Intractable Graph Optimization Problems

(4541.554 Introduction to Computer-Aided Design)

School of EECS

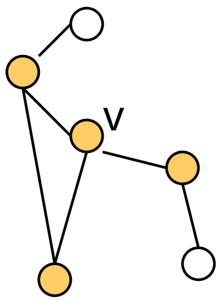
Seoul National University

Intractability

- **Problem**
  - **Optimization version**
    - **Find the optimal feasible solution**
  - **Evaluation version**
    - **Find the cost of the optimal solution**
    - **Not harder than solving the optimization version**
  - **Recognition version**
    - **Is there a feasible solution $f \in F$ such that $c(f) \leq L$**
    - **Evaluate and compare the cost with L**
    - **Not harder than solving the evaluation version**
  - **Consider recognition version only**
    - **No polynomial time algorithm for recognition version --> no polynomial time algorithm for optimization version**
  - **Recognition version --> Evaluation version**
    - **Binary search**
    - **Assume c(f) is an integer and logc(f) is bounded by a polynomial in the size of the input**
  - **Evaluation version --> Optimization version**
    - **No known general method**

- **Example:**

procedure MaxClique(G) -- Returns the largest clique of G
if G has no nodes then return null
else
begin
let v be a node such that
CliqueSize(G(v))=CliqueSize(G),
where G(v) is the subgraph of G consisting of v and all of its adjacent nodes;
return {v} ∪ MaxClique(G(v)-v);
end



Complexity of CliqueSize: C(n)

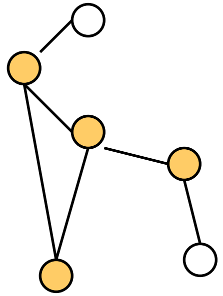
--> Complexity of MaxClique: $T(n) \le (n+1)C(n)+O(n)+T(n-1)$

$$= (n+1)C(n)+O(n)+nC(n-1)+O(n-1)+\ldots$$

$$\le n(n+1)C(n)+nO(n) = O(n^2C(n))$$

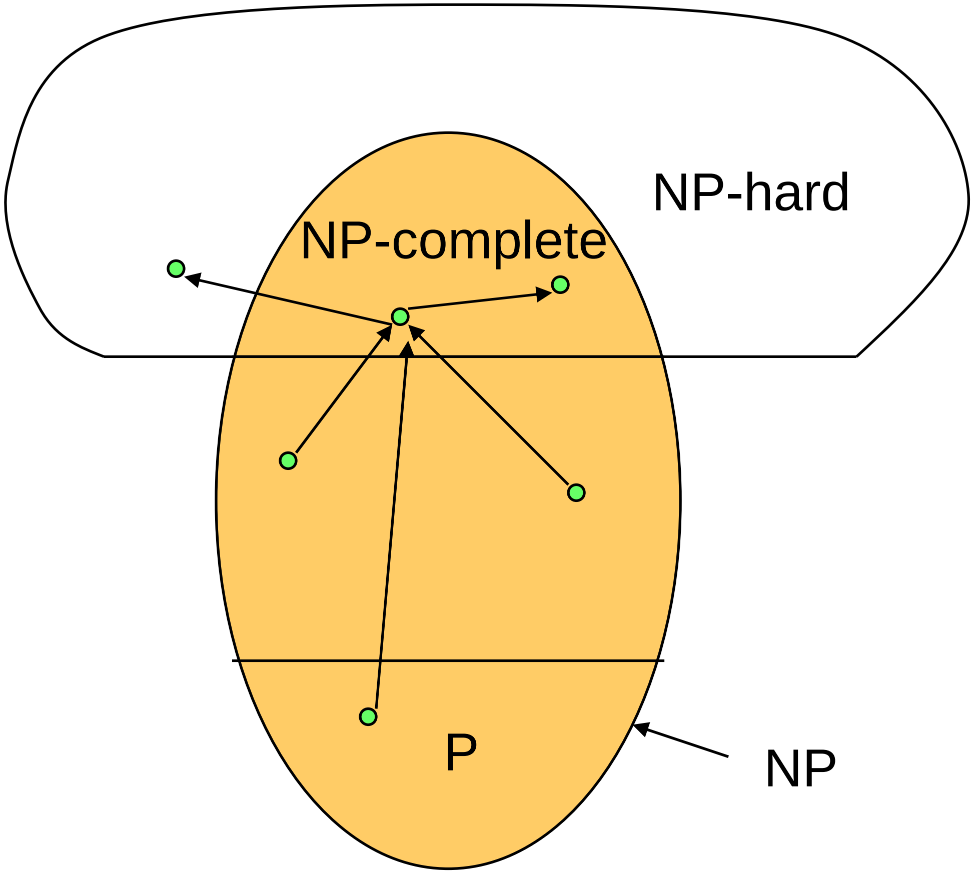
$$T(n) = O(n^2C(n))$$

- **Definition of P and NP**
  - **P**
    - **Class of problems which can be solved in polynomial time by a deterministic machine**
  - **NP**
    - **Class of problems which can be solved in polynomial time by a nondeterministic machine**
    - **With input x$c(x), where x is a yes instance, c(x) is the certificate, and $ marks the end of the input, there exists a certificate checking algorithm that reaches the answer 'yes' after at most polynomial steps**
- **Example of an NP problem**
  - **Recognition version of the Maximum Clique problem: Given a graph G(V, E), is there a clique of size K?**
  - **x: graph G and integer K**
    c(x): a set of vertices C, $|C|=K \leq p(|x|)$
  - **Checking whether there is an edge (u, v) in G for all u, v $\in$ C takes $O(n^2)$ steps**



- **Polynomial-Time Transformation**
  - **A recognition problem A_1 polynomially transforms to another recognition problem A_2, if given x (any instance of A_1), we can construct y (an instance of A_2) within polynomial time, such that x is a yes instance of A_1 if and only if y is a yes instance of A_2**
- **Definition of NP-Completeness**
  - **A recognition problem A $\in$ NP is said to be NP-complete if all other problems in NP polynomially transform to A**
- **Proof of NP-Completeness**
  - **Prove that the given problem is in NP --- (1)**
  - **Then prove that all other problems in NP polynomially transform to the given problem**

    Or prove that a known NP-complete problem is polynomially transformable to the given problem--- (2)
  - **A problem that satisfies (2) is said to be in NP-hard**



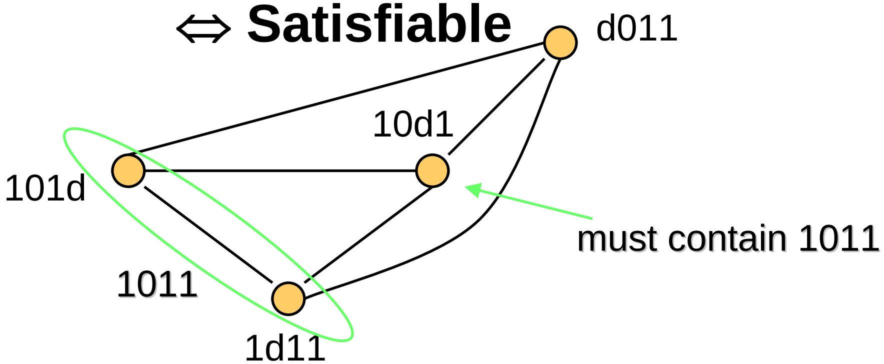
S. A. Cook proved that the Satisfiability problem is NP-complete

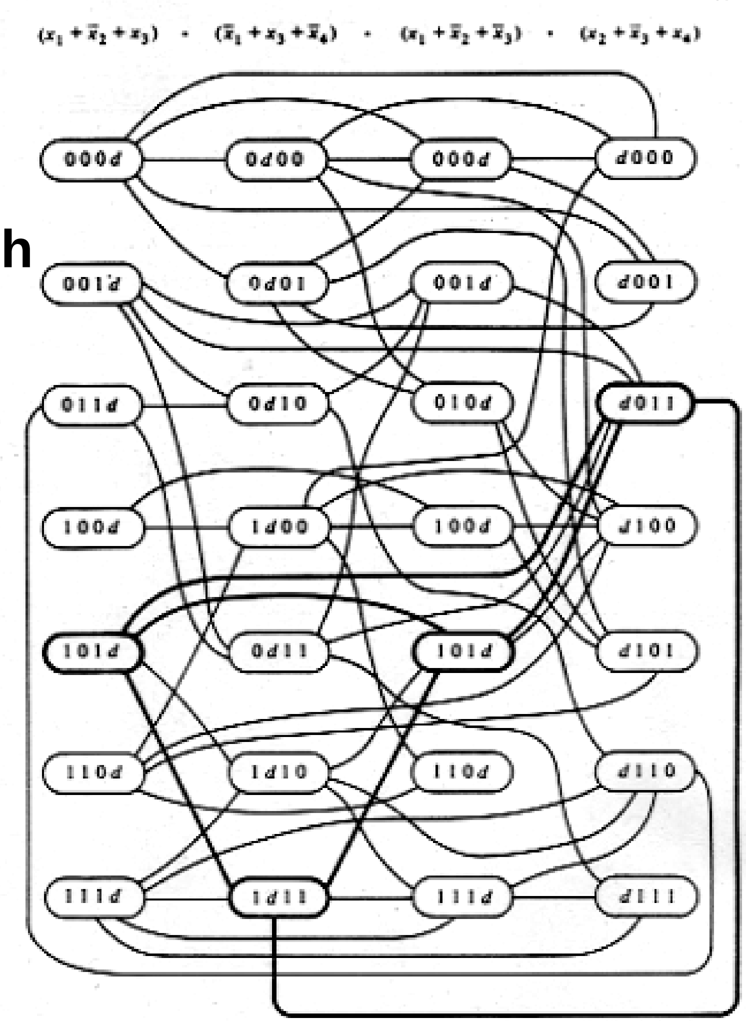
(x1 + x2’ + x3) (x1’ + x2' + x3)(x2)(x3') ...

- **Example**
  - **Prove that the Clique problem is NP-complete**
  - **Clique problem : Given a graph G(V, E), is there a clique of size K?**
  - **Clique problem is in NP**
    - **previously proven**
  - **Let's polynomially transform the 3-Satisfiability problem which is known to be NP-complete to the Clique problem**

    There is a clique of size K

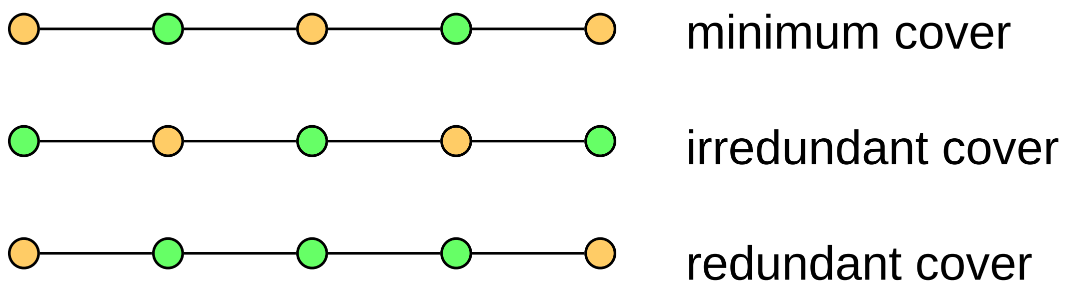
    ⇔ Satisfiable



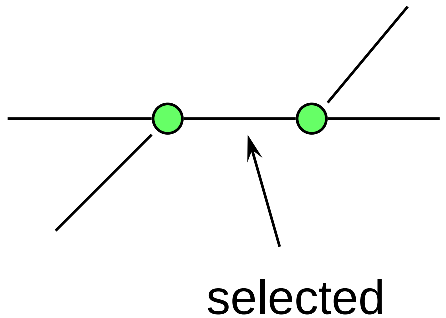


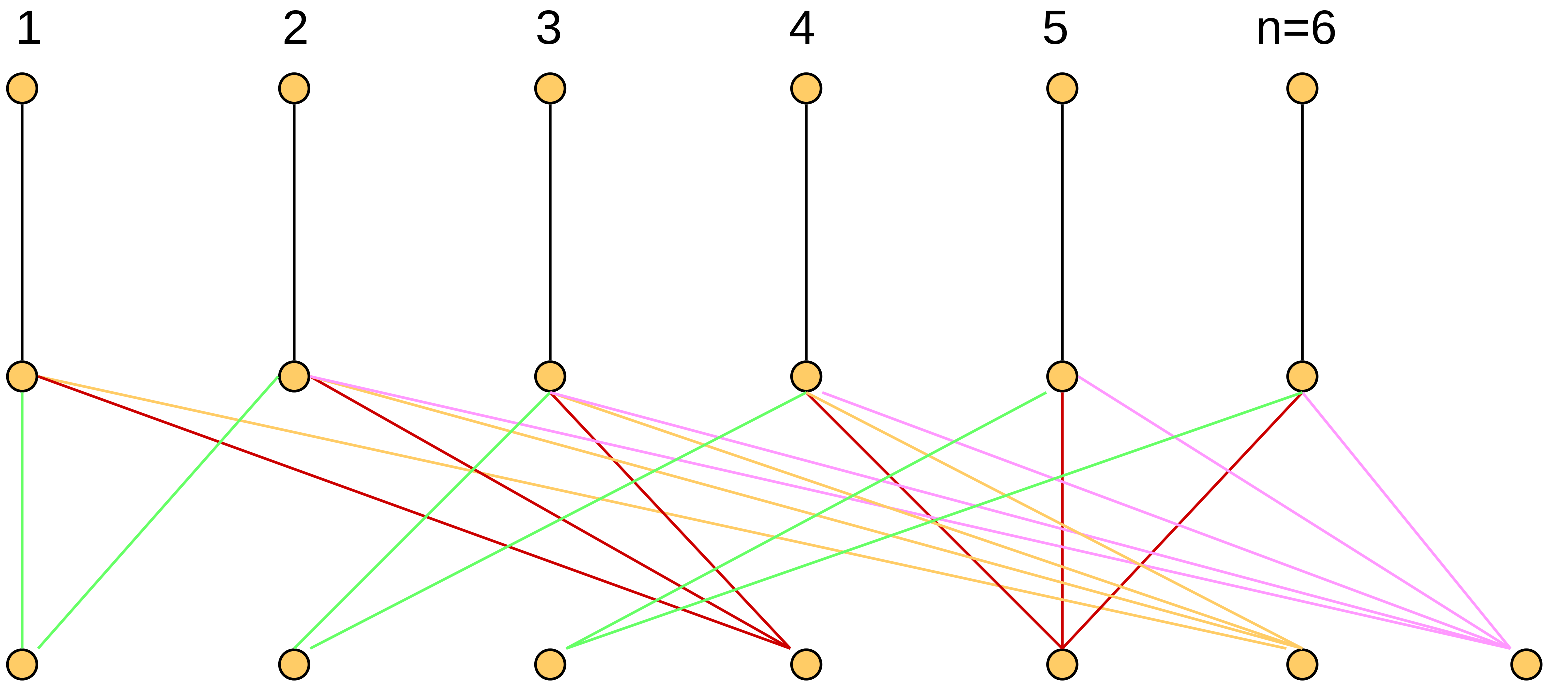
Vertex Cover

- **Intractable (NP-complete)**
- **Greedy algorithm**
  - **select vertices with largest degree (possibly exceeds twice the minimum)**



  - **select edges (at most twice the minimum)**





n	6	10	30	100
|vertex cover|/n	2.17	2.6	3.67	4.8

$$|\text{vertex cover}| = n + \sum_{j=2}^{n-1} \lfloor n/j \rfloor$$

Graph Coloring

- **Intractable**
- **VERTEX_COLOR (G(V, E)) {**

    for (i = 1 to |V|) {

        c = 1;

        while (a vertex adjacent to v_i has color c) do {

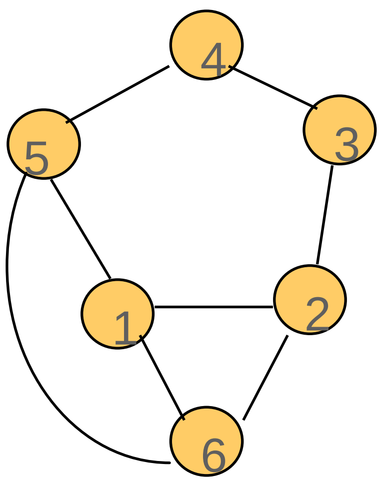
            c = c + 1;

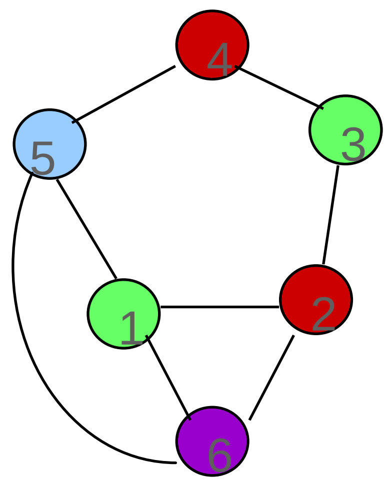
        }

        label v_i with color c;

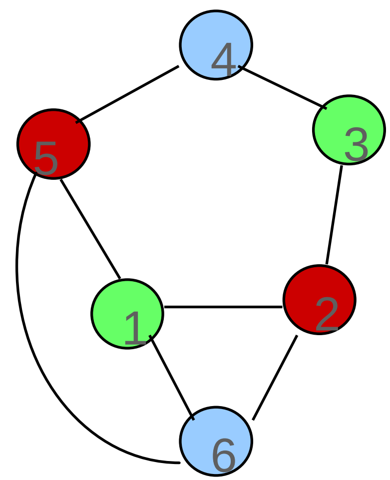
    }

  }





greedy

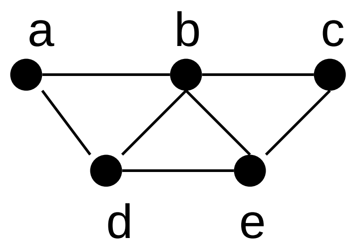


backtrack

– **Chordal graph --> has perfect vertex elimination scheme**

--> coloring in O(|V| + |E|) time

– **Perfect vertex elimination scheme : An ordering of vertices [v_1, v_2, ..., v_n] such that {$v_j \in$ Adj(v_i) | j > i} is complete**

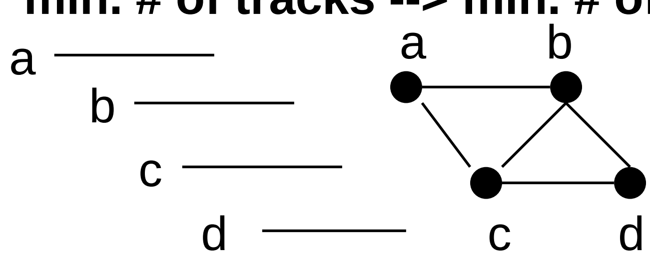


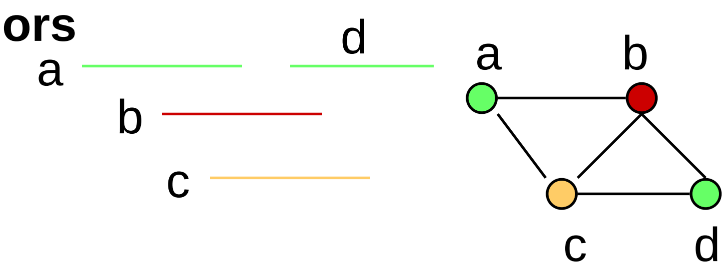
[a, d, b, e, c]

– **Interval graph --> chordal --> use perfect vertex elimination scheme**

– **When a problem is specified by a set of intervals --> use left edge algorithm --> O(|V|log|V|)**

- **intervals in a row --> no intersection --> no edge --> same color**
- **min. # of tracks --> min. # of colors**





– **Left edge algorithm**

```
LEFT_EDGE(L) {
    Sort elements in a list L in ascending order of l_i;
            /* l_i=coordinate of left edge of element i */
    Build a heap priority queue containing only root node q such that
        track_number(q)=0 and coordinate of rightmost edge(q)=0;
            /* priority queue containing nodes, one for each track */
    n=1;                            /* initialize max # of tracks */
    while (L is not empty) do {
        s=First element in L ;
        if (l_s ≥ r_min) {          /* r_min=coordinate of root in the queue */
            assign s to track of the root;
            update priority queue;
        }
        else {
            n=n+1;                  /* add a new track */
            assign s to track n;
            update priority queue with a new node;
        }
    }
}
```

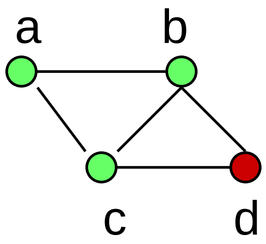
- **Complexity:**
  - **Sorting: O(|V|log|V|)**
  - **If the elements sorted are in a limited range, we can use linear time sorting techniques such as radix sort (may be less efficient due to high constant).**

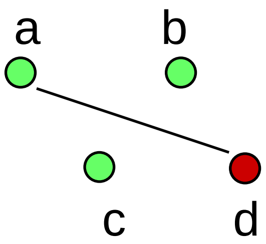
    --> complexity becomes O(|V|log(d)), where d is the density

    --> can be reduced further down to linear time
- **Proof of optimality**
  - **Assuming density d, lower bound of the number of tracks required is d.**
  - **So it is sufficient to prove that the left edge algorithm always places all intervals on d tracks, which is optimum.**
  - **Now, assume the left edge algorithm does not give the optimum. That is, during the run of the algorithm, an interval cannot be assigned to any of the d tracks. This implies that the density is d+1, which is a contradiction.**

Clique Partitioning

- **Intractable**
- **Clique partitioning <---> coloring the complement**
  - **clique <--> independent set in the complement**
  - **find max. clique <--> find max. independent set in the complement**
  - **vertices not in a vertex cover --> no edge between any pair of these vertices --> independent set**





- **Can be solved in O(|V|+|E|) for chordal graph using perfect vertex elimination scheme**
- **Can be solved in polynomial time for comparability graph by transforming it into a minimum network flow problem**