Introduction to Parallel Computing

George Karypis

Graph Algorithms

Outline

- Graph Theory Background
- Minimum Spanning Tree
  - Prim’s algorithm
- Single-Source Shortest Path
  - Dijkstra’s algorithm
- All-Pairs Shortest Path
  - Dijkstra’s algorithm
  - Floyd’s algorithm
- Maximal Independent Set
  - Luby’s algorithm

Background

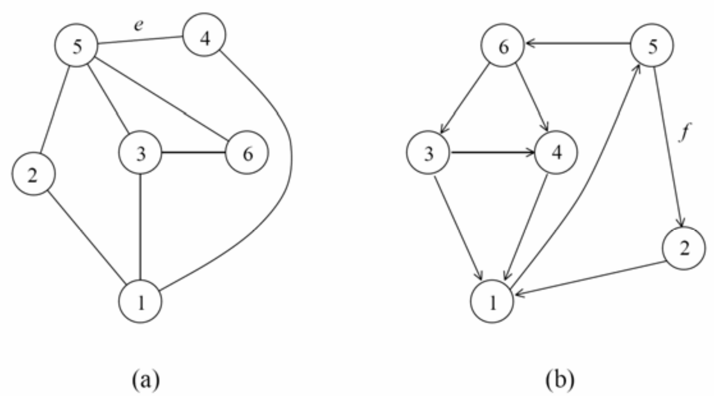


Figure 10.1 (a) An undirected graph and (b) a directed graph.

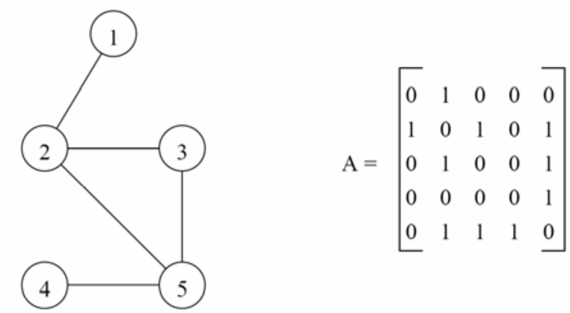


Figure 10.2 An undirected graph and its adjacency matrix representation.

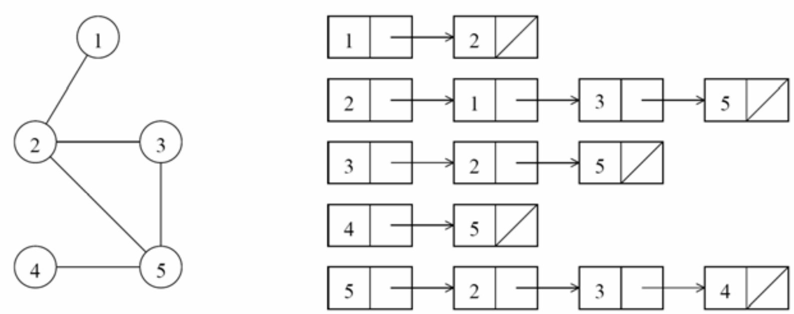


Figure 10.3 An undirected graph and its adjacency list representation.

Minimum Spanning Tree

- Compute the minimum weight spanning tree of an undirected graph.

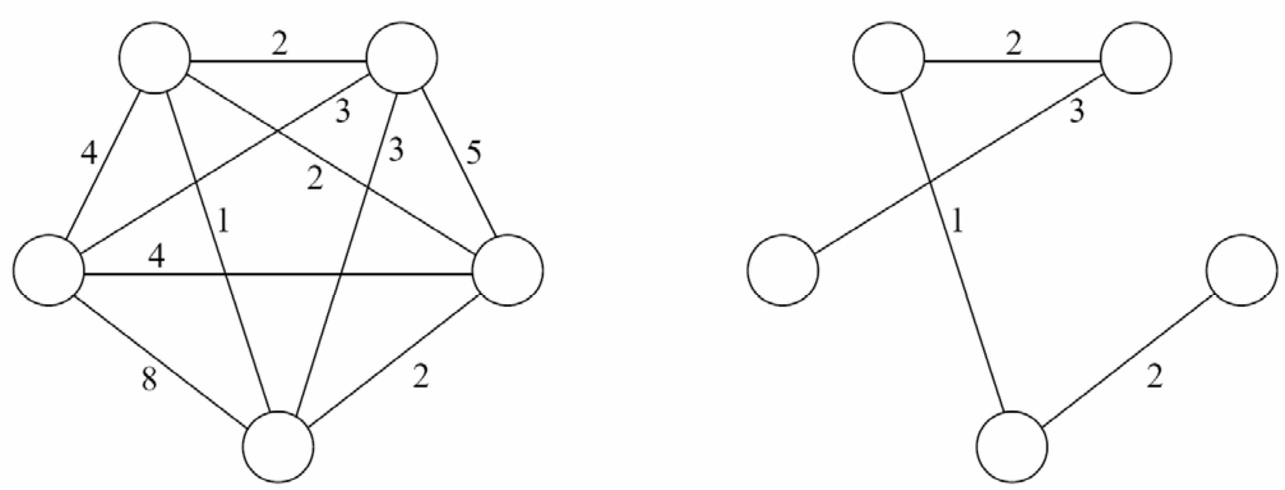
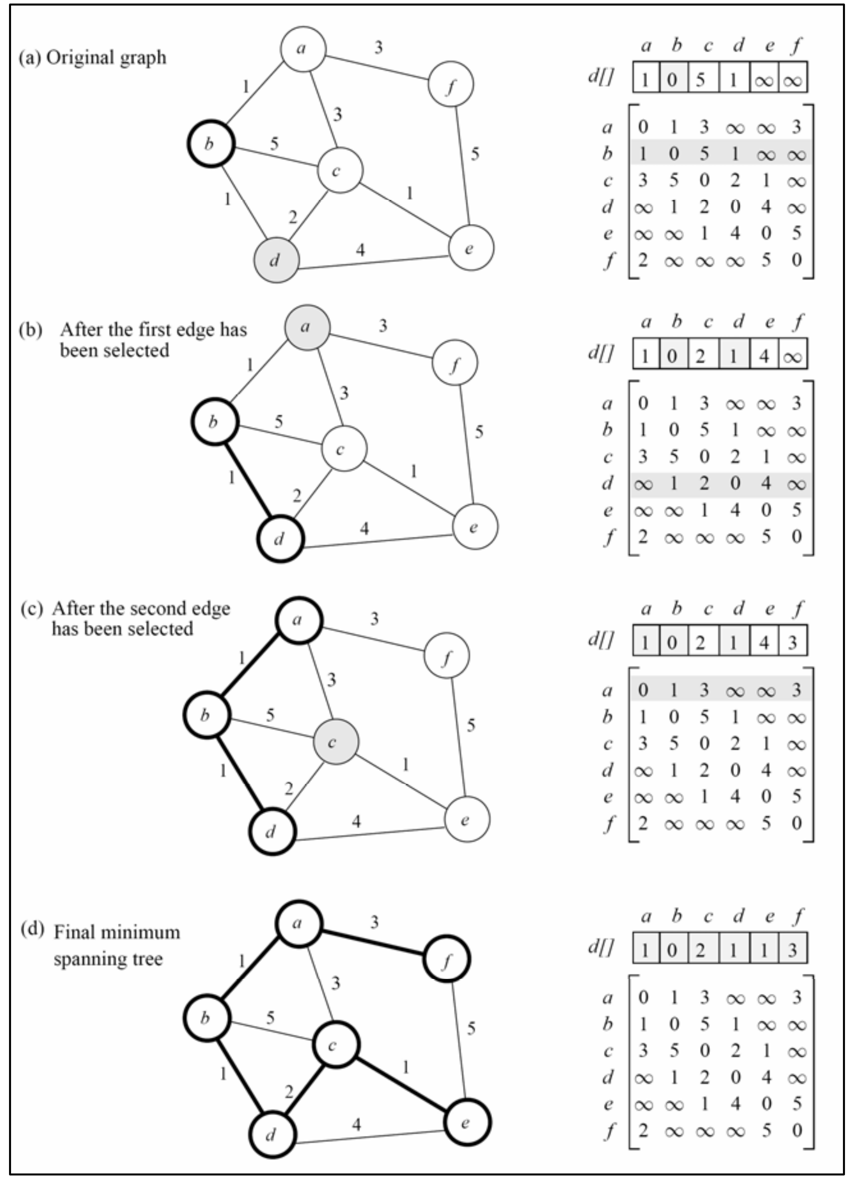


Figure 10.4 An undirected graph and its minimum spanning tree.

Prim's Algorithm

- Prim's Algorithm
  - $\Theta(n^2)$ serial complexity for dense graphs.
    - why?
- How can we parallelize this algorithm?
- Which steps can be done in parallel?

```
procedure PRIM_MST(V, E, w, r)
begin
    V_T := {r};
    d[r] := 0;
    for all v ∈ (V − V_T) do
        if edge (r, v) exists set d[v] := w(r, v);
        else set d[v] := ∞;
    while V_T ≠ V do
    begin
        find a vertex u such that d[u] := min{d[v]|v ∈ (V − V_T)};
        V_T := V_T ∪ {u};
        for all v ∈ (V − V_T) do
            d[v] := min{d[v], w(u, v)};
    endwhile
end PRIM_MST
```



Parallel Formulation of Prim's Algorithm

- Parallelize the inner-most loop of the algorithm.
  - Parallelize the selection of the "minimum weight edge" connecting an edge in V_T to a vertex in V-V_T.
  - Parallelize the updating of the *d*[] array.
- What is the maximum concurrency that such an approach can use?
- How do we "implement" it on a distributed-memory architecture?

```
procedure PRIM_MST(V, E, w, r)
begin
    V_T := {r};
    d[r] := 0;
    for all v ∈ (V − V_T) do
        if edge (r, v) exists set d[v] := w(r, v);
        else set d[v] := ∞;
    while V_T ≠ V do
    begin
        find a vertex u such that d[u] := min{d[v]|v ∈ (V − V_T)};
        V_T := V_T ∪ {u};
        for all v ∈ (V − V_T) do
            d[v] := min{d[v], w(u, v)};
    endwhile
end PRIM_MST
```

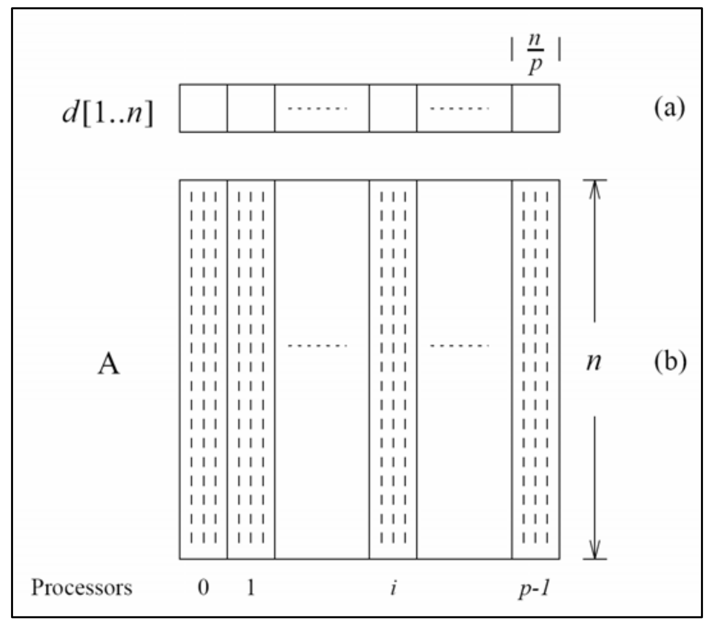
Parallel Formulation of Prim's Algorithm

- Decompose the graph *A* (adjacency matrix) and vector *d* vector using a 1D block partitioning along columns.
  - Why columns?
- Assign each block of size *n/p* to one of the processors.
- How will lines 10 & 12—13 be performed?
- Complexity?

```
1.   procedure PRIM_MST(V, E, w, r)
2.   begin
3.      V_T := {r};
4.      d[r] := 0;
5.      for all v ∈ (V − V_T) do
6.         if edge (r, v) exists set d[v] := w(r, v);
7.         else set d[v] := ∞;
8.      while V_T ≠ V do
9.      begin
10.        find a vertex u such that d[u] := min{d[v]|v ∈ (V − V_T)};
11.        V_T := V_T ∪ {u};
12.        for all v ∈ (V − V_T) do
13.           d[v] := min{d[v], w(u, v)};
14.     endwhile
15.  end PRIM_MST
```

$$T_P = \overbrace{\Theta\left(\frac{n^2}{p}\right)}^{\text{computation}} + \overbrace{\Theta(n \log p).}^{\text{communication}}$$

Isoefficiency: $\Theta(p^2 \log^2 p)$



Single-Source Shortest Path

- Given a *source* vertex *s* find the shortest-paths to all other vertices.
- Dijkstra's algorithm.
- How can it be parallelized for dense graphs?

```
procedure DIJKSTRA_SINGLE_SOURCE_SP(V, E, w, s)
begin
    V_T := {s};
    for all v ∈ (V − V_T) do
        if (s, v) exists set l[v] := w(s, v);
        else set l[v] := ∞;
    while V_T ≠ V do
    begin
        find a vertex u such that l[u] := min{l[v]|v ∈ (V − V_T)};
        V_T := V_T ∪ {u};
        for all v ∈ (V − V_T) do
            l[v] := min{l[v], l[u] + w(u, v)};
    endwhile
end DIJKSTRA_SINGLE_SOURCE_SP
```

Algorithm 10.2 Dijkstra's sequential single-source shortest paths algorithm.

All-pairs Shortest Paths

- Compute the shortest paths between all pairs of vertices.
- Algorithms
  - Dijkstra’s algorithm
    - Execute the single-source algorithm *n* times.
  - Floyd’s algorithm
    - Based on dynamic programming.

All-Pairs Shortest Path Dijkstra's Algorithm

- Source-partitioned formulation
  - Partition the sources along the different processors.
    - Is it a good algorithm?
      - Computational & memory scalability
      - What is the maximum number of processors that it can use?
- Source-parallel formulation
  - Used when *p* > *n*.
  - Processors are partitioned into *n* groups each having *p/n* processors.
  - Each group is responsible for one single-source SP computation.
  - Complexity?

$$T_P = \Theta(n^2).$$

$$\Theta(p^3),$$

$$T_P = \overbrace{\Theta\left(\frac{n^3}{p}\right)}^{\text{computation}} + \overbrace{\Theta(n \log p)}^{\text{communication}}.$$

$$\Theta((p \log p)^{1.5}).$$

Floyd's Algorithm

- Solves the problem using a dynamic programming algorithm.
  - Let $d^{(k)}_{i,j}$ be the shortest path distance between vertices *i* and *j* that goes only through vertices 1,…, *k*.

$$d_{i,j}^{(k)} = \begin{cases} w(v_i, v_j) & \text{if} \quad k = 0 \\ \min\left\{d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right\} & \text{if} \quad k \geq 1 \end{cases}$$

```
procedure FLOYD_ALL_PAIRS_SP(A)
begin
    D^(0) = A;
    for k := 1 to n do
        for i := 1 to n do
            for j := 1 to n do
                d^(k)_{i,j} := min(d^(k-1)_{i,j}, d^(k-1)_{i,k} + d^(k-1)_{k,j});
end FLOYD_ALL_PAIRS_SP
```

  - Complexity: $\Theta(n^3)$.
  - Note: The algorithm can run in-place.
- How can we parallelize it?

Parallel Formulation of Floyd's Algorithm

- Distribute the matrix using a 2D block decomposition.
- Parallelize the double inner-most loop.

procedure FLOYD_ALL_PAIRS_SP(A)
begin
  $D^{(0)} = A$;
  for $k := 1$ **to** n **do**
    for $i := 1$ **to** n **do**
      for $j := 1$ **to** n **do**
        $d_{i,j}^{(k)} := \min\left(d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\right)$;
end FLOYD_ALL_PAIRS_SP

- Communication pattern?
- Complexity?

$$T_P = \overbrace{\Theta\left(\frac{n^3}{p}\right)}^{\text{computation}} + \overbrace{\Theta\left(\frac{n^2}{\sqrt{p}}\log p\right)}^{\text{communication}}.$$

$$\Theta(p^{1.5}\log^3 p).$$

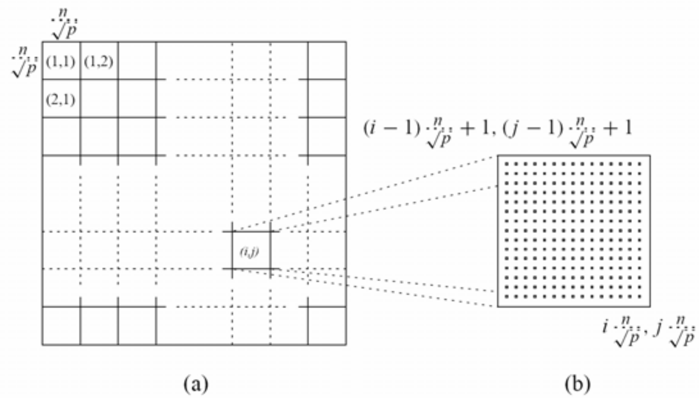


Figure 10.7 (a) Matrix $D^{(k)}$ distributed by 2-D block mapping into $\sqrt{p} \times \sqrt{p}$ subblocks, and (b) the subblock of $D^{(k)}$ assigned to process $P_{i,j}$.

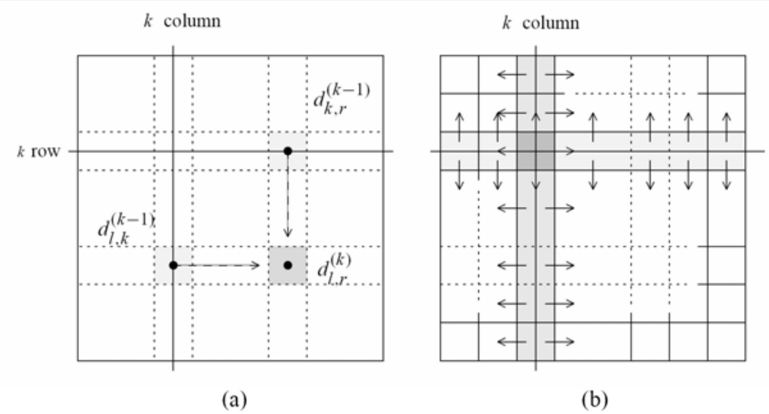


Figure 10.8 (a) Communication patterns used in the 2-D block mapping. When computing $d_{i,j}^{(k)}$, information must be sent to the highlighted process from two other processes along the same row and column. (b) The row and column of $\sqrt{p}$ processes that contain the k^{th} row and column send them along process columns and rows.

procedure FLOYD_2DBLOCK($D^{(0)}$)
begin
  for $k := 1$ **to** n **do**
  begin
    each process $P_{i,j}$ that has a segment of the k^{th} row of $D^{(k-1)}$;
      broadcasts it to the $P_{*,j}$ processes;
    each process $P_{i,j}$ that has a segment of the k^{th} column of $D^{(k-1)}$;
      broadcasts it to the $P_{i,*}$ processes;
    each process waits to receive the needed segments;
    each process $P_{i,j}$ computes its part of the $D^{(k)}$ matrix;
  end
end FLOYD_2DBLOCK

Algorithm 10.4 Floyd's parallel formulation using the 2-D block mapping. $P_{*,j}$ denotes all the processes in the j^{th} column, and $P_{i,*}$ denotes all the processes in the i^{th} row. The matrix $D^{(0)}$ is the adjacency matrix.

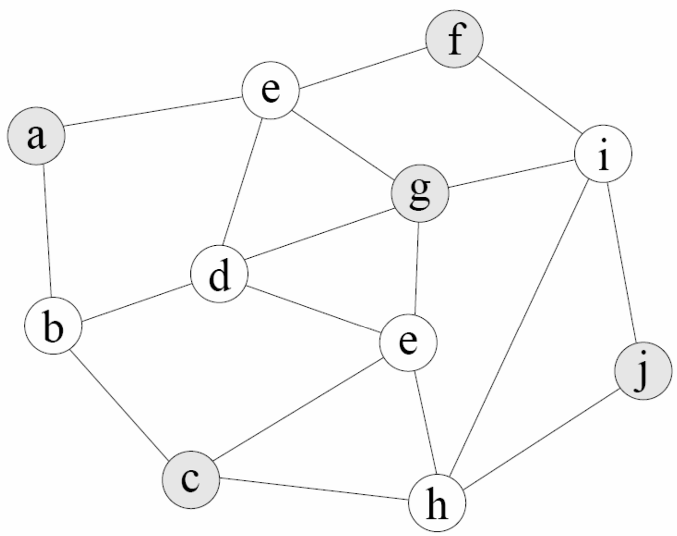
Comparison of All-Pairs SP Algorithms

Table 10.1 The performance and scalability of the all-pairs shortest paths algorithms on various architectures with $O(p)$ bisection bandwidth. Similar run times apply to all $k-d$ cube architectures, provided that processes are properly mapped to the underlying processors.

	Maximum Number of Processes for $E = \Theta(1)$	Corresponding Parallel Run Time	Isoefficiency Function
Dijkstra source-partitioned	$\Theta(n)$	$\Theta(n^2)$	$\Theta(p^3)$
Dijkstra source-parallel	$\Theta(n^2/\log n)$	$\Theta(n \log n)$	$\Theta((p \log p)^{1.5})$
Floyd 1-D block	$\Theta(n/\log n)$	$\Theta(n^2 \log n)$	$\Theta((p \log p)^3)$
Floyd 2-D block	$\Theta(n^2/\log^2 n)$	$\Theta(n \log^2 n)$	$\Theta(p^{1.5} \log^3 p)$
Floyd pipelined 2-D block	$\Theta(n^2)$	$\Theta(n)$	$\Theta(p^{1.5})$

Maximal Independent Sets

- Find the maximal set of vertices that are not adjacent to each other.



{a, d, i, h} is an independent set

{a, c, j, f, g} is a maximal independent set

{a, d, h, f} is a maximal independent set

Figure 10.15 Examples of independent and maximal independent sets.

Serial Algorithms for MIS

- Practical MIS algorithms are incremental in nature.
  - Start with an empty set.
  1. Add the vertex with the smallest degree.
  2. Remove adjacent vertices
  3. Repeat 1—2 until the graph becomes empty.
- These algorithms are impossible to parallelize.
  - Why?
- Parallel MIS algorithms are based on the ideas initially introduced by Luby.

Luby's MIS Algorithm

- Randomized algorithm.
  - Starts with an empty set.
  1. Assigns random numbers to each vertex.
  2. Vertices whose random number are smaller than all of the numbers assigned to their adjacent vertices are included in the MIS.
  3. Vertices adjacent to the newly inserted vertices are removed.
  4. Repeat steps 1—3 until the graph becomes empty.
- This algorithms will terminate in O(log (n)) iterations.
- Why is this a good algorithm to parallelize?
- How will the parallel formulation proceed?
  - Shared memory
  - Distributed memory

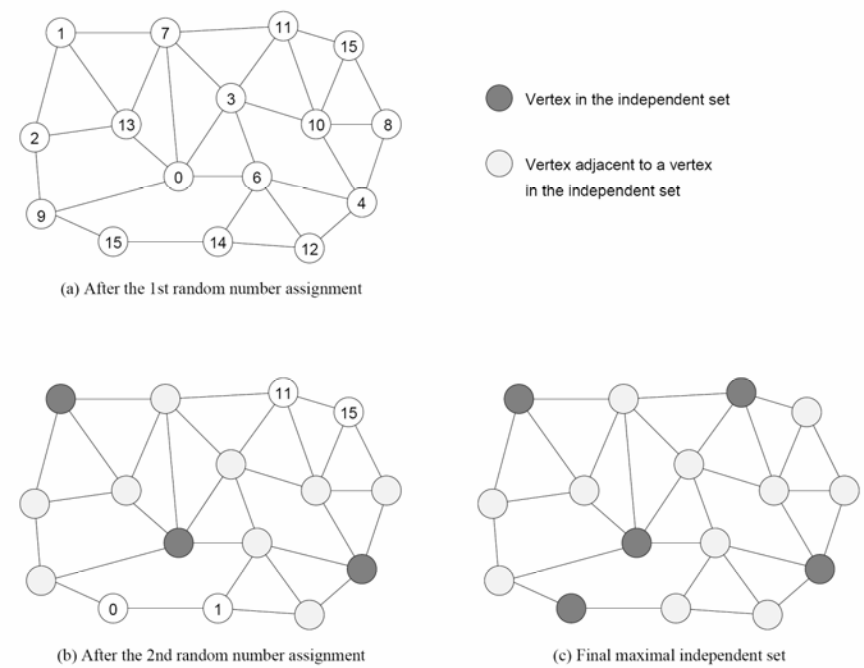


Figure 10.16 The different augmentation steps of Luby's randomized maximal independent set algorithm. The numbers inside each vertex correspond to the random number assigned to the vertex.