THREE
Chapter-3

On the recognition of some classes of sc perfectly orderable graphs

3.1 Introduction

In this chapter, we extend our study to the other subclasses of sc perfect graphs, such as sc brittle graphs, sc quasi-chordal graphs and $P_3$-free sc weakly chordal graphs. One of the important relation between all these 3 classes is that they are subclasses of sc perfectly orderable graphs.

In section 3.2, we study brittle and sc brittle graphs and propose a recognition algorithm for sc brittle graphs and then obtain the catalogue of sc brittle graphs. In section 3.3, we study quasi-chordal and sc quasi-chordal graphs and present an algorithm for the recognition of sc quasi-chordal graphs followed by its catalogue compilation. Last section 3.4 is devoted for the same problem for $P_3$-free sc weakly chordal graphs.

3.2 Sc brittle graphs and its recognition

Let us first recall an induced $P_4$ with vertices $a, b, c, d$ and edges $ab, bc, cd$ then we refer to the vertices $a$ and $d$ as the endpoints, and the vertices $b$ and $c$ as the midpoints of the $P_4$. Based on this notation, Chvátal [25] defined brittle graph as follows: a graph $G$ is brittle if each induced subgraph $H$ of $G$ contains a vertex that is not a midpoint of any $P_4$ or not an endpoint of any $P_4$. 

55
In [74], Hoàng and Khouzam studied brittle graphs and showed among other things that brittle graphs can be recognized in $O(n^3m)$ time. Using the concept of brittle order Hoàng and Reed [72] gave an algorithm which recognizes brittle graph in $O(n^5)$ time. After that Schäffer [120] dealt specifically with the recognition problem for brittle graphs and gave $O(m^3)$ time recognition algorithm derived from definition but this algorithm is much complicated. In a technical report [126], Spinrad and Johnson also designed an $O(n^3 \log^2 n)$ algorithm for brittle graphs.

Later Eschen et al. [42] presented two algorithms for recognition of brittle graphs by direct application of the definition. First they presented an algorithm that requires $O(n)$ adjacency matrix multiplication as its bottleneck step which yields an $O(n^{3.376})$ time bound algorithm. Then they presented an algorithm that uses modular (or substitution) decomposition. When used together with known algorithms for modular decomposition and on-line maintenance of spanning trees, this approach yields an $O(n^3 \log^2 n)$ time deterministic or $O(n^3)$ time randomized recognition algorithm for brittle graphs.

In this section, we study the problem of recognition of sc brittle graphs, the proposed recognition algorithm employs the idea similar to those used in above mentioned algorithms. Sc brittle graphs can be recognized using the vertex elimination scheme i.e. if all its vertices eliminated by successive deletion of no-mid and no-end vertices, where a vertex of a graph $G$ is called no-mid if it is not the midpoint of any $P_4$ in $G$. Similarly, a vertex of a graph $G$
is called no-end if it is not the endpoint of any $P_4$ in $G$. If a graph $G$ does not contain any induced $P_4$ then all the vertices should be treated as no-mid as well as no-end vertices. No-mid and no-end vertices in sc brittle graphs have same structure. To ensure this we give following Theorem, which relates no-mid vertex to no-end vertex in a sc brittle graph.

**Theorem 3.1.** Let $G$ be a sc graph, then if there exists any no-mid vertex in $G$, then there also exists a no-end vertex in $G$. Converse is also true.

**Proof.** Let $G$ be a sc graph, suppose a vertex $v$ be a no-mid in $G$, then by definition of no-mid vertex it is not the middle vertex of any $P_4$ in $G$. Now in the complement of the graph $G$, the same vertex $v$ becomes no-end vertex because the complement of a $P_4$ is also a $P_4$. Since $G$ is sc graph, therefore if there exists any no-mid vertex in $G$ then there also exists a no-end vertex in $G$. The same argument is also true for the converse. Hence the Theorem. □

The following corollary is immediate from the above Theorem-3.1.

**Corollary 3.2.** Let $G$ be a sc graph, if there exists no no-mid vertex in $G$ then there will be no no-end vertex in $G$. Converse is also true.

To recognize sc brittle graphs, we have to find first a vertex which is either a no-mid or no-end, for this we give an algorithm-3.1, which finds a no-mid or no-end vertex in a given sc graph, moreover it is used as a subroutine in the next algorithm-3.2. Algorithm-3.1, first computes all the induced $P_4$'s of the given graph $G$, then it checks whether the given sc graph $G$ contains any no-mid or no-end vertex or none of these two.
Algorithm 3.1: An Algorithm for no-mid and no-end vertices.

**Input:** A sc graph $G$.

**Output:** Vertex set of no-mid and no-end vertices.

**Step-1:** no-mid set = $\phi$, no-end set = $\phi$ and $R_v = \phi$ (where $R_v$ is the set of vertices which are neither no-mid nor no-end in $G$)

- list all the induced $P_4$'s of $G$.

**Step-2:** If no induced $P_4$ found then return “Input graph has no induced $P_4$”.

- Stop.

**Step-3:** select arbitrary vertex ‘$u$’

- if it is not a middle vertex of any $P_4$ in $G$, then
  - put the vertex ‘$u$’ in no-mid set.
- else
  - if it is not an end-vertex of any $P_4$ in $G$, then
    - put the vertex ‘$u$’ in no-mid set.
  - else
    - put the vertex ‘$u$’ in $R_v$.

**Step-4:** If all the vertices scanned then

- Stop.

Else goto step-3.

**End.**

The time complexity of the algorithm-3.1 is as follows.

**Complexity.** Since all the $P_4$'s are generated in $O(n^2 m)$ time [120] and their can be at most $O(n^2 m)$ $P_4$'s in $G$, so step-1 can be done $O(n^2 m)$ time. From step-3 to step-4, we require one more iteration of size $n$, each of which requires $O(n^2 m)$ time. So overall time complexity is $O(n^3 m)$.
Once we compute no-mid vertex set and no-end vertex set, we are in a position to discuss algorithm for recognition of sc brittle graphs. Algorithm-3.2, which recognizes whether a given sc graph is sc brittle or not works as follows: first it computes no-mid and no-end vertex set in step-1 by using algorithm-3.1, as well as if it finds no induced $P_4$'s then algorithm decides that the given graph is sc brittle. Now if both the sets i.e. no-mid and no-end vertex sets are empty then algorithm terminates and gives output that the given sc graph is not sc brittle, otherwise algorithm proceeds to the next step i.e. step-4. In this step, it deletes vertex either from no-mid or no-end vertex set (note that the choice of no-mid or no-end vertex to delete is arbitrary since the deletion of a vertex cannot cause another vertex to become the midpoint or endpoint of a $P_4$). Then the algorithm repeats the procedure from step-1 to step-4. In this way if all the vertices are eliminated then algorithm-3.2 decides that the given sc graph is sc brittle. Algorithm-3.2 is as follows.

Algorithm 3.2: *An Algorithm for recognition of sc brittle graph.*

| Input: | A sc graph $G$. |
| Output: | "$G$ is sc brittle graph" or "$G$ is not sc brittle graph". |

**Step-1:** Compute the no-mid and no-end vertex set (using algorithm-3.1), and elimination order $= \phi$

**Step-2:** If graph has no induced $P_4$'s then print "Graph $G$ is sc brittle".

Stop.

**Step-3:** If no-mid set $= \phi = $ no-end set, then
return “G is not sc brittle”.

Stop.

**Step-4:** eliminate vertex ‘u’ either from no-mid set or no-end set.

Put ‘u’ in elimination order.

**Step-5:** update the graph and goto step-1.

**Step-6:** If all the vertices of G are eliminated in this way, then

“G is sc brittle” and

print the vertices of the elimination order.

else

“G is not sc brittle”.

End.

The time complexity of the algorithm-3.2, can be computed as follows.

**Complexity.** In algorithm-3.2, the bottleneck is step-1, which requires $O(n^3m)$ time. All the other steps have lower complexity. So the overall time complexity is $O(n^3m)$.

The following result justifies the claim of algorithm-3.2.

**Theorem 3.3.** Algorithm-3.2 checks whether an input sc graphs is sc brittle or not correctly.

**Proof.** By the definition of brittle graphs, its each induced subgraph contains either a no-mid or no-end vertex. So we start with a sc graph G and look for no-mid or no-end vertex, if found, remove that vertex (let it be u). Now by definition of brittle graphs, $G - u$ again contains no-mid or no-end vertex, if
found, remove that vertex again. Clearly if all the vertices are removed in this manner, then we are left with no vertex and the graph $G$ is SC brittle. If while deleting the vertices we found that there does not exist any no-mid or no-end vertex in any subgraph of $G$, then at that stage we decide that the graph is not SC brittle. Hence the Theorem. □

To illustrate algorithm-3.2, we consider the following SC graphs $G_1$ and $G_2$ as shown in figure-3.1(a) and figure-3.1(b) respectively.

Figure-3.1

Let graph $G_1$ be the input to algorithm-3.2. Step-1 of algorithm-3.2 computes its no-mid set as $\{v_3,v_4\}$ and no-end set as $\{v_7,v_8\}$, by calculating all induced $P_4$’s as $[v_1,v_3,v_6,v_2], [v_1,v_5,v_6,v_3], [v_2,v_3,v_5,v_4], [v_2,v_5,v_6,v_4], [v_2,v_6,v_8,v_4], [v_2,v_7,v_1,v_4], [v_2,v_7,v_8,v_4], [v_2,v_8,v_6,v_4], [v_3,v_6,v_8,v_3], [v_3,v_6,v_8,v_5], [v_3,v_7,v_1,v_4], [v_3,v_7,v_1,v_5], [v_3,v_7,v_2,v_3], [v_3,v_7,v_8,v_4], [v_3,v_7,v_8,v_5], [v_4,v_1,v_7,v_6]$ and $[v_5,v_1,v_7,v_6]$ using algorithm 3.1. Since both the no-mid and no-end sets are not empty thus algorithm proceeds to step-4. In step-4 any vertex from either no-
mid or no-end vertex set can be deleted, let it be \( v_3 \). The graph is updated in step-5 and then algorithm repeats the procedure. In this way algorithm-3.2 successfully eliminates all the vertices of \( G_1 \) and produces output as \( G_1 \) is a sc brittle graph. The overall procedure of elimination of vertices from no-mid and no-end set can be seen in figure-3.2, where a vertex which is enclosed by dotted line is deleted vertex.

### 3.2.1 Catalogue compilation of sc brittle graphs

Using algorithm 3.2, we compile the catalogue of sc brittle graphs with at most 17 vertices from the available catalogue of sc graphs with at most 17 vertices.

<table>
<thead>
<tr>
<th>Number of vertices $n$</th>
<th>4</th>
<th>5</th>
<th>8</th>
<th>9</th>
<th>12</th>
<th>13</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of sc graphs</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td>36</td>
<td>720</td>
<td>5600</td>
<td>703760</td>
<td>11220000</td>
</tr>
<tr>
<td>Number of sc brittle graphs</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>82</td>
<td>82</td>
<td>5912</td>
<td>5912</td>
</tr>
</tbody>
</table>

Table-3.1
3.3 Quasi-chordal and sc quasi-chordal graphs

Quasi-chordal graphs were introduced by Voloshin [147] as a generalization of chordal graphs. Hoàng and Mahadev also gave the similar concept in [75], where they called quasi-chordal graphs as good graphs. The following Theorem was known to researchers and referred in the literature but its proof had never been published. Recently Gorges et al. [57] proved the Theorem.

**Theorem 3.5 [57].** For a graph $G$, the following conditions are equivalent:

(i) $G$ is quasi-chordal.

(ii) $G$ does not contain a latticed subgraph as an induced subgraph.

(iii) $G$ admits a good order.

where an order $v_1 < v_2 < ... < v_n$ on a graph $G$ is good if, for any induced subgraph $H$ of $G$, either the largest vertex of $(H, <)$ is simplicial or the smallest vertex of $(H, <)$ is co-simplicial and a graph with each vertex belonging to some hole and some antihole is called latticed.

Recently, in [79] Hoàng et al. reported the following result.

**Theorem 3.6 [79].** If $G$ is a weakly chordal graph such that every pair of squares meet in a non-edge, then $G$ is a quasi-chordal graph.

The above Theorem does not guarantee that if at least one pair of squares meets in an edge then whether $G$ is quasi-chordal or not? To answer this question, we further study sc weakly chordal graphs for above case and obtain the following result, i.e. when at least one pair of squares meets in an edge.
**Theorem 3.7.** Let $G$ be a sc graph, such that at least one pair of squares meets in an edge then $G$ contains one or more of the following graphs as induced subgraphs.

![Graphs](image)

**Proof.** Let $G$ be sc graph, suppose there exists at least one pair of squares which meets in an edge. Now whenever two squares meet in an edge then the graph formed in this way will always have six vertices only, however they may have seven vertices but in this case the graph has no edge common as shown below in figure-3.3, so we do not consider this graph.

![Figure-3.3](image)

Now consider a six cycle graph with vertices $v_1, v_2, v_3, v_4, v_5, v_6$ with edges $v_1v_2, v_2v_3, v_3v_4, v_4v_5, v_5v_6, v_6v_1$. We add an edge between the vertices $v_2$ and $v_5$ as $v_2v_5$. This obtained graph is shown below in figure-3.4 as graph $D_1$. The graph $D_1$ has clearly two squares as $\{v_1, v_2, v_5, v_6\}, \{v_2, v_3, v_4, v_5\}$ and they meet on edge $v_2v_5$. 

65
If we add edges on graph $D_1$ one by one between the non-adjacent pair of vertices then we get following graphs $D_2$, $D_3$, $D_4$, $D_5$, $D_6$, $D_7$, $D_8$, and $D_9$ as follows:

The graphs $D_2$ and $D_3$ as shown below in figure-3.5 are obtained by just adding a single edge on graph $D_1$ between the vertices $v_1, v_4$ and $v_1, v_3$ respectively.

The graphs $D_4$, $D_5$, and $D_6$ as shown below in figure-3.6 are obtained by adding edges on $D_1$ as $(v_1, v_4), (v_3, v_6)$ for $D_4$, edges $(v_1, v_4), (v_1, v_3)$ for $D_5$ and edges $(v_1, v_3), (v_4, v_6)$ for $D_6$.

Similarly graphs $D_7$, $D_8$, and $D_9$ as shown in figure-3.7, can be obtained from $D_1$ by adding the edges as follows; $(v_1, v_4), (v_1, v_3), (v_3, v_6)$ for $D_7$, $(v_1, v_4), (v_1, v_3)$ $(v_4, v_6)$ for $D_8$ and $(v_1, v_4), (v_1, v_3), (v_3, v_6), (v_4, v_6)$ for $D_9$. 

![Figure-3.4](image)

![Figure-3.5](image)

![Figure-3.6](image)

![Figure-3.7](image)
Clearly graphs $D_1$ to $D_6$ are the only possible graphs on six vertices such that two squares meet in an edge. Hence the Theorem. □

We note that the graph $D_3$ given in figure-3.5(b) contains an induced cycle $C_5$: $v_1, v_3, v_4, v_5, v_6$ and $v_1$ and the graph $D_6$ is the complement of induced cycle $C_6$ as given in figure-3.6(c). This observation shows that both the graphs $D_3$ and $D_6$ are not weakly chordal, so we have the following result.

**Corollary 3.8.** Let $G$ be a sc weakly chordal graph such that at least one pair of squares meet in an edge then $G$ does not contain the following graphs as induced subgraphs.

\[ D_3 \quad \text{and} \quad D_6 \]

### 3.3.1 Recognition of sc quasi-chordal graphs

Many graph classes are defined or characterized in terms of an elimination scheme. For example chordal graphs are defined as having no induced cycles of length greater than 3. They are characterized as those graphs which have an elimination scheme with the property that neighbors of $v_i$ induce a clique. Trees can be characterized as those graphs such that every eliminated vertex (except for $v_n$) has degree 1 in the remaining graph. Similarly for quasi-chordal graph there also exists an elimination scheme, which is ensured by the following result given by Voloshin [146].
Theorem 3.9 [146]. Let $G$ be a graph on $n$ vertices. Then $G$ is quasi-chordal if and only if each of its induced subgraph contains a simplicial or co-simplicial vertex.

Based on the above result, recognition of quasi-chordal was first studied by Voloshin in [146] and he proposed an $O(n^3)$ time algorithm. Later Spinrad [128] proposed an $O(n^{2.77})$ time algorithm. Hoàng [78] also independently proposed an $O(nm)$ time algorithm. Much recently Gorgos et al. [57] also proposed an $O(nm)$ time algorithm for recognizing quasi-chordal graphs.

Quasi-chordal graph may admit many different elimination schemes i.e. if it has eliminated scheme only on the basis of simplicial vertices then the graph is chordal and if the eliminated vertices are only co-simplicial then the graph is co-chordal. Now if the vertices of quasi-chordal graphs are eliminated by first removing simplicial and then co-simplicial vertices then the graph is called semi-chordal. The following lemma shows the connection between the simplicial vertices and no-mid vertices as well as co-simplicial vertices and no-end vertices.

Lemma 3.10. Every simplicial vertex is no-mid vertex and every co-simplicial vertex is no-end vertex. Converse need not to be true.

Proof. Since simplicial vertex cannot be middle vertex of any induced $P_3$ and every induced $P_4$ always contains two, $P_3$ as a induced subgraphs. This implies that the middle vertices of both $P_3$ are always mid vertices of $P_4$, therefore simplicial vertices cannot be middle vertex of any $P_4$. Hence they are always
no-mid vertex. Now suppose \( x \) is any end vertex of an induced \( P_4 \) then the non-neighbors of \( x \) will never form stable set as one of its non-neighbors is an end-vertex and the other is mid-vertex, therefore end vertices of any \( P_4 \) cannot be co-simplicial. Hence co-simplicial vertex is always no-end vertex.

Now for converse consider a chordless cycle of length 4 (i.e. \( C_4 \)), then each vertex of this chordless \( C_4 \) is no-mid but not simplicial. Similarly in the complement of this \( C_4 \) (i.e. \( 2K_2 \)), every vertex is no-end but not co-simplicial.

Hence the result. \( \square \)

The following result shows the relation between simplicial and co-simplicial vertices in sc graphs.

**Theorem 3.11.** Let \( G \) be a sc graph, if there exists any simplicial vertex in \( G \) then there also exists a co-simplicial vertex, converse is also true.

**Proof.** Let \( G \) be a sc graph, suppose a vertex \( v \) be a simplicial in \( G \), then by definition of simplicial vertex, all the neighborhood vertices of \( v \) are adjacent to each other. Now, in the complement of the \( G \), same vertex \( v \) and adjacent vertices make an independent set of vertices i.e. \( v \) becomes co-simplicial in \( \overline{G} \).

Since \( G \) and \( \overline{G} \) are isomorphic, thus in \( G \), there also exist a co-simplicial vertex. The same argument also holds for the converse. \( \square \)

The following corollary is immediate from the above Theorem.

**Corollary 3.12.** Let \( G \) be a sc graph, if there exist no simplicial vertex in \( G \) then there exist no co-simplicial vertex, converse is also true.
For recognizing whether a sc graph is quasi-chordal graph, we use a
different method as compare to algorithms discussed in [128], [57]. The basic
difference between the algorithms in [128], [57] and the one given here is that
we find the simplicial and co-simplicial vertices within the set of no-mid and
no-end vertices while the other algorithms find simplicial and co-simplicial
vertices in whole graph. So to decide whether a vertex is simplicial or co-
simplicial or not first we present an algorithm-3.3 for simplicial and co-
simplicial vertices. Further algorithm-3.3 is used as subroutine in algorithm-
3.4. The algorithm-3.3 is as follows.

**Algorithm 3.3: An Algorithm for recognizing simplicial and co-simplicial
vertices.**

*Input:* A graph $G$ and a vertex $u$.

*Output:* Return either simplicial or co-simplicial vertex.

**Step-1:** Compute neighborhood of $u$ i.e. $N(u)$.

**Step-2:** If $N(u)$ induces a complete subgraph of $G$, then

return "‘$u$’ is a simplicial vertex"

Stop.

else
compute its non-neighbors $N'(u)$ i.e. $N'(u) = V(G) - N[u]$

If $N'(u)$ induces a stable set of $G$, then

return "‘$u$’ is co-simplicial vertex"

Stop.

else
return "‘$u$’ is neither simplicial nor co-simplicial vertex”.

**End.**

**Complexity.** The time complexity of the algorithm-3.3 is $O(n^2)$ by [79].
The algorithm-3.4 as given here mainly depends on finding no-mid and no-end vertex set in the input graph $G$. As soon as it computes no-mid and no-end vertex set, algorithm goes for the search of simplicial and co-simplicial vertices within the set of no-mid and no-end vertices. If no-mid and no-end vertex sets do not contain any simplicial and co-simplicial vertices respectively then at the initial stage of algorithm it is possible to get the output i.e. the input graph is not quasi-chordal.

**Algorithm 3.4: An Algorithm for recognizing sc quasi-chordal graph.**

**Input:** A sc graph $G$.

**Output:** "$G$ is sc quasi-chordal graph" or "$G$ is not sc quasi-chordal graph".

---

**Step-1:** Compute set of no-mid, no-end, $R$, (using algorithm-3.1).

and elimination order = $\phi$

**Step-2:** If all the vertices of no-mid and no-end set are simplicial and co-simplicial respectively (using algorithm-3.3).

and no-mid $\cup$ no-end = vertex set of graph, then

print "$G$ is sc quasi-chordal graph"

and print "vertex set of $G$"

Stop.

else

if no-mid and no-end set contains no simplicial and no co-simplicial vertices respectively, then

print "$G$ is not sc quasi-chordal graph"

Stop.

**Step-3:** select either a simplicial vertex from no-mid set or co-simplicial vertex from no-end set (let this vertex be $u$)

remove vertex $'u'$ and put $'u'$ in elimination order.
Step-4: update the graph and goto step-1.

Step-5: If all the vertices are eliminated in this way then

   print “G is sc quasi-chordal graph”

   and print “elimination order set”

Else print “G is not sc quasi-chordal graph”

End.

The time complexity of the algorithm-3.4 is as follows.

**Complexity.** Algorithm-3.4 first uses algorithm 3.1 in step-1, which has time complexity $O(n^3m)$. All the other steps require lesser time. Hence the overall time complexity is $O(n^3m)$.

The correctness of algorithm-3.4 is as follows.

**Theorem 3.13.** Algorithm-3.4 checks whether an input sc graphs is quasi-chordal or not correctly.

**Proof.** Simplicial vertices are no-mid vertices and co-simplicial vertices are no-end vertices. So while running the algorithm-3.4, when we compute set of no-mid and no-end vertices simplicial vertex always lies in no-mid and co-simplicial vertex lies in no-end set. Now from the statements (iii) and (i) of Theorem-3.5, it is clear that if we eliminate all the vertices in such a way then the resulting graph is always a quasi-chordal. Hence the Theorem. □

To illustrate algorithm 3.4 we consider the sc graphs $G_1$ and $G_2$ on 9 vertices as shown in figure-3.8(a) and figure-3.8(b) respectively.

Let the sc graph $G_1$ as shown in figure-3.8(a) be input to the algorithm-3.4.

Step-1 finds no-mid set, no-end set and $R_v$ as follows.

no-mid = \{v_3, v_4\}, no-end = \{v_7, v_8\}, $R_v$ = \{v_1, v_2, v_5, v_6, v_9\},

72
this is done by using algorithm-3.1. Now step-2 of algorithm-3.4 finds that vertices \( v_3, v_4 \) from no-mid set and vertices \( v_7, v_8 \) from no-end set. These are simplicial and co-simplicial vertices respectively (this is done by using algorithm-3.3). Although all the vertices of no-mid and no-end sets are simplicial and co-simplicial respectively, but no-mid \( \cup \) no-end \( \neq \) vertex set of graphs i.e. \( \{v_3, v_4\} \cup \{v_7, v_8\} \neq \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9\} \) therefore algorithm-3.4 proceeds to step-3. In this step let vertex \( v_3 \) be eliminated first. Then step-4 updates the graph and repeats the process again. Eventually algorithm-3.4 successfully eliminates all the vertices of \( G_1 \) one by one and produces the elimination order set \( = \{v_3, v_4, v_5, v_6, v_7, v_8, v_9\} \). Thus algorithm-3.4 decides that the input graph \( G_1 \) is sc quasi-chordal graph. The overall procedure of elimination of vertices can be seen in figure-3.9.

Let graph \( G_2 \) as shown in figure-3.8(b) be the input to algorithm-3.4. The algorithm first finds its no-mid set, no-end set and \( R_v \) as follows.

\[
\text{no-mid} = \{v_1, v_4, v_5, v_6\}, \quad \text{no-end} = \{v_2, v_3, v_7, v_8\} \quad \text{and} \quad R_v = \{v_9\},
\]
this is done by using algorithm-3.1. In step-2 the algorithm-3.4 does not find any simplicial vertex in no-mid set and no co-simplicial vertex in no-end set, therefore the algorithm terminates and decides that the input graph $G_2$ is not sc quasi-chordal graph.

Figure-3.9
3.3.2 Catalogue compilation of sc quasi-chordal graphs

Using algorithm 3.4 we compile the catalogue of sc quasi-chordal graphs with at most 17 vertices from the available catalogue of sc graphs with at most 17 vertices.

<table>
<thead>
<tr>
<th>Number of vertices n</th>
<th>4</th>
<th>5</th>
<th>8</th>
<th>9</th>
<th>12</th>
<th>13</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of sc graphs</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td>36</td>
<td>720</td>
<td>5600</td>
<td>703760</td>
<td>11220000</td>
</tr>
<tr>
<td>Number of sc quasi-chordal graphs</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>62</td>
<td>62</td>
<td>2406</td>
<td>2406</td>
</tr>
</tbody>
</table>

Table 3.2

3.4 $P_5$-free weakly chordal and $P_5$-free sc weakly chordal graphs

Weakly chordal graphs are not necessarily perfectly orderable [69], however many known classes of perfectly orderable graphs are weakly chordal (such as chordal, quasi-chordal and brittle). In [77], Hoàng showed that determining whether a graph is perfectly orderable remains NP-complete for the class of weakly chordal graphs. In 1990 Chvátal conjectured [23] that every $P_5$-free weakly chordal graph is perfectly orderable. Hayward [69] proved this conjecture by presenting a polynomial time algorithm to find a perfect order of any $P_5$-free weakly chordal graphs.

**Theorem 3.15 [69]:** $P_5$-free weakly chordal graphs are perfectly orderable.

On the other hand, it is known that $\bar{P}_3$-free weakly chordal graphs are not necessarily perfectly orderable [69]. In this section we show that $\bar{P}_3$-free sc
weakly chordal graphs are perfectly orderable. To prove this we need the following result.

**Theorem 3.16.** Let $G$ be a sc graph. Then $G$ is $P_5$-free if and only if it is $\overline{P}_5$-free.

**Proof.** Let $G$ be a sc graph. Suppose $G$ contains $P_5$ with vertices $v_1, v_2, v_3, v_4$ and $v_5$. Then these vertices $v_1, v_2, v_3, v_4, v_5$ induces a house ($P_5$) in $G$ as shown below in figure-3.10

Since $G$ is sc, so if $G$ contains $P_5$ it also contains $\overline{P}_5$, therefore if $G$ is $P_5$-free then $G$ is $\overline{P}_5$-free. Conversely, suppose $G$ contain $\overline{P}_5$ with vertices $v_1, v_2, v_3, v_4, v_5$, then in $\overline{G}$ these vertices become a chordless path, $P_5$ as shown above in figure-3.10. Hence $G$ is $\overline{P}_5$-free implies $G$ is $P_5$-free. Hence the Theorem. □

Using above Theorem, we get the following result.

**Theorem 3.17.** $\overline{P}_5$-free sc weakly chordal graphs are perfectly orderable.

**Proof.** By Theorem-3.16, it is clear that in the case of sc weakly chordal graphs $P_5$-free sc weakly chordal graph class and $\overline{P}_5$-free sc weakly chordal graph class are exactly same i.e. whenever sc weakly chordal graph is $P_5$-free then it is also $\overline{P}_5$-free, and vice-versa. So by Theorem-3.15 and Theorem-3.16,
\(P_5\)-free sc weakly chordal graphs are perfectly orderable. Hence the

Theorem. \(\square\)

While studying \(P_5\)-free sc weakly chordal graphs another graph class

known as charming graph [76] has almost same structure as of \(P_5\)-free sc

weakly chordal graph.

A vertex \(v\) in a graph \(G\) is said to be charming if \(v\) is not end vertex in a \(P_5\) in

\(G\), is not end vertex in a \(P_5\) in \(\overline{G}\) and does not lie on a \(C_5\) in \(G\). A graph is

charming in which every induced subgraph has a charming vertex [76]. In the

same paper Hoàng proved that every charming graph is perfectly orderable,

moreover he showed the following.

**Theorem 3.18 [76].** Every weakly chordal graph with no induced \(P_5\) and \(\overline{P_6}\) is

charming.

Now for sc weakly chordal graphs we have following Corollary.

**Corollary 3.19.** Every sc weakly chordal graph with no \(P_5\) and \(P_6\) is

Charming.

**Proof.** Since sc weakly chordal graph is isomorphic to its complement.

Therefore the condition on \(\overline{P_6}\) in the Theorem-3.18 is reduced to \(P_6\) for sc

weakly chordal graph. Hence the result. \(\square\)

We know that if \(G\) is \(P_5\)-free then \(G\) is \(P_6\)-free. Thus we have the

following result.

**Corollary 3.20.** Every \(P_5\) free sc weakly chordal graph is Charming.

**Proof.** Follows from Corollary-3.19. \(\square\)
Now it is clear from Corollary-3.20 that if we recognize $P_5$-free sc weakly chordal graphs then these graphs are also charming.

### 3.4.1 Recognition of $P_5$-free sc weakly chordal graph

In [69] Hayward proposed an $O(n^3)$ time algorithm for recognizing a $P_5$-free weakly chordal graphs. In fact he used complex concepts like separating sets and handles for their algorithm, which is not easy to implement. Recently Nikolopous and Paliou [96] also presented parallel algorithms for recognizing $P_5$-free and $\overline{P}_5$-free weakly chordal graphs. Their algorithms are based on one of the following result.

**Theorem 3.21 [96].** Let $G$ be a weakly chordal graph with vertex set $V(G)$. Then $G$ is $\overline{P}_5$-free weakly chordal if and only if none of its edges is a $\overline{P}_5$-witness.

An edge $e = (a,b)$ is $\overline{P}_5$-witness in a weakly chordal graph if there exist a vertex $u \in V(G) - N[e]$ such that $N(u)$ contains vertices from both $A(a,e)$ and $A(b,e)$, (where $A(a,e) = N(a) - N[b]$, $A(b,e) = N(b) - N[a]$ and $N[e] = N[a] \cap N[b]$).

We use method similar to [96] for recognizing $P_5$-free sc weakly chordal graphs. Since $P_5$-free sc weakly chordal graph class and $\overline{P}_5$-free sc weakly chordal graph class are same, so we can use above result for recognizing $P_5$-free sc weakly chordal graphs.

The following algorithm recognizes $P_5$-free sc weakly chordal graphs as well as sc charming graphs.

78
Chapter 3

On the recognition of some classes of sc perfectly orderable graphs

Algorithm 3.5: An Algorithm for recognizing $P_5$-free sc weakly chordal graph.

Input: A sc weakly chordal graph $G$.

Output: $G$ is $P_5$-free sc weakly chordal graph or $G$ is not $P_5$-free sc weakly chordal graph.

Step-1: Select any arbitrary unmarked edge, say $e = (a, b)$

If no unmarked edge remains

Print "$P_5$-free sc weakly chordal graph"

Stop.

Step-2: Find $N(a)$, $N(b)$, $N[a]$ and $N[b]$

Step-3: Compute

$A(a,e) = N(a) - N[b]$, $A(b,e) = N(b) - N[a]$, $N[e] = N[a] - N[b]$ and $R_V = V(G) - N[e]$.

Step-4: for every vertex $u \in R_V$

if $N(u)$ contains vertices from both $A(a,e)$ & $A(b,e)$, then

Print "$G$ is not $P_5$-free sc weakly chordal graph".

Stop.

else mark the edge $e$.

goto step-1

End.

Now, we compute the time complexity of the algorithm-3.5. The graph $G$ is assumed to be given in its adjacency list representation.

**Complexity.** We compute the complexity of each step separately. In step-1, since we have to analyze each edge as worst case, so it takes $O(m)$-time. Step-2 is executed for each vertex of edge $e$, taking maximum $(n-1)$ executions for each vertex so this step takes $O(n-1) = O(n)$ time. Step-3 takes obviously
constant time. Now the step-4 takes $O(n-1) = O(n)$ time to be executed as worst case. Taking into consideration the time complexity of each step of algorithm, we have total complexity $O(mn)$.

**Theorem 3.22.** Algorithm-3.5 checks whether an input sc weakly chordal graphs is $P_3$-free sc weakly chordal or not correctly.

**Proof.** The correctness of algorithm follows from Theorem-3.21. □

To illustrate algorithm-3.5, we consider the following sc weakly chordal graphs $G_1$ and $G_2$ as shown in figure-3.11(a) and figure-3.11(b) respectively.

![Figure-3.11](image)

Let us input graph $G_1$ in algorithm-3.5 as shown in figure-3.11(a). Step-1 selects an edge $e = (v_1, v_2)$ arbitrarily. Step-2 computes $N(v_1) = \{v_3, v_6, v_7, v_8\}$, $N(v_2) = \{v_1, v_2, v_3, v_4\}$, $N[v_3] = \{v_3, v_5, v_6, v_7, v_8\}$ and $N[v_7] = \{v_1, v_2, v_3, v_4, v_7\}$. Then step-3 computes $A(v_3, e) = N(v_3) - N[v_7] = \{v_5, v_6, v_8\}$, $A(v_7, e) = N(v_7) - N[v_3] = \{v_1, v_2, v_4\}$, $N[e] = N[v_3] \cap N[v_7] = \{v_3, v_7\}$ and $R_e = V(G) - N[e] = \{v_1, v_2, v_4, v_5, v_6, v_8\}$.
Now step-4 checks for all the vertices of $R_v$, i.e. let $u = v_1$ then $N(v_1) = \{v_4, v_7, v_8\}$. Now it is clear that $N(v_1)$ contains a vertex $v_7$ which does not lie either in $A(v_1, e)$ or in $A(v_7, e)$. Similarly $N(v_2)$, $N(v_4)$, $N(v_5)$, $N(v_6)$ and $N(v_8)$ contain vertices $v_7, v_7, v_3, v_3, v_3$ respectively which are neither in $A(v_3, e)$ nor in $A(v_7, e)$. Hence edge $e = (v_3, v_7)$ is not $P_5$-witness and edge $e = (v_3, v_7)$ is now marked edge. Now algorithm-3.5 checks for every other edge of $G_1$ and finally finds no edge as $P_5$-witness. Hence algorithm-3.5 decides that the input sc weakly chordal graph $G_1$ is $P_5$-free sc weakly chordal graph.

Now consider the other sc weakly chordal graph $G_2$ as shown in figure-3.11(b). Step-1 of algorithm-3.5, selects an edge $e = (v_2, v_3)$ arbitrarily then step-2 computes $N(v_2) = \{v_5, v_6, v_7\}$, $N(v_3) = \{v_1, v_2, v_6\}$, $N[v_2] = \{v_2, v_5, v_6, v_7\}$ and $N[v_3] = \{v_1, v_2, v_3, v_8\}$. Then step-3 computes $A(v_2, e) = N(v_2) - N[v_3] = \{v_6, v_7\}$, $A(v_3, e) = N(v_3) - N[v_2] = \{v_1, v_8\}$, $N[e] = N[v_2] \cap N[v_3] = \{v_2, v_3\}$ and $R_v = V(G) - N[e] = \{v_1, v_3, v_4, v_6, v_7, v_8\}$. Now step-4 checks i.e. let $u = v_1$ then $N(v_1) = \{v_4, v_5, v_7, v_8\}$. Now it is clear that $N(v_1)$ contains vertices $v_7$ & $v_8$ which lie in $A(v_2, e)$ and $A(v_3, e)$ respectively. Hence step-4 decides that the edge $e = (v_2, v_3)$ is $P_5$-witness, therefore the algorithm-3.5 decides that the input sc weakly chordal graph $G_2$ is not $P_5$-free sc weakly chordal graph.

### 3.4.2 Catalogue compilation of $P_5$-free sc weakly chordal graphs

We compile the catalogue of $P_5$-free sc weakly chordal graphs with at most 17 vertices from the available catalogue of sc graphs with at most 17
vertices. Moreover all the graphs which are $P_5$-free sc weakly chordal graph are also sc charming.

<table>
<thead>
<tr>
<th>$Number$ of $vertices$ $n$</th>
<th>4</th>
<th>5</th>
<th>8</th>
<th>9</th>
<th>12</th>
<th>13</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Number$ of sc graphs</td>
<td>1</td>
<td>2</td>
<td>10</td>
<td>36</td>
<td>720</td>
<td>5600</td>
<td>703760</td>
<td>11220000</td>
</tr>
<tr>
<td>$Number$ of $P_5$-free sc weakly chordal graphs</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>23</td>
<td>23</td>
<td>275</td>
<td>275</td>
</tr>
</tbody>
</table>

Table-3.3