FUNDAMENTAL PROBLEMS AND ALGORITHMS

Graph Theory and Combinational

© *Giovanni De Micheli*
Stanford University

Shortest/Longest path problem

- ***Single-source shortest path problem***.
- Model:
  - Directed graph **G(V, E)** with **N** vertices.
  - Weights on each edge.
  - A source vertex.
- ***Single-source shortest path problem.***
  - Find shortest path from the source to any vertex.
  - Inconsistent problem:
    - Negative-weighted cycles.

Shortest path problem

Bellman's equations: **G(V, E)** with **N** vertices

$$- \; s_j = \min_{k \neq j} (s_k + w_{kj}); \quad j = 1, 2, \ldots, N$$

- Acyclic graphs:
  - Topological sort **O(N [2]).**
  - $$s_j = \min_{k < j} (s_k + w_{kj}); \quad j = 1, 2, \ldots, N$$
- All positive weights:
  - Dijkstra's algorithm.

Dijkstra's algorithm

$G(V, E)$ with **N** vertices

DIJKSTRA(G(V, E, W))

Apply to Korea's map, robot tour, etc

```
{
    s_0 = 0;
    for (i = 1 to N)
        s_i = w_{0,i} ,
    repeat {
        select unmarked v_q such that s_q is minimal;
        mark v_q ;
        foreach (unmarked vertex v_i )
            s_i = min {s_i , (s_q + w_{q,i} )},
    }
    until (all vertices are marked)
}
```

In notation: $s_0 = 0$; **for** ($i = 1$ to N) $s_i = w_{0,i}$, **repeat** { select unmarked v_q such that s_q is minimal; mark v_q; **foreach** (unmarked vertex v_i) $s_i = \min\{s_i, (s_q + w_{q,i})\}$, } **until** (all vertices are marked)

Bellman-Ford's algorithm

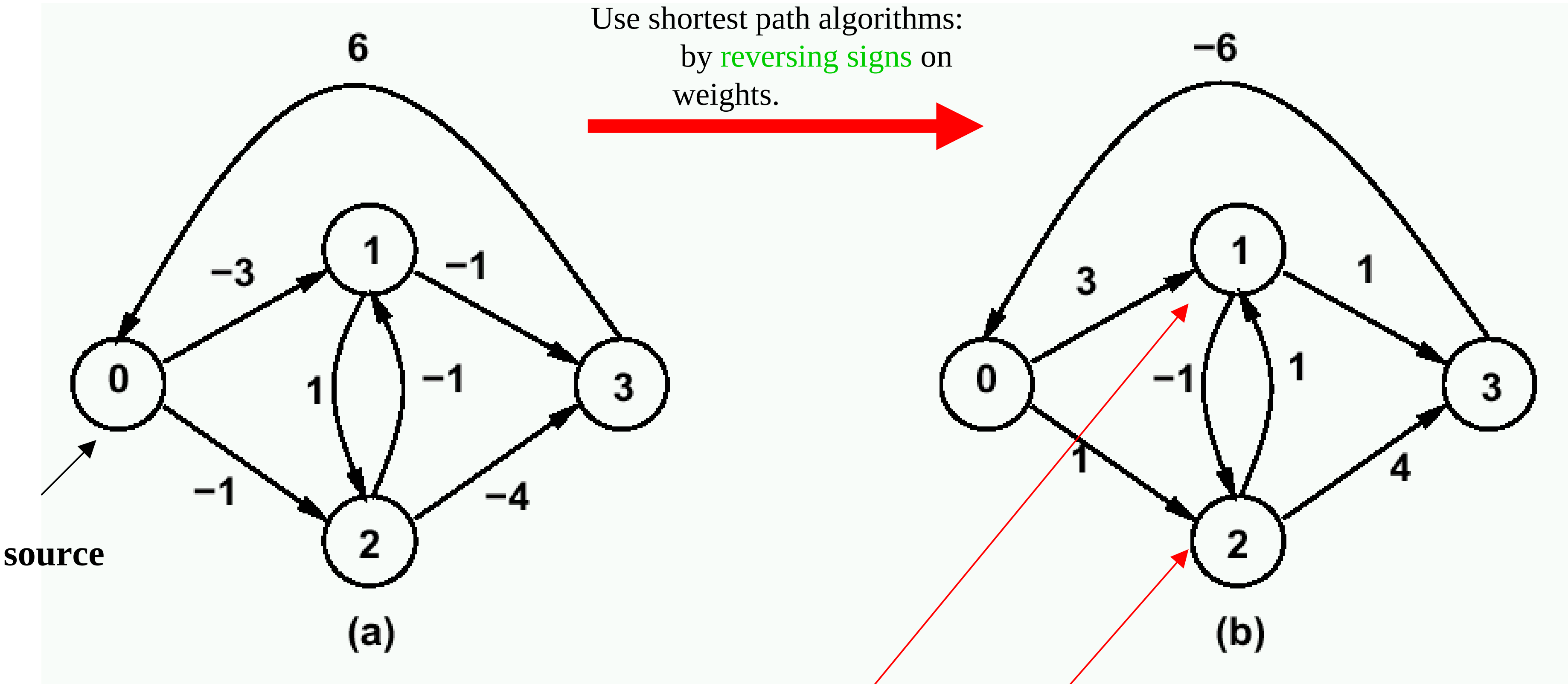
```
BELLMAN_FORD(G(V, E, W))
{
        s^1_0 = 0;
        for (i = 1 to N)
            s^1_i = w_{0,i} ;
        for (j =1 to N){
            for (i =1 to N){
                s^{j+1}_i = min_{k≠i} { s^j_i , (s^j_k + w_{q,i} )},
            }
            if (s^{j+1}_i == s^j_i  ∀i ) return (TRUE) ;
        }
        return (FALSE)
}
```

$s^1_0 = 0$; $s^1_i = w_{0,i}$; $s^{j+1}_i = \min_{k \neq i} \{ s^j_i , (s^j_k + w_{q,i}) \}$; if $(s^{j+1}_i == s^j_i \; \forall i)$ return (TRUE)

Longest path problem

- Use shortest path algorithms:
  - by reversing signs on weights.
- Modify algorithms:
  - by changing min with max.
- Remarks:
  - Dijkstra’s algorithm is not relevant.
  - Inconsistent problem:
    - Positive-weighted cycles.

Example – Bellman-Ford



- Iteration 1: $l_0 = 0$, $l_1 = 3$, $l_2 = 1$, $l_3 = \infty$.
- Iteration 2: $l_0 = 0$, $l_1 = 3$, $l_2 = 2$, $l_3 = 5$.
- Iteration 3: $l_0 = 0$, $l_1 = 3$, $l_2 = 2$, $l_3 = 6$.

Liao-Wong's algorithm

LIAO WONG(G(V, E $\cup$ F, W))

{

for (i = 1 to N)

$l^{1}_{i} = 0$;

for (j = 1 to |F|+ 1) {

foreach vertex v_i

l^{j+1}_{i} = longest path in G(V, E,W_E) ;

flag = TRUE;

foreach edge (v_p, v_q) $\in$ F {

if ($l^{j+1}_{q} < l^{j+1}_{p} + w_{p,q}$){

flag = FALSE;

E = E $\cup$ (v_0 , v_q) with weight ($l^{j+1}_{p} + w_{p,q}$)

}

}

if (flag) **return** (TRUE) ;

}

return (FALSE)

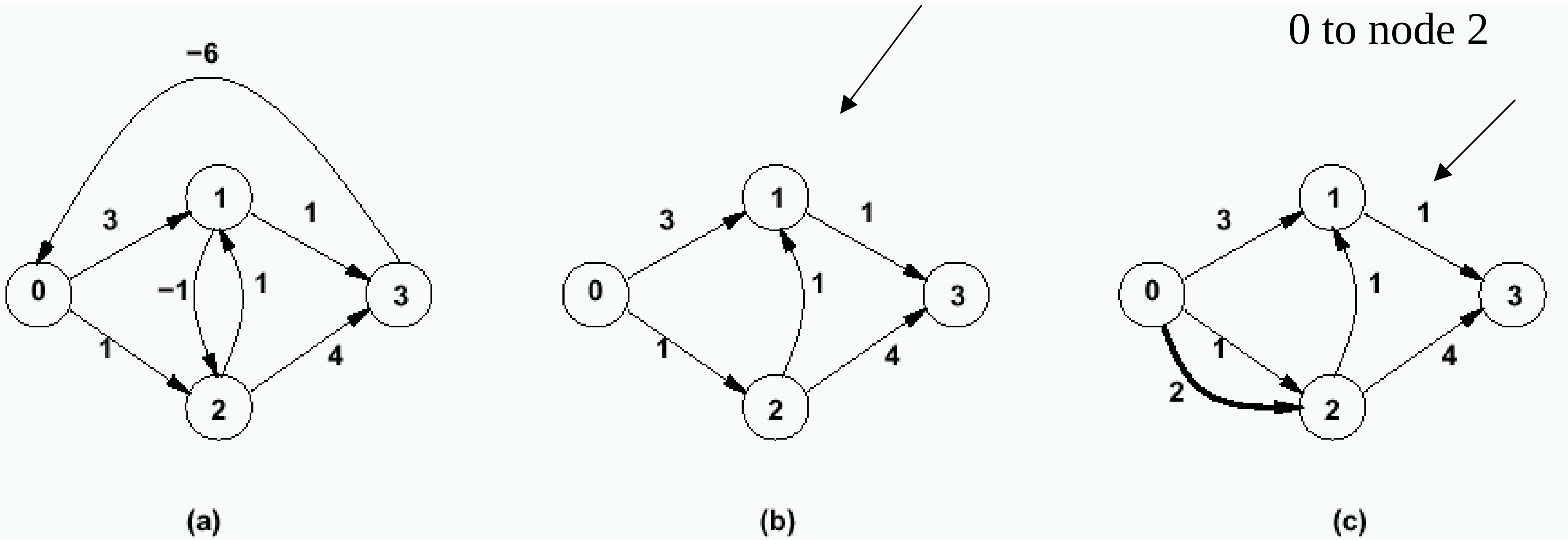
adjust

Example – Liao-Wong

Looking for longest path from node 0 to node 3

Only positive edges from (a)

(b) adjusted by adding longest path from node 0 to node 2

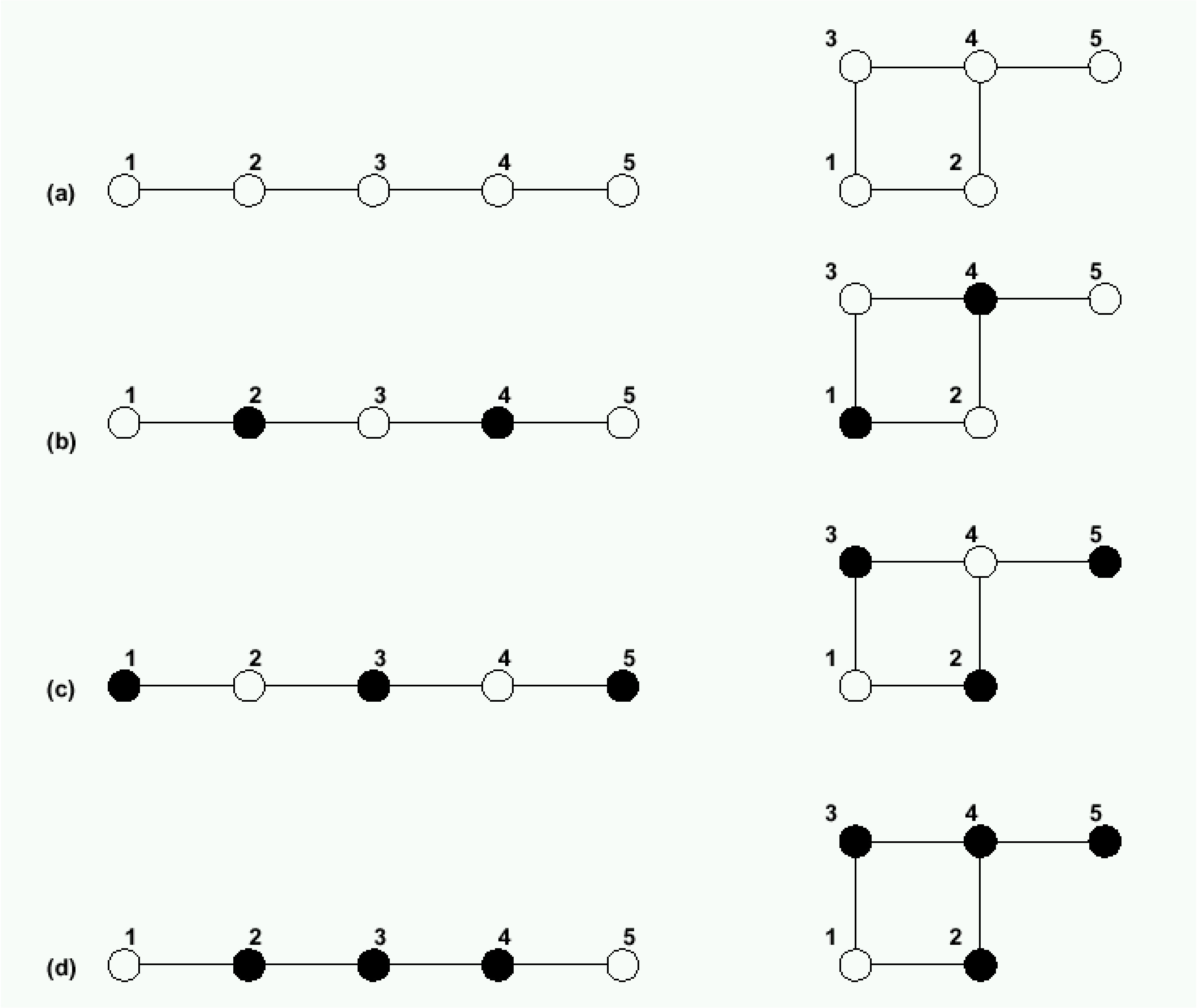


- **Iteration 1:** $l_0 = 0, l_1 = 3, l_2 = 1, l_3 = 5.$
- Adjust: add edge (v_0, v_1) with weight 2.
- **Iteration 2:** $l_0 = 0, l_1 = 3, l_2 = 2, l_3 = 6.$

Vertex cover

- Given a graph G(V, E)
  - Find a subset of the vertices
    - covering all the edges.
- Intractable problem.
- Goals:
  - Minimum cover.
  - Irredundant cover:
    - No vertex can be removed.

Example



Heuristic algorithm vertex based

VERTEX_COVERV(G(V; E))

{

$C = \emptyset$;

while ($E \neq \emptyset$) **do** {

select a vertex $v \in V$;

delete v from $G(V, E)$;

$C = C \cup \{f_v\}$;

}

}

Heuristic algorithm edge based

VERTEX_COVERE(G(V, E))

```
{
      C = ∅;
       while (E ≠ ∅) do {
              select an edge {u, v} ∈ E;
              C=C ∪ {u, v};
              delete from G(V, E) any edge incident
                     to either u or v ;
                     }
}
```

Graph coloring

- Vertex labeling (coloring):
  - No edge has end-point with the same label.
- Intractable on general graphs.
- Polynomial-time algorithms for chordal (and interval) graphs:
  - Left-edge algorithm.

Graph coloring heuristic algorithm

```
VERTEX_COLOR(G(V, E))
{
    for (i =1 to |V| ) {
        c =1
        while (∃ a vertex adjacent to v_i
                    with color c) do {
                        c = c +1;
                        color v_i with color c ;
                        }
                    }
}
```

Graph coloring exact algorithm

```
EXACT_COLOR( G( V, E) , k)
{
repeat {
        NEXT VALUE( k) ;
        if ( c k == 0)
                return ;
        if ( k == n)
                c is a proper coloring;
        else
                EXACT COLOR( G( V, E) , k+ 1)
        }
}
```

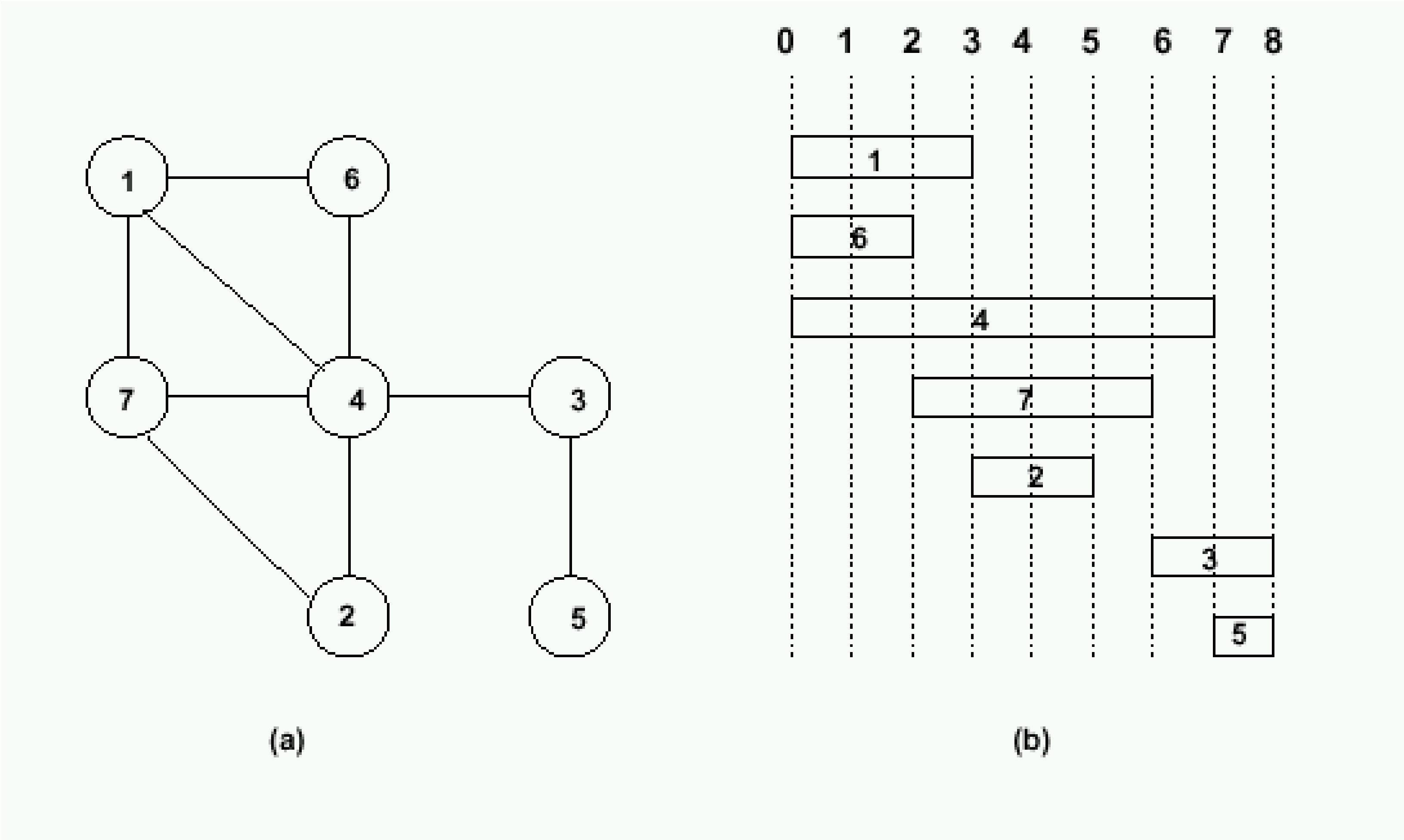
Graph coloring exact algorithm

```
NEXT VALUE( k)
{
repeat {
        c_k = c_k + 1;
        if ( there is no adjacent vertex to v_k
              with the same color c_k )
          return ;
      } until ( c_k =< maximum number of colors ) ;
      c_k = 0;
}
```

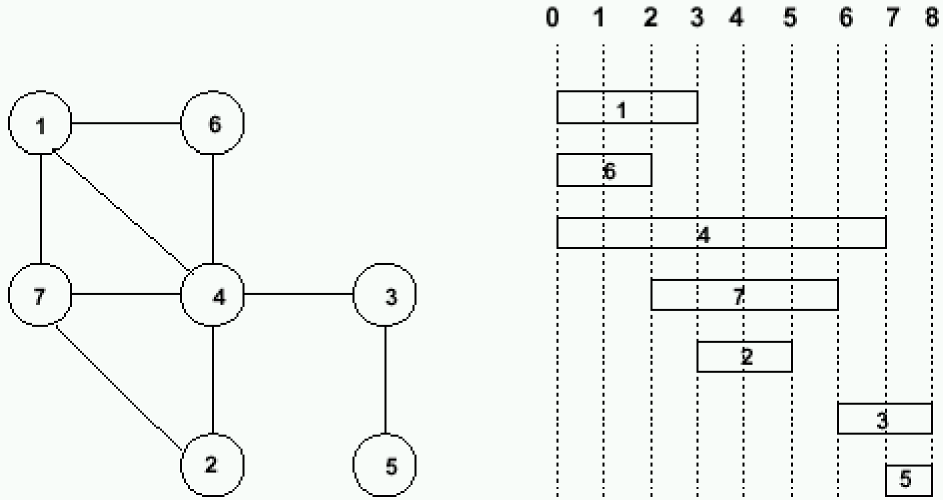
Interval graphs

- Edges represent interval intersections.
- Example:
  - Restricted channel routing problem with no vertical constraints.
- Possible to sort the intervals by left edge.

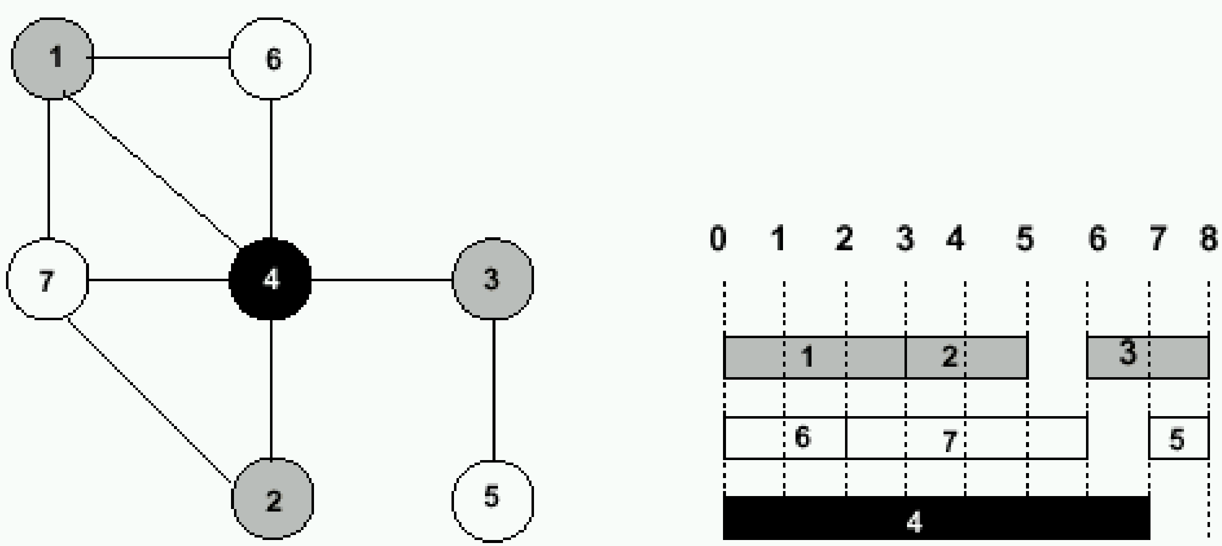
Example



Example



(a) (b)



(c) (d)

Left-edge algorithm

```
LEFT EDGE( I)
{
        Sort elements of I in a list L with ascending order of l_i ;
        c = 0;
        while (Some interval has not been colored ) do {
              S = ∅;
            repeat {
                    s = first element in the list L whose left edge
                          l s is higher than the rightmost edge in S.
                    S= S ∪ { s} ;
                  } until ( an element s is found );
                  c = c + 1;
                  color elements of S with color c;
                  delete elements of S from L;
                                                   }
}
```

Clique partitioning and covering

- A clique partition is a cover.
- A clique partition can be derived from a cover by making the vertex subsets disjoint.
- Intractable problem on general graphs.
- Heuristics:
  - Search for maximal cliques.
- Polynomial-time algorithms for chordal graphs.

Heuristic algorithm

```
CLIQUEPARTITION( G( V, E) )
{
        =∅;
        while ( G( V, E) not empty ) do {
                        compute largest clique C V in G( V, E) ;
                          =∪ C;
                         delete C from G( V, E) ;
                        }
}

CLIQUE( G( V, E) )
{
        C = seed vertex;
        repeat {
                select vertex v ∈ V , v∉ C
                    and adjacent to all vertices of C;
                if (no such vertex is found) return
                    C = C ∪ {v} ;
              }
}
```

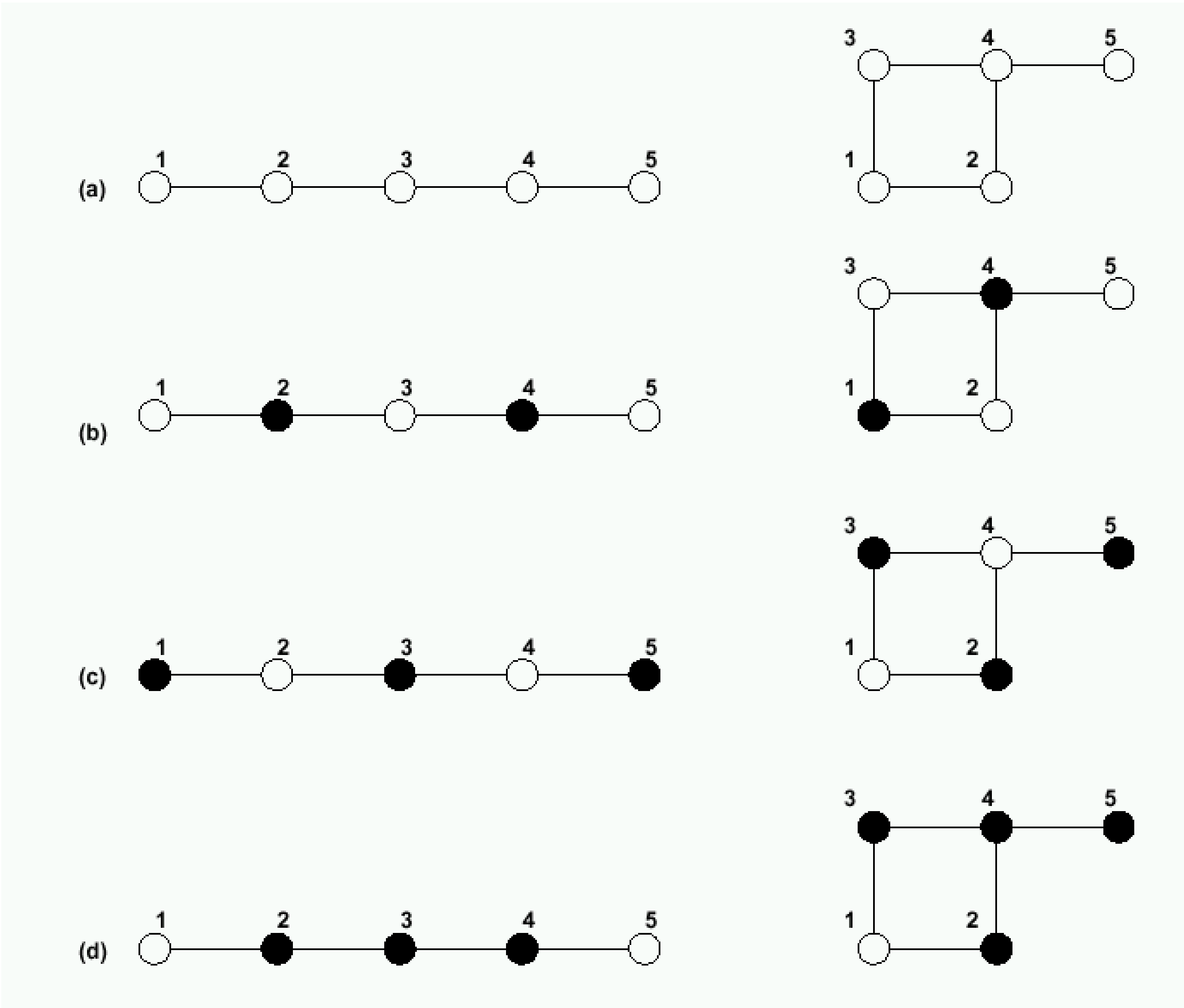
Covering and Satisfiability

- Covering problems can be cast as satisfiability.
- Vertex cover.
  - **Ex1:** $(x_1 + x_2) (x_2 + x_3) (x_3 + x_4) (x_4 + x_5)$
  - **Ex2:** $(x_3 + x_4) (x_1 + x_3) (x_1 + x_2) (x_2 + x_4) (x_4 + x_5)$
- Objective function:
- **Result:**
  - **Ex1:** $x_2 = 1, x_4 = 1$
  - **Ex2:** $x 1 = 1, x_4 = 1$

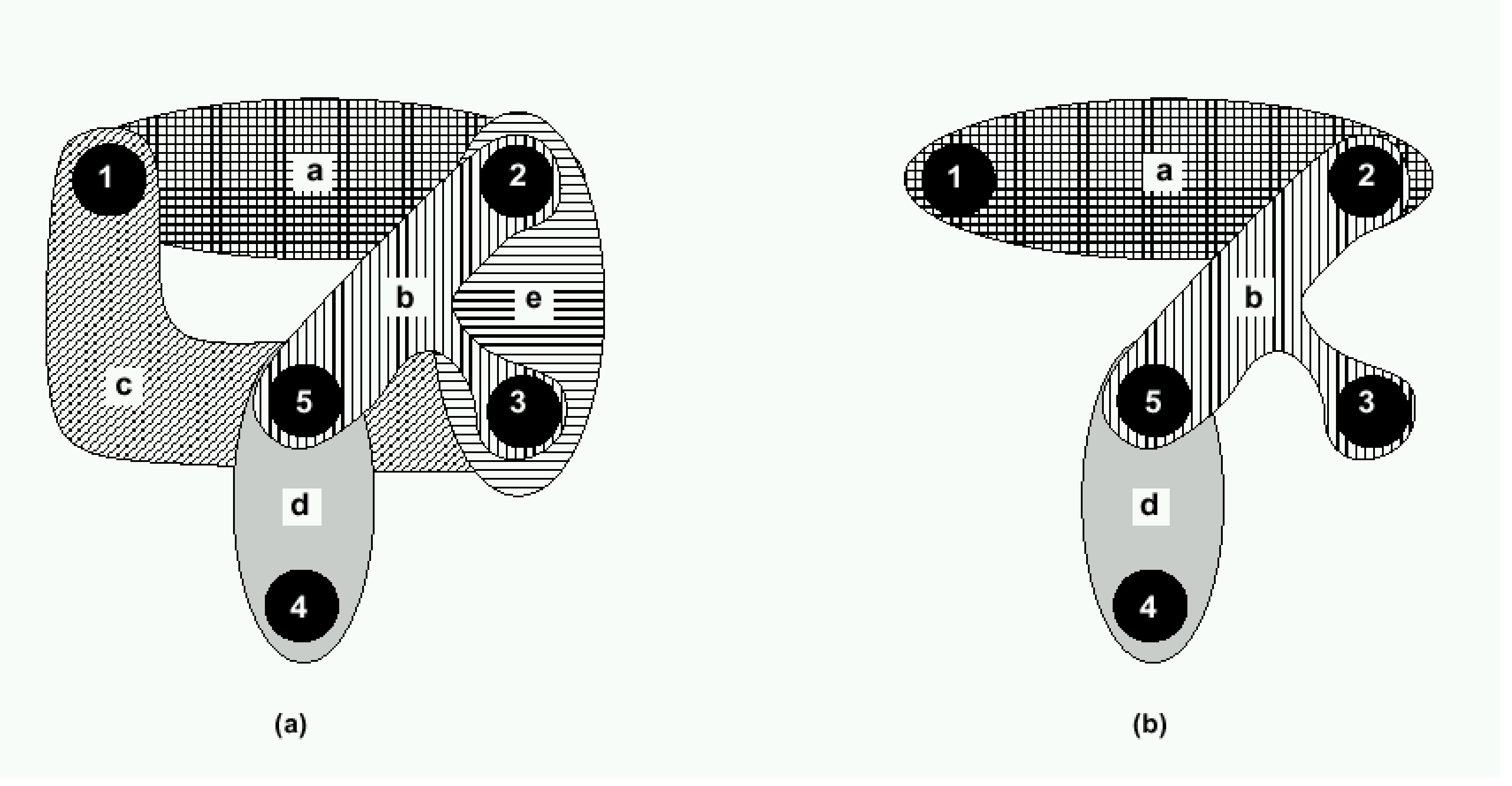
Covering problem

- Set covering problem:
  - A set S.
  - A collection C of subsets.
  - Select fewest elements of C to cover S.
- Intractable.
- Exact method:
  - Branch and bound algorithm.
- Heuristic methods.

Example
vertex-cover of a graph



Example
edge-cover of a hypergraph



Matrix representation

- Boolean matrix: **A.**
- Selection Boolean vector: **x.**
- Determine **x** such that:
  - **A x** $\geq$ 1.
  - Select enough columns to cover all rows.
- Minimize cardinality of **x.**

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Branch and bound algorithm

- Tree search of the solution space:
  - Potentially exponential search.
- Use bounding function:
  - If the lower bound on the solution cost that can be derived from a set of future choices exceeds the cost of the best solution seen so far:
  - Kill the search.
- Good pruning may reduce run-time.

Branch and bound algorithm

```
BRANCH_AND_BOUND{
Current best = anything;
Current cost = 1;
S= s 0 ;
        while (S6 = ; ) do {
            Select an element in s 2S;
            Remove s from S ;
            Make a branching decision based on s
               yielding sequences f s i ; i= 1; 2; ...; mg ;
        for ( i = 1 to m) {
           Compute the lower bound b i of s i ;
           if ( b i  Current cost) Kill s i ;
            else {
                if ( s i is a complete solution ) {
                  Current best = s i ,
                 Current cost = cost of s i,
                  }
           else
              Add s i to set S;
                }
          }
      }
}
```

Example