Approximation Algorithms

- There are two classes of problems:

  1) Problems whose solution is by a polynomial time algorithm
  2) Problems for which no polynomial time algorithm is known

- The first group consists of problems whose solution is bounded by a polynomial of small degree. e.g., Binary searching in $O(\log n)$, sorting $O(n \log n)$, and matrix multiplication $O(n^{2.81})$.

- The second group contains problems whose best known algorithms are nonpolynomial, e.g., traveling salesperson in $O(n^2 2^n)$, knapsack problem in $O(2n^{1/2})$

- The theory of NP-completeness does not provide a method of obtaining polynomial time algorithms in the second group. It cannot also say that polynomial time algorithms do not exist.

- Instead, it can say that many of the problems for which there is no known polynomial time algorithm are computationally related.

- **Decision Problem:** Any problem for which the answer is 0 or 1 is called decision problem and the corresponding algorithm is called decision algorithm.

- **Optimization Problem:** Any problem which seeks for an optimal solution is called optimization problem and the corresponding algorithm is called optimization algorithm.

- In fact, we shall establish two classes of problems. These will be given the names NP-hard and NP-complete.

- A problem which is NP-complete will have the property that it can be solved in polynomial time iff all other NP-complete problems can also be solved in polynomial time.

- If an NP-hard problem can be solved in polynomial time then all NP-complete problems can be solved in polynomial time.

- All NP-complete problems are NP-hard but all NP-hard problems are not NP-complete.

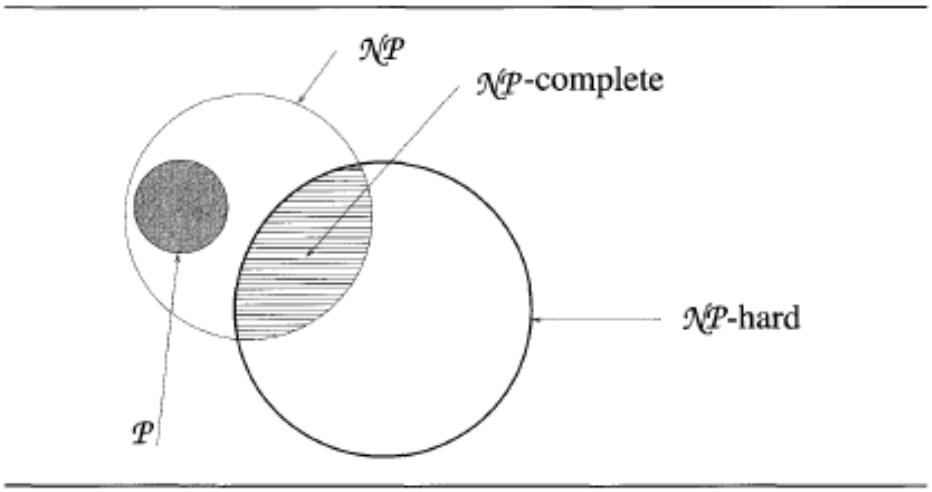


Figure 11.2 Commonly believed relationship among $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-complete, and $\mathcal{NP}$-hard problems

Note 2: What we do not know, and what has become perhaps the most famous unsolved problem in computer science is whether $P = NP$ or $P \neq NP$.

What is an Approximation Algorithm?

An algorithm that returns near optimal solution.

What is the Motivation? NP-complete problems.

Two approaches:

1) For small size problem, an exponential running time algorithm may be satisfactory.
2) Find a near optimal algorithm.

Comment: The second approach gives the birth of an approximation algorithm.

How to evaluate it?

Performance bound: An approximation algorithm has a ratio bound of $f(n)$ if

$$Max(\frac{C}{C*}, \frac{C*}{C}) \leq f(n)$$

where n is input size, C: Cost of the approximation algorithm and C^*: Cost of the optimal solution.

The Vertex Cover Problem:

Statement: A vertex cover of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $(u, v) \in E$, then either $u \in V'$ or $v \in V'$ or both. It is to find a vertex cover of minimum size.

Algorithm APPROX-VERTEX COVER (G)

```
{
 C = Φ
 E' = E [G]
 While E' ≠ Φ do
       Let (u, v) be an arbitrary edge of E'
       C = C ∪ {u, v}
       Remove from E' every edge incident on either u or v
  Return C
 }
```

Running Time: $O(|E|)$

Near Optimality: The above algorithm returns a vertex cover whose size is guaranteed to be no more than twice the size of an optimal vertex cover.

An Illustration:

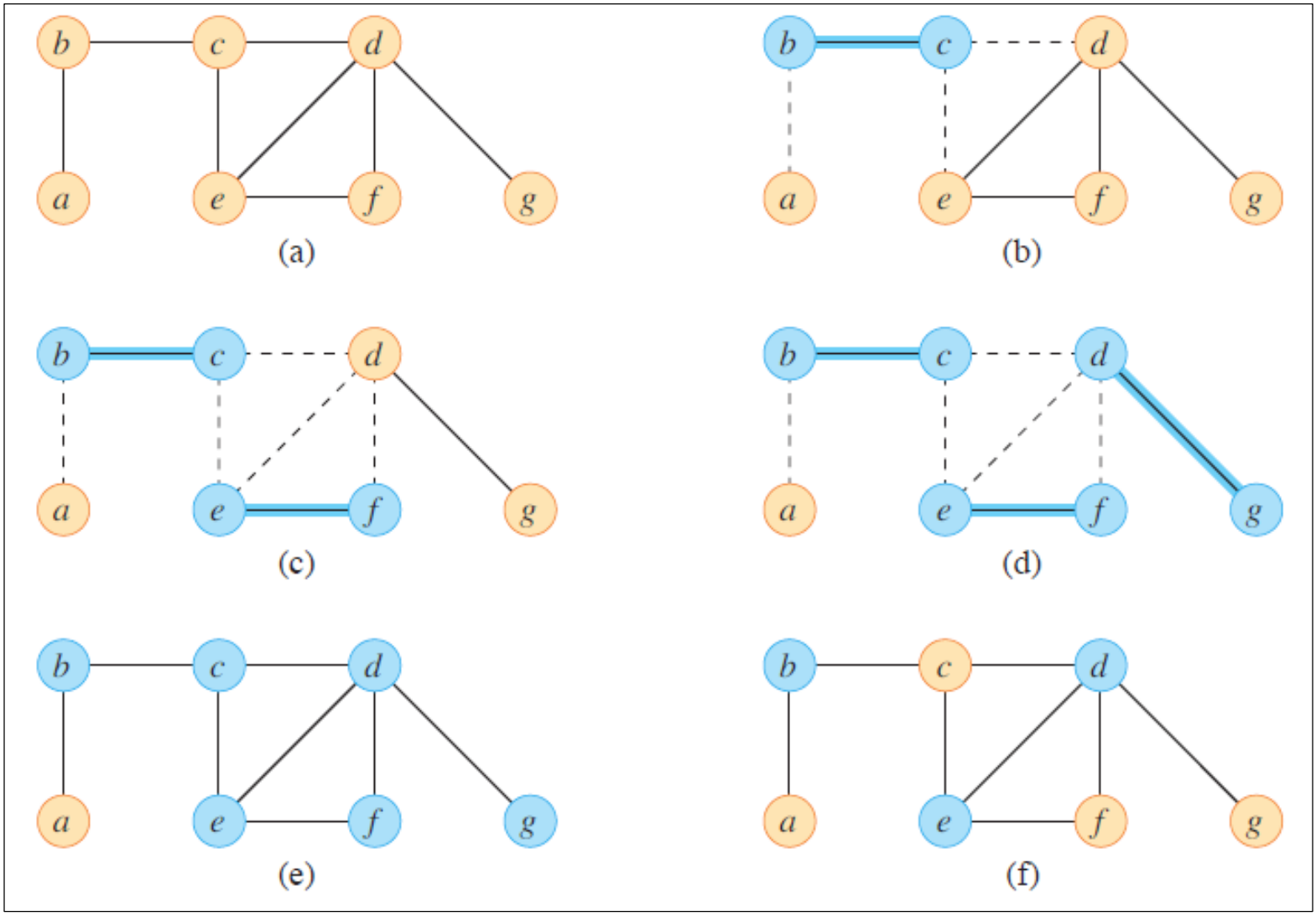


Fig. (a):Input Graph

Fig. (e): Final Vertex Cover C = {*b, c, e, f, d, g*}

Fig. (f): The Optimal vertex cover {*b, e, d*}

Comments: Optimal Sol. is almost half of Approx. Sol.

Theorem: APPROX-VERTEX-COVER is a polynomial-time 2-approximation algorithm.

Proof:

We have already shown that APPROX-VERTEX-COVER runs in polynomial time. The set C of vertices that is returned by APPROX-VERTEX-COVER is a vertex cover, since the algorithm loops until every edge in $E[G]$ has been covered by some vertex in C.

Now we have to see that APPROX-VERTEX-COVER returns a vertex cover that is at most twice the size of an optimal cover.

Let A denote the set of edges that were picked in line 4 of APPROX-VERTEX-COVER. In order to cover the edges in A, any vertex cover—in particular, an optimal cover $C*$—must include at least one endpoint of each edge in A. No two edges in A share an endpoint, since once an edge is picked in line 4, all other edges that are incident on its endpoints are deleted from $E\_$ in line 6. Thus, no two edges in A are covered by the same vertex from $C*$, and we have the lower bound

$$|C*| \geq |A| \qquad (1)$$

on the size of an optimal vertex cover. Each execution of line 4 picks an edge for which neither of its endpoints is already in C, yielding an upper bound (an exact upper bound, in fact) on the size of the vertex cover returned:

$$|C| = 2\,|A| \qquad (2)$$

Combining equations (1) and (2), we obtain

$$\begin{aligned} |C| &= 2\,|A| \\ &\leq 2\,|C*| \end{aligned}$$

Hence the proof.

The Travelling Sales Person Problem:

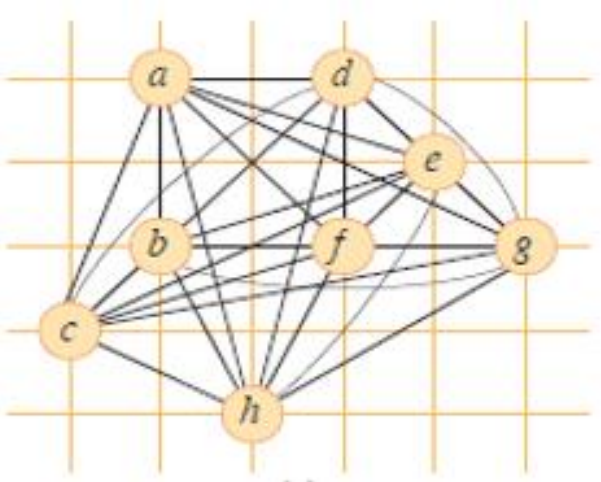
Statement: Given a complete undirected graph $G = (V, E)$ that has a non-negative integer cost $C(u, v)$ associated with each edge $(u, v) \in E$, find a Hamiltonian cycle of G with minimum cost.

let $c(A)$ denote the total cost of the edges in the subset $A \subseteq E$:

$$c(A) = \sum_{(u,v)\in A} c(u, v) .$$

It is always cheapest to go directly from a node u to another node v instead of going via an intermediate node w, i.e., it follows the triangle inequality:

$c(u, v) \leq c(u, w) + c(w, v)$



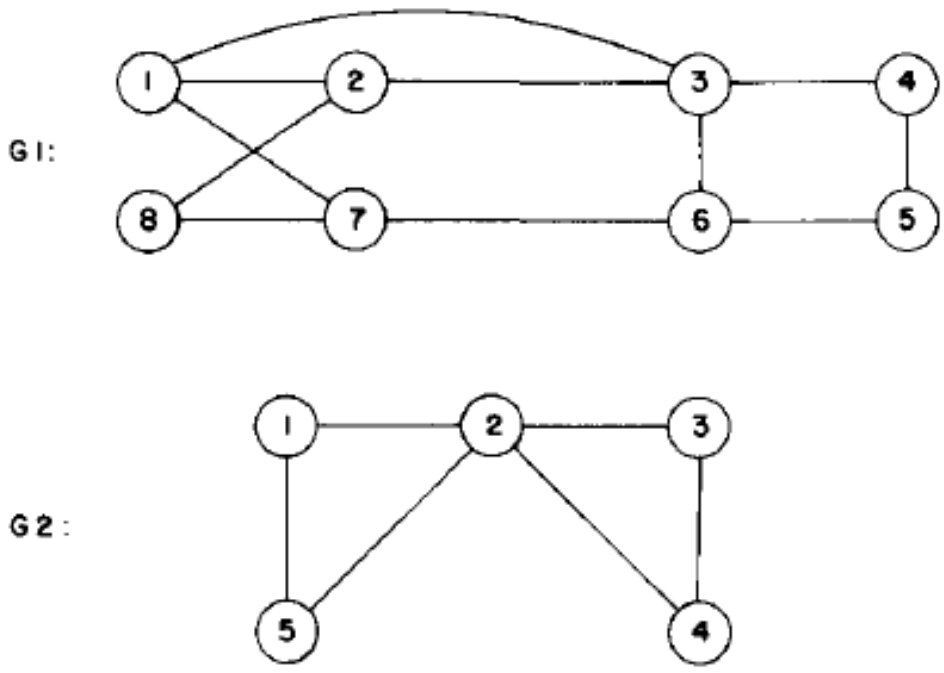


Fig. 1: Two graphs, one containing a Hamiltonian cycle, other does not have.

Approximation Algorithm with Triangle Inequality:

APPROX-TSP-TOUR (*G, C*)

1. Select a vertex $r \in V(G)$ to be a root vertex
2. Grow a minimum spanning tree T for G from the root r.
3. Let L be the list of vertices in a preorder walk of T.
4. Return the Hamiltonian cycle H that visits the vertices in the order L.

Time complexity: $O(|V|^2)$ as step 2 requires $O(|V|^2)$ time using prim's algorithm.

An Illustration:

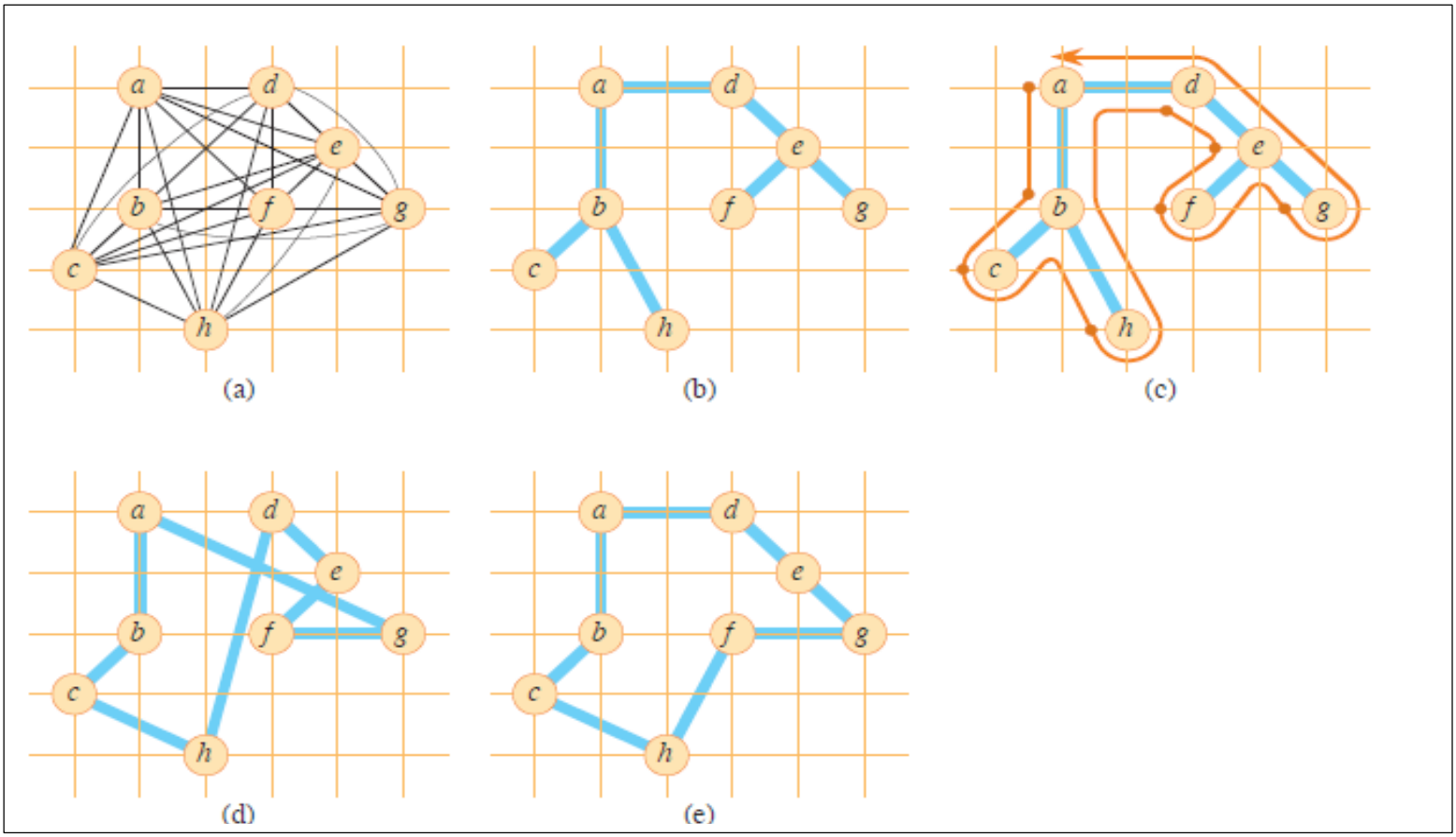


Fig. (a): Input Grid

(b): MST

(c): A preorder Walk {*a, b, c, b, h, b, a, d, e, f, e, g, e, d, a*}

(d): Approx. Tour {*a, b, c, h, d, e, f, g, a*} of **Cost 19.074**

(e): **Optimal Tour of Cost 14.715**

Comment: The Optimal Tour Cost is **23 % shorter** than the Approx. Sol. Cost.

Theorem: **Approx-TSP-Tour is a polynomial time 2-approximation algorithm for TSP with triangle inequality.**

Proof: The algorithm is correct because it produces a Hamiltonian circuit.

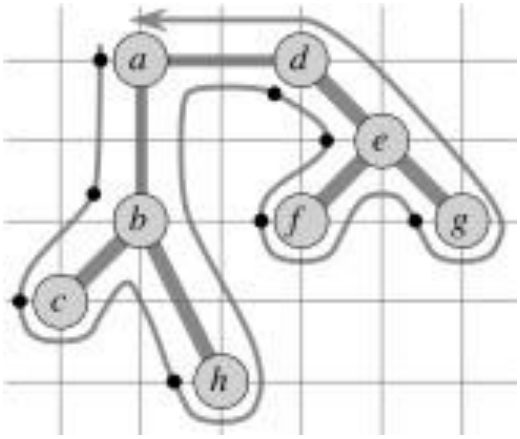
The algorithm is polynomial time because the most expensive operation is MST-Prim, which can be computed in $O(|V|^2)$

For the approximation result, let T be the spanning tree found in line 2, H be the tour found and H^* be an optimal tour for a given problem.

If we delete any edge from H^*, we get a spanning tree that can be no cheaper than the *minimum* spanning tree T, because H^* has one more (nonegative cost) edge than T:

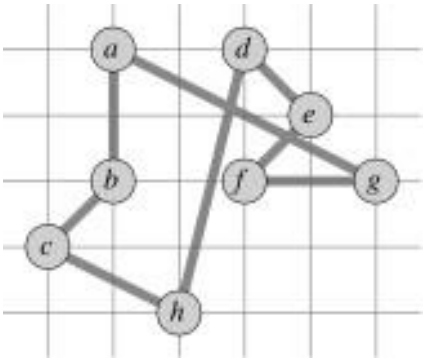
$c(T) \leq c(H^*)$

Consider the cost of the **full walk** W that traverses the edges of T exactly twice starting at the root. (For our example, W is $\langle\{a, b\}, \{b, c\}, \{c, b\}, \{b, h\}, \{h, b\}, \{b, a\}, \{a, d\}, \ldots \{d, a\}\rangle$.)



Since each edge in T is traversed twice in W: $c(W) = 2\,c(T)$

This walk W is not a tour because it visits some vertices more than once, but we can skip the redundant visits to vertices once we have visited them, producing the same tour H as in line 3. (For example, instead of $\langle\{a, b\}, \{b, c\}, \{c, b\}, \{b, h\}, \ldots\rangle$, go direct: $\langle\{a, b\}, \{b, c\}, \{c, h\}, \ldots\rangle$.)



By the triangle inequality, which says it can't cost any more to go direct between two vertices,

$c(H) \leq c(W)$

Noting that H is the tour constructed by Approx-TSP-Tour, and putting all of these together:

$c(H) \leq c(W) = 2\,c(T) \leq 2\,c(H^*)$

So, $c(H) \leq 2\,c(H^*)$, and thus Approx-TSP-Tour is a 2-approximation algorithm for TSP. (The CLRS text notes that there are even better solutions, such as a 3/2-approximation algorithm.)

The set-covering problem

An instance $(X, \mathcal{F})$ of the ***set-covering problem*** consists of a finite set X and a family $\mathcal{F}$ of subsets of X, such that every element of X belongs to at least one subset in $\mathcal{F}$:

$$X = \bigcup_{S \in \mathcal{F}} S .$$

We say that a subfamily $\mathcal{C} \subseteq \mathcal{F}$ ***covers*** a set of elements U if

$$U \subseteq \bigcup_{S \in \mathcal{C}} S .$$

The problem is to find a minimum-size subfamily $\mathcal{C} \subseteq \mathcal{F}$ whose members cover all of X:

$$X = \bigcup_{S \in \mathcal{C}} S .$$

An Illustration:

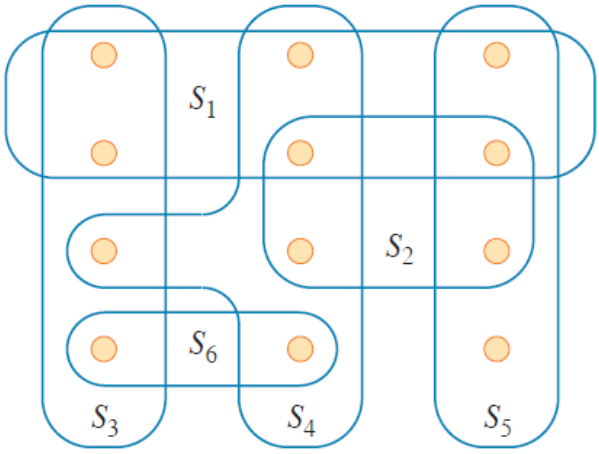
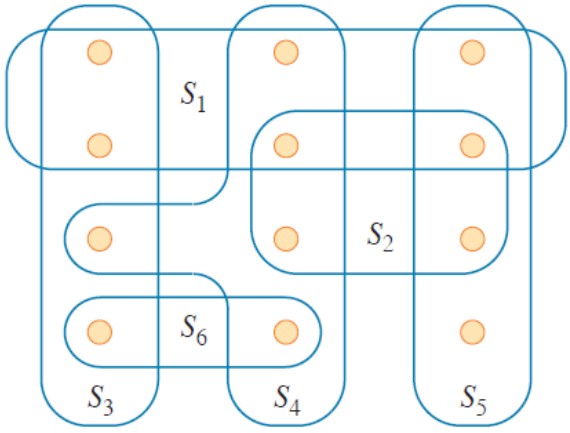


Figure 35.3 An instance $(X, \mathcal{F})$ of the set-covering problem, where X consists of the 12 tan points and $\mathcal{F} = \{S_1, S_2, S_3, S_4, S_5, S_6\}$. Each set $S_i \in \mathcal{F}$ is outlined in blue. A minimum-size set cover is $\mathcal{C} = \{S_3, S_4, S_5\}$, with size 3. The greedy algorithm produces a cover of size 4 by selecting either the sets S_1, S_4, S_5, and S_3 or the sets S_1, S_4, S_5, and S_6, in order.

Note: The set-covering problem is an abstraction of many commonly arising combinatorial problems. As a simple example, suppose that X represents a set of skills that are needed to solve a problem and that we have a given set of people available to work on the problem. We wish to form a committee, containing as few people as possible, such that for every requisite skill in X, there is a member of the committee having that skill.

A greedy approximation algorithm

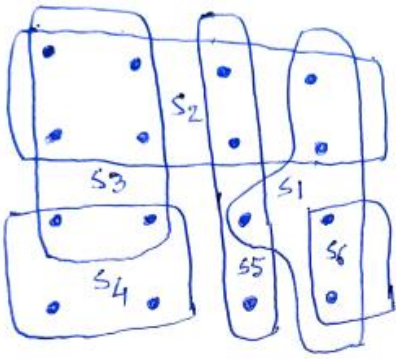
```
GREEDY-SET-COVER(X, F)
1  U ← X
2  C ← ∅
3  while U ≠ ∅
4       do select an S ∈ F that maximizes |S ∩ U|
5          U ← U − S
6          C ← C ∪ {S}
7  return C
```



Result of the approximation algorithm: S1, S4, S5, S3 or S1, S4, S5, S6 in order.

Time Complexity: $O(|X|\,|F|)$

Ex 1: Apply the approximation algorithm to solve set covering problem for the following graph by showing stepwise results.



U = X = (All black points.)

F = { S1, S2, S3, S4, S5, S6 }

T = ϕ

S2 ∩ U is maximum.

∴ U = U - all points in S2

T = { S2 }

S4 ∩ U is maximum

∴ U = U - all points in S4

T = { S2, S4 }

S1 ∩ U is maximum

∴ U = U - all points in S1

T = { S2, S4, S1 }

S5 ∩ U is maximum

U = U - all points in S5

T = { S2, S4, S1, S5 }