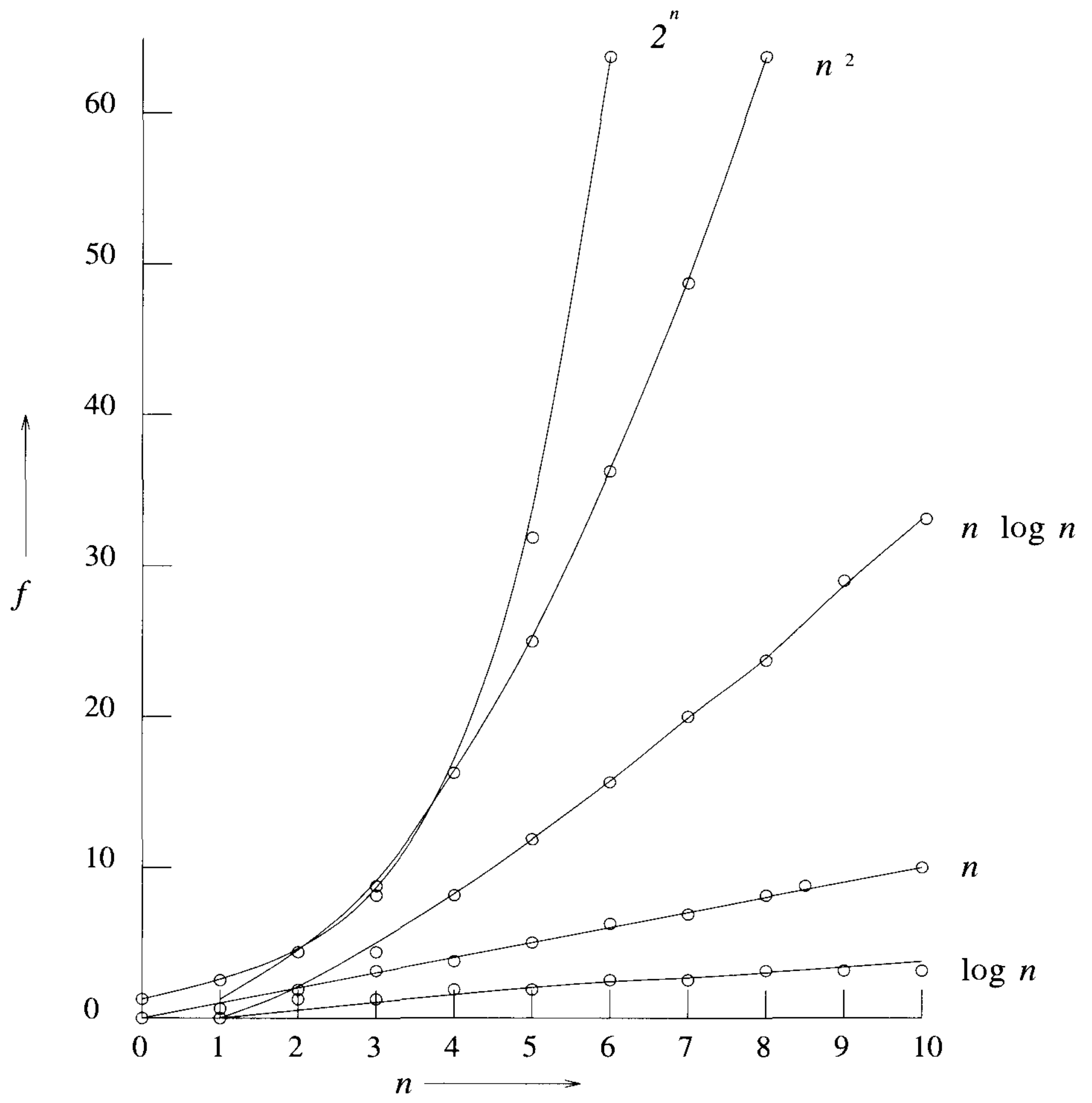
Computer Algorithms

Unit-6

Approximation Algorithms

- 0/1 Knapsack - $O(2^n)$
- Traveling salesperson $O(2^n n^2)$

log n	n	n log n	n^2	n^3	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296


2^n
n^2
$n \log n$
n
$\log n$
f
n
0
10
20
30
40
50
60
0 1 2 3 4 5 6 7 8 9 10

INTRODUCTION

- NP-hard optimization problems have great practical importance
- it is desirable to solve large instances of these problems in a reasonable amount of time.
- The best-known algorithms for NP-hard problems have a worst-case complexity - exponential in terms of number of inputs.
- there is still plenty of room for improvement in an exponential algorithm.

- Algorithms with sub-exponential complexity
- $2^{n/c}$ (for $c > 1$), $2^{\sqrt{n}}$, or $2^{\log n}$.
- what is needed is an algorithm of low polynomial complexity $O(n)$ or $O(n^2)$ or $O(n^3)$.
- If we want to produce an algorithm of low polynomial complexity to solve NP-hard optimization problem,
- then it is necessary to relax the meaning of "solve"
- we discuss two relaxations of the meaning of "solve".

- In the first, we remove the requirement that the algorithm that solves the optimization problem P must always generate an optimal solution.
- This requirement is replaced by the requirement that the algorithm for P must always generate a feasible solution with value close to the value of an optimal solution.
- A feasible solution with value close to the value of an optimal solution is called an approximate solution.
- An approximation algorithm for P is an algorithm that generates approximate solutions for P.

- In the second relaxation we look for an algorithm for P that almost always generates optimal solutions with high probability.
- Algorithms with this property are called probabilistically good algorithms.

- **Terminology :**

P	: Represents an NP-Hard problem such as 0/1 Knapsack or Traveling Salesperson Problem.
I	: Represents an instance of problem P
F*(I)	: Represents an optimal solution to I
F^(I)	: Represents feasible solution produced for I by an approximation algorithm
A	: Represents an algorithm that generates a feasible solution to every instance I of a problem P

F*(I) > F^(I) if P is a maximization problem

F^(I) > F*(I) if P is a minimization problem

- Absolute Approximation
- Let A be an algorithm that generates a feasible solution to every instance I of a problem P.
- Let F*(I) be the value of an optimal solution to I
- Let F^(I) be the value of the feasible solution generated by A
- A is an absolute approximation algorithm for problem P, if and only if for every instance I of P, $|F^*(I) - \hat{F}(I)| \leq k$, for some constant k.

f(n) - Approximation

- Let A be an algorithm that generates a feasible solution to every instance I of a problem P.
- Let F*(I) be the value of an optimal solution to I
- Let F^(I) be the value of the feasible solution generated by A
- A is an f(n) approximation algorithm for problem P, if and only if for every instance I of size n,
- $\dfrac{|F*(I) - F^\wedge(I)|}{F*(I)} \leq f(n)$
- For F*(I) > 0

- ## ε - Approximation
- An ε-approximation algorithm is an f(n) approximation algorithm, for which $f(n) \leq \varepsilon$, for some constant ε.

- ε - Approximation
- Let A be an algorithm that generates a feasible solution to every instance I of a problem P.
- Let F*(I) be the value of an optimal solution to I
- Let F^(I) be the value of the feasible solution generated by A
- A is an ε - approximation algorithm for problem P, if and only if for every instance I,
- $\dfrac{|F*(I) - F\wedge(I)|}{F*(I)} \leq \varepsilon$
- For F*(I) > 0 and ε > 0

- Polynomial Time Approximation Scheme
- An approximation scheme is a polynomial time approximation scheme
- if and only if for every fixed $\varepsilon > 0$,

  it has a computing time that is polynomial in terms of the problem size n

- Fully Polynomial Time Approximation Scheme
- An approximation scheme is a fully polynomial time approximation scheme
- If it has a computing time that is polynomial in terms of the problem size n and 1/ε

Let A be an algorithm that generates a feasible solution to every instance I of a problem P. Let $F^*(I)$ be the value of an optimal solution to I and let $\hat{F}(I)$ be the value of the feasible solution generated by A.

Definition 12.1 A is an *absolute approximation* algorithm for problem P if and only if for every instance I of P, $|F^*(I) - \hat{F}(I)| \leq k$ for some constant k. □

Definition 12.2 A is an $f(n)$-*approximate* algorithm if and only if for every instance I of size n, $|F^*(I) - \hat{F}(I)|/F^*(I) \leq f(n)$ for $F^*(I) > 0$. □

Definition 12.3 An ϵ-*approximate* algorithm is an $f(n)$-approximate algorithm for which $f(n) \leq \epsilon$ for some constant ϵ. □

Definition 12.4 $A(\epsilon)$ is an *approximation scheme* if and only if for every given $\epsilon > 0$ and problem instance I, $A(\epsilon)$ generates a feasible solution such that $|F^*(I) - \hat{F}(I)|/F^*(I) \leq \epsilon$. Again, we assume $F^*(I) > 0$. □

Definition 12.5 An approximation scheme is a *polynomial time approximation scheme* if and only if for every fixed $\epsilon > 0$, it has a computing time that is polynomial in the problem size. □

Definition 12.6 An approximation scheme whose computing time is a polynomial both in the problem size and in $1/\epsilon$ is a *fully polynomial time approximation scheme.* □

- Consider a Knapsack Problem
- n=3 m = 100
- $\{p_1, p_2, p_3\} = \{20, 10, 19\}$
- $\{w_1, w_2, w_3\} = \{65, 20, 35\}$

- The Solution $(x_1, x_2, x_3) = (1, 1, 1)$ is not feasible.
- The Solution $(x_1, x_2, x_3) = (1, 0, 1)$ is optimal.
  Profit=39
- The Solution $(x_1, x_2, x_3) = (1, 1, 0)$ is sub-optimal.
  Profit=30
- It is one of the approximate solution.

- $F^*(I) = 39$
- $F\hat{}(I) = 30$
- $|F^*(I) - F\hat{}(I)| = 9$
- $\dfrac{|F*(I) - F\hat{}(I)|}{F*(I)} = 0.3$

- Consider an instance of 0/1 Knapsack
- n=2 m = 4
- $\{p_1, p_2\} = \{100, 20\}$
- $\{w_1, w_2\} = \{4, 1\}$

- The Solution $(x_1, x_2) = (1, 1)$ is not feasible.
- The Solution $(x_1, x_2) = (1, 0)$ is optimal. Profit=100
- The Solution $(x_1, x_2) = (0, 1)$ is sub-optimal. Profit=20

- F*(I) = 100
- F^(I) = 20

- |F*(I) – F^(I)| = 80
- It is too high.
- It is not an approximate solution

- $\frac{|F*(I) - F\wedge(I)|}{F*(I)} = 0.8$

- Consider an instance of 0/1 Knapsack
- n=2 m = r
- $\{p_1, p_2\} = \{2, r\}$
- $\{w_1, w_2\} = \{1, r\}$

- The Solution $(x_1, x_2) = (1, 1)$ is not feasible.
- When $r>2$
- The Solution $(x_1, x_2) = (0, 1)$ is optimal.
- Profit = r
- The Solution $(x_1, x_2) = (1, 0)$ is sub-optimal.

  Profit=2

- $F^*(I) = r$
- $F^\wedge(I) = 2$

- $|F^*(I) - F^\wedge(I)| = r - 2$
- $r - 2$ is not constant.
- So, it is not an absolute approximation scheme.

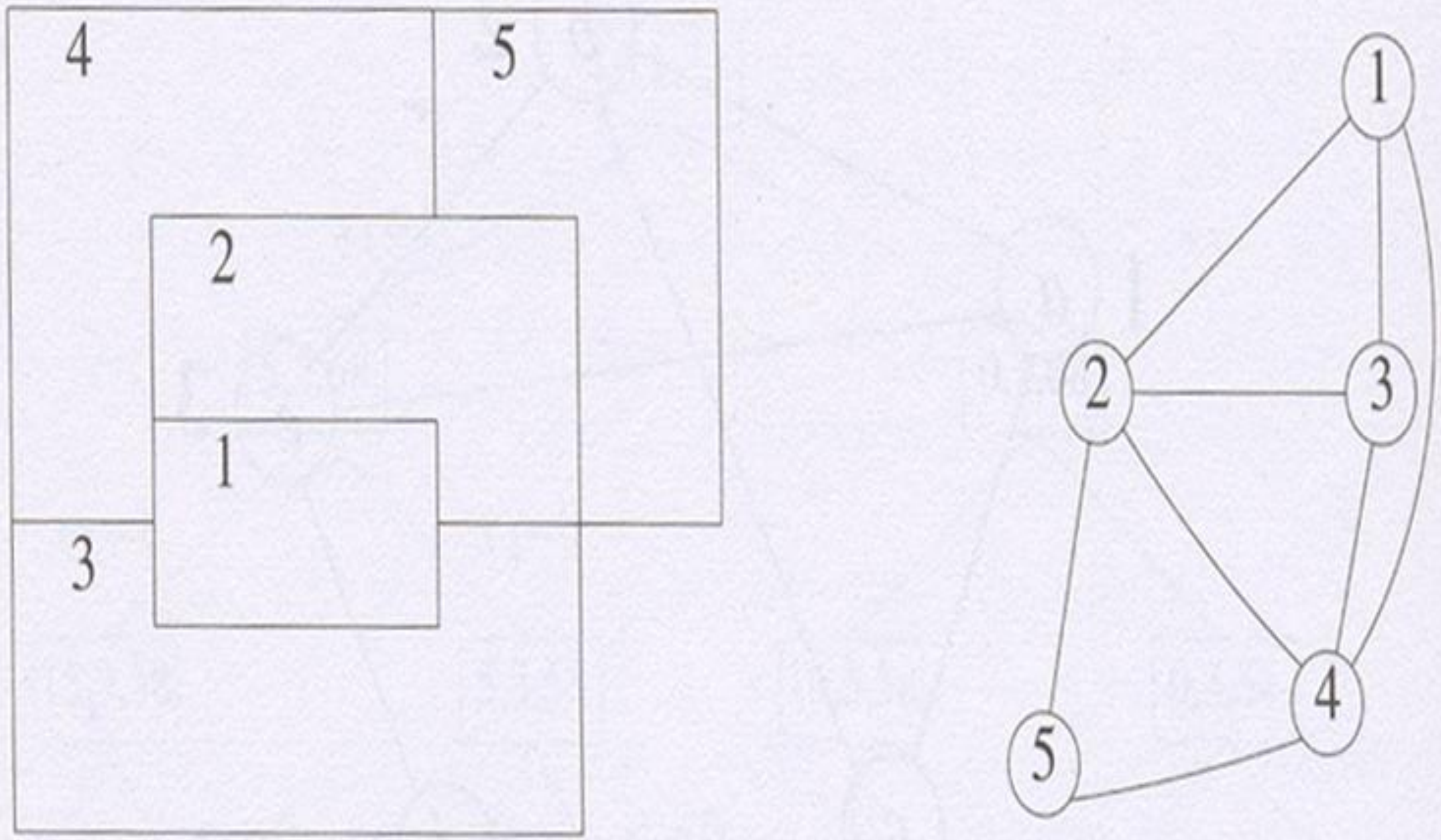
- $F^*(I) = r$
- $F\hat{}(I) = 2$
- $\frac{|F*(I) - F\hat{}(I)|}{F*(I)} = \frac{r-2}{r} = 1 - \frac{2}{r}$
- It approaches 1 as r becomes too large.
- So,
- $\frac{|F*(I) - F\hat{}(I)|}{F*(I)} \leq 1$
- It is 1- approximation scheme. (value of ε is 1)
- However, it is not an ε - approximation scheme for any ε < 1.

Absolute Approximation Schemes

- Absolute Approximation
- Let A be an algorithm that generates a feasible solution to every instance I of a problem P.
- Let F*(I) be the value of an optimal solution to I
- Let F^(I) be the value of the feasible solution generated by A
- A is an absolute approximation algorithm for problem P, if and only if for every instance I of P, $|F^*(I) - F\hat{}(I)| \leq k$, for some constant k.

Examples of Absolute Approximation Schemes

- Planar Graph Coloring
- There are very few NP-hard optimization problems for which polynomial time absolute approximation algorithms are known.
- One problem is that of determining the minimum number of colors needed to color a planar graph
- G = (V,E).
- It is known that every planar graph is four colorable.
- One can easily determine whether a graph is zero, one, or two colorable.



A map and its planar graph representation

- It is zero colorable iff V = 0.
- It is one colorable iff E = 0.
- It is two colorable iff it is bipartite
- Determining whether a planar graph is three colorable is NP-hard.
- However, all planar graphs are four colorable.
- An absolute approximation algorithm with

  $|F^*(I) - F\hat{}(I)| \le 1$ is easy to obtain.

```
Algorithm AColor(V, E)
// Determine an approximation to the minimum number of colors.
{
    if $V = \emptyset$ then return 0;
    else if $E = \emptyset$ then return 1;
        else if $G$ is bipartite then return 2;
            else return 4;
}
```

Algorithm Approximate coloring

Maximum Programs Stored Problem

- Assume that we have n programs and two storage devices (say disks or tapes).
- We assume the devices are disks.
- Let Zj be the amount of storage needed to store the i^th program.
- Let L be the storage capacity of each disk.
- Determining the maximum number of these n programs that can be stored on the two disks
- (without splitting a program over the disks) is NP-hard.

- Let, L = 10, n = 4 and $\{l_1, l_2, l_3, l_4\} = \{2, 4, 5, 6\}$
- Optimal Solution is – Store programs 1 and 4 on disk 1 and Store programs 2 and 3 on disk 2
- F*(I) = 4 ………… All 4 programs stored.
- Approximate Solution is store programs 1 and 2 on disk 1 and program 3 on disk 2.
- F^(I) = 3 ……….. Program 4 is not stored
- $|F^*(I) – F\hat{}(I)| \le 1$

```
Algorithm PStore(l, n, L)
// Assume that $l[i] \le l[i+1]$, $1 \le i < n$.
{
    $i := 1$;
    for $j := 1$ to 2 do
    {
        $sum := 0$; // Amount of disk $j$ already assigned
        while $(sum + l[i]) \le L$ do
        {
            write ("Store program", $i$, "on disk", $j$);
            $sum := sum + l[i]$; $i := i + 1$;
            if $i > n$ then return;
        }
    }
}
```

Algorithm Approximation algorithm to store programs

ε – Approximation Schemes

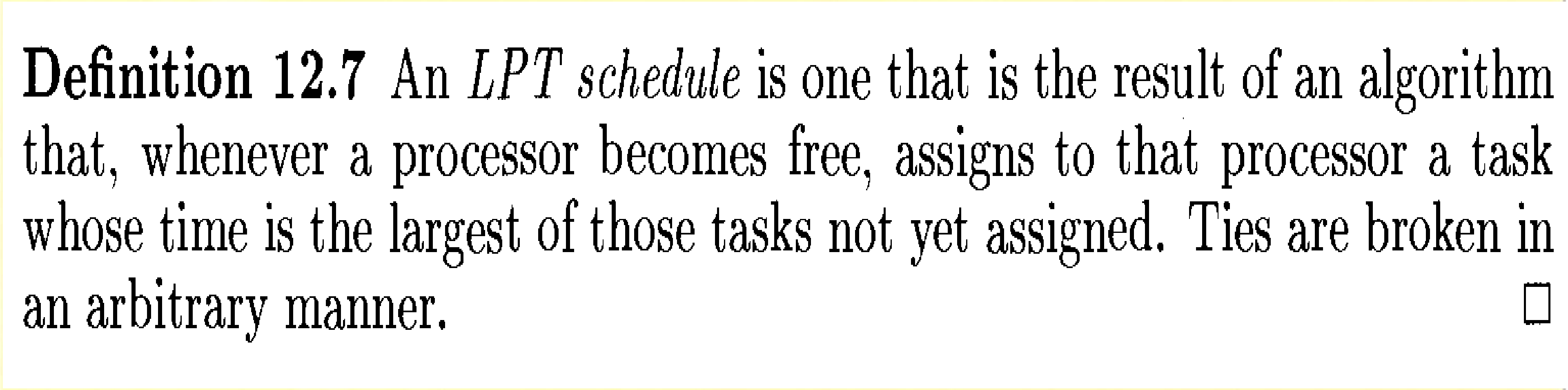
- ε - Approximation
- An ε-approximation algorithm is an $f(n)$ approximation algorithm, for which $f(n) \leq \varepsilon$, for some constant ε.

- ε - Approximation
- Let A be an algorithm that generates a feasible solution to every instance I of a problem P.
- Let F*(I) be the value of an optimal solution to I
- Let F^(I) be the value of the feasible solution generated by A
- A is an ε - approximation algorithm for problem P, if and only if for every instance I,
- $\dfrac{|F*(I) - F\wedge(I)|}{F*(I)} \leq \varepsilon$
- For $F*(I) > 0$ and $\varepsilon > 0$

Examples of ε – Approximation Schemes

- Scheduling Independent Tasks
- Obtaining minimum finish time schedules on m, $m \geq 2$, identical processors is NP Hard.
- There exists a very simple scheduling rule that generates schedules with a finish time very close to that of an optimal schedule.
- An instance I of the scheduling problem is defined by a set of n tasks t_i, $1 \leq i \leq n$,
- and m, the number of processors.

- The scheduling rule is known as the LPT (longest processing time) rule.
- An LPT schedule is a schedule that results from this rule.



Definition 12.7 An *LPT schedule* is one that is the result of an algorithm that, whenever a processor becomes free, assigns to that processor a task whose time is the largest of those tasks not yet assigned. Ties are broken in an arbitrary manner. □

Example 12.6 Let $m = 3, n = 6$, and $(t_1, t_2, t_3, t_4, t_5, t_6) = (8, 7, 6, 5, 4, 3)$. In an LPT schedule, tasks 1, 2, and 3 are assigned to processors 1, 2, and 3 respectively. Tasks 4, 5, and 6 are respectively assigned to processors 3, 2, and 1. Figure 12.2 shows this LPT schedule. The finish time is 11. Since $\sum t_i/3 = 11$, the schedule is also optimal. □

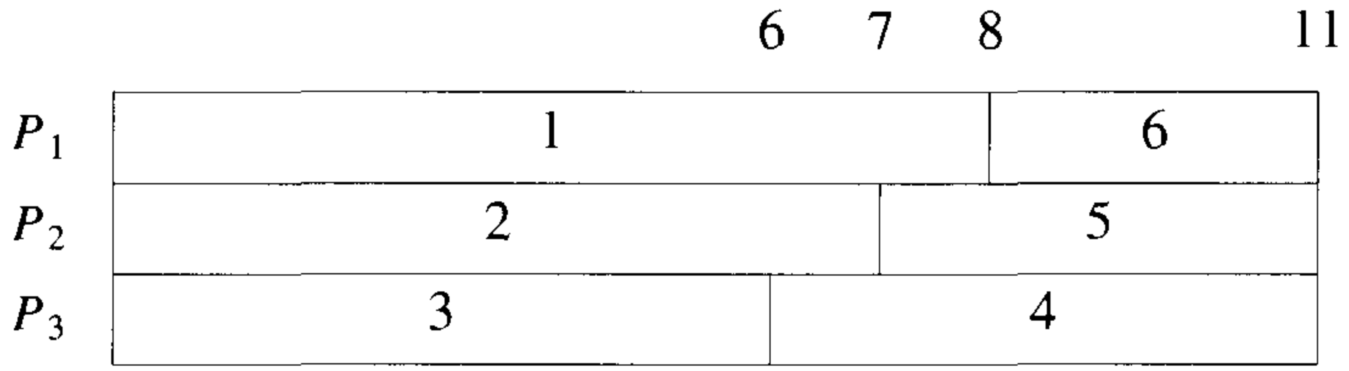
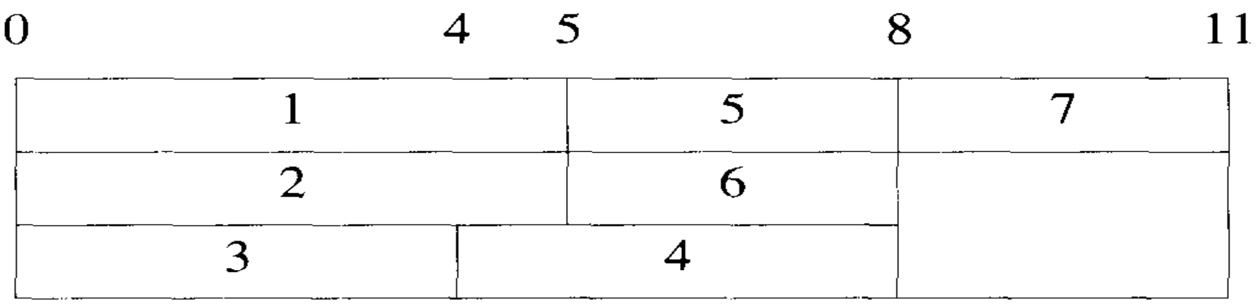
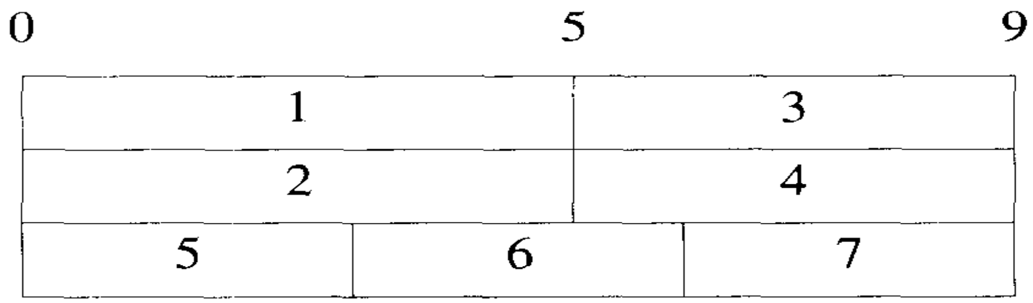


Figure 12.2 LPT schedule for Example 12.6

Example 12.7 Let $m = 3$, $n = 7$, and $(t_1, t_2, t_3, t_4, t_5, t_6, t_7) = (5, 5, 4, 4, 3, 3, 3)$. Figure 12.3(a) shows the LPT schedule. This has a finish time of 11. Figure 12.3(b) shows an optimal schedule. Its finish time is 9. Hence, for this instance $|F^*(I) - \hat{F}(I)|/F^*(I) = (11 - 9)/9 = 2/9$. □



(a) LPT schedule



(b) Optimal schedule

Figure 12.3 LPT and optimal schedules for Example 12.7

Bin Packing

- In this problem we are given n objects that have to be placed in bins of equal capacity L.
- Object i requires lj units of bin capacity.
- The objective is to determine the minimum number of bins needed to accommodate all n objects.
- No object may be placed partly in one bin and partly in another.

Example 12.9 Let $L = 10$, $n = 6$, and $(l_1, l_2, l_3, l_4, l_5, l_6) = (5, 6, 3, 7, 5, 4)$. Figure 12.5 shows a packing of the six objects using only three bins. Numbers in bins are object indices. Obviously, at least three bins are needed. □

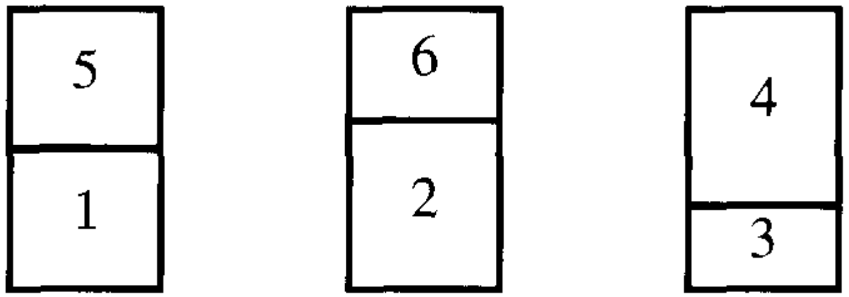


Figure 12.5 Optimal packing for Example 12.9

Polynomial Time Approximation Schemes

- Polynomial Time Approximation Scheme
- An approximation scheme is a polynomial time approximation scheme
- if and only if for every fixed $\varepsilon > 0$,

  it has a computing time that is polynomial in terms of the problem size n

Polynomial Time Approximation Schemes

- A family of algorithms that can achieve any constant approximation in polynomial time is called a polynomial-time approximation scheme or PTAS.

Example - Polynomial Time Approximation Schemes

- Scheduling Independent Tasks
- A polynomial time approximation scheme can be used for scheduling independent tasks.
- Let k be some specified fixed integer.
- Obtain an optimal schedule for the k longest tasks.
- Schedule the remaining n-k tasks using the LPT rule.

Example 12.12 Let $m = 2$, $n = 6$, $(t_1, t_2, t_3, t_4, t_5, t_6) = (8, 6, 5, 4, 4, 1)$, and $k = 4$. The four longest tasks have task times 8, 6, 5, and 4 respectively. An optimal schedule for these has finish time 12 (Figure 12.8(a)). When the remaining two tasks are scheduled using the LPT rule, the schedule of Figure 12.8(b) results. This has finish time 15. Figure 12.8(c) shows an optimal schedule. This has finish time 14. □

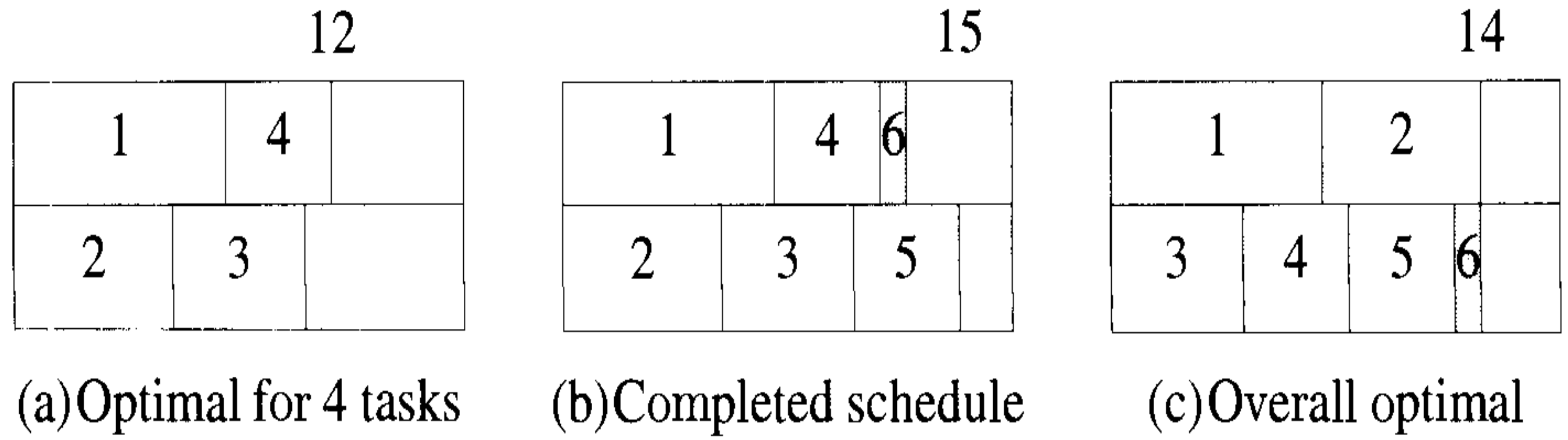
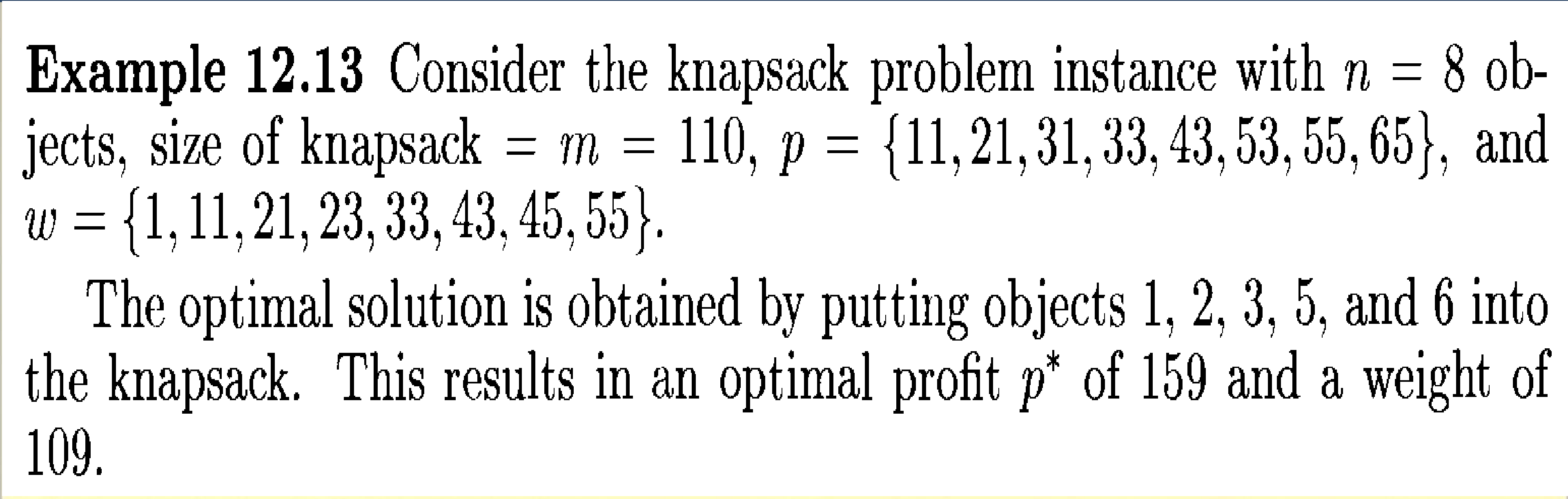


Figure 12.8 Using the approximation schedule with $k = 4$

0/1 Knapsack



Example 12.13 Consider the knapsack problem instance with $n = 8$ objects, size of knapsack $= m = 110$, $p = \{11, 21, 31, 33, 43, 53, 55, 65\}$, and $w = \{1, 11, 21, 23, 33, 43, 45, 55\}$.

The optimal solution is obtained by putting objects 1, 2, 3, 5, and 6 into the knapsack. This results in an optimal profit p^* of 159 and a weight of 109.

- Get pi/wi ratio
- Add some initial elements having good pi/wi ratio
- If any space remaining in knapsack select other elements by considering k as size of elements to be added

	I1	I2	I3	I4	I5	I6	I7	I8
P(i)	11	21	31	33	43	53	55	65
W(i)	1	11	21	23	33	43	45	55
(pi/wi)	11	1.90	1.47	1.43	1.30	1.23	1.22	1.18
Order	I1	I2	I3	I4	I5	I6	I7	I8
Add items		I1	I2	I3	I4	I5		

Profit = 139 Weight = 89

Works in Polynomial Time

Fully Polynomial Time Approximation Schemes

- Fully Polynomial Time Approximation Scheme
- An approximation scheme is a fully polynomial time approximation scheme
- If it has a computing time that is polynomial in terms of the problem size n and 1/ε

Fully Polynomial Time Approximation Scheme

- The approximation algorithms and schemes we have seen so far are particular to the problem considered.
- There is no set of well-defined techniques that we can use to obtain such algorithms.
- The heuristics used depend very much on the particular problem being solved.
- approach is identical to the dynamic programming solution methodology.

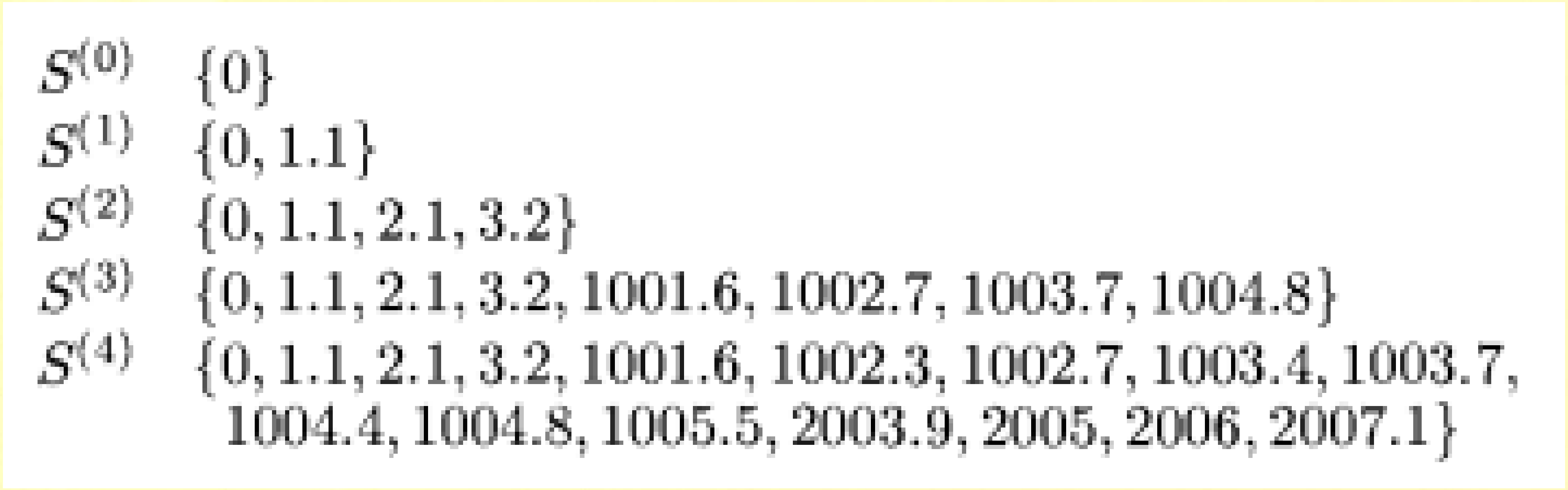
- A family of algorithms that can achieve any approximation $\varepsilon > 0$ in time polynomial in both $1/\varepsilon$ and n is called a **fully polynomial time approximation scheme** or FPTAS.
- Fully polynomial-time approximation schemes do not appear to exist for many problems,

Rounding

- The aim of rounding is to start from a problem instance I and to transform it to another problem instance I’ that is easier to solve.
- This transformation is carried out in such a way that the optimal solution value of I’ is close to the optimal solution value of I.
- The value of ε, which represent the bound on the fractional difference between the exact and approximate solution values is provided.

- $\frac{|F*(I)-F^{\wedge}(I')|}{F*(I)} \leq \varepsilon$
- Where
- F*(I) represent optimal solution value of I
- And F^(I) represent optimal solution value of I’

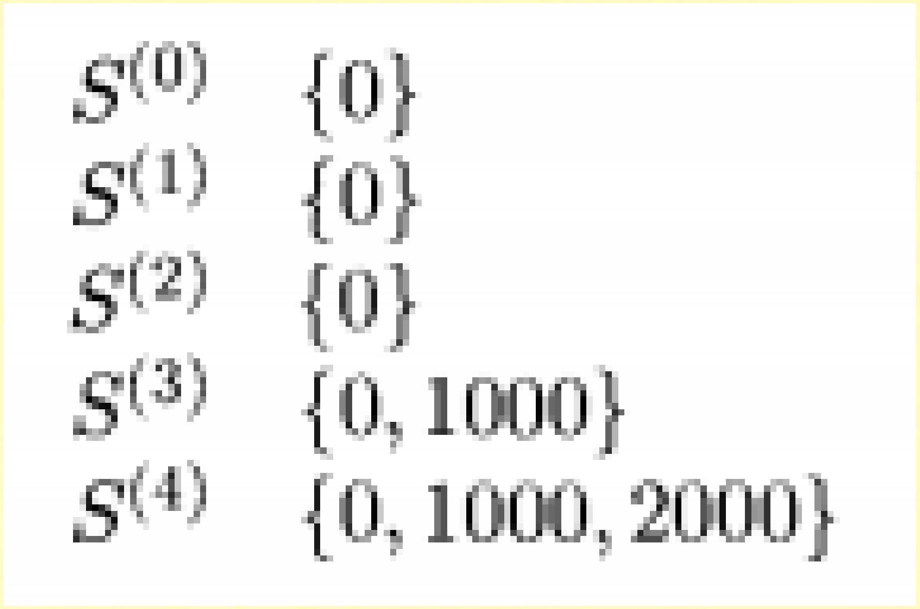
- Consider an instance of 0/1 Knapsack
- $\{p_1, p_2, p_3, p_4\} = \{1.1, 2.1, 1001.6, 1002.3\}$
- Solving it consumes too much time.



$S^{(0)}$	$\{0\}$
$S^{(1)}$	$\{0, 1.1\}$
$S^{(2)}$	$\{0, 1.1, 2.1, 3.2\}$
$S^{(3)}$	$\{0, 1.1, 2.1, 3.2, 1001.6, 1002.7, 1003.7, 1004.8\}$
$S^{(4)}$	$\{0, 1.1, 2.1, 3.2, 1001.6, 1002.3, 1002.7, 1003.4, 1003.7, 1004.4, 1004.8, 1005.5, 2003.9, 2005, 2006, 2007.1\}$

- $F^*(I) = 2007.1$

- Round the values and transform to I'
- $\{q_1, q_2, q_3, q_4\} = \{0, 0, 1000, 1000\}$



$S^{(0)}$	$\{0\}$
$S^{(1)}$	$\{0\}$
$S^{(2)}$	$\{0\}$
$S^{(3)}$	$\{0, 1000\}$
$S^{(4)}$	$\{0, 1000, 2000\}$

- F^(I) = 2000
- $|F*(I) - F^\wedge(I')| = 2007.1 - 2000 = 7.1$
- $\frac{|F*(I) - F^\wedge(I')|}{F*(I)} \leq 0.007$

- I’ can be solved in 1/4th time compared to I.
- Inaccuracy is less than 0.7%

Probabilistically Good Algorithms

Probabilistically Good Algorithms

- The approximation algorithms seen in preceding sections had the property that their worst-case performance could be bounded by some constants.
- k in the case of an absolute approximation and ε in the case of ε approximation algorithms

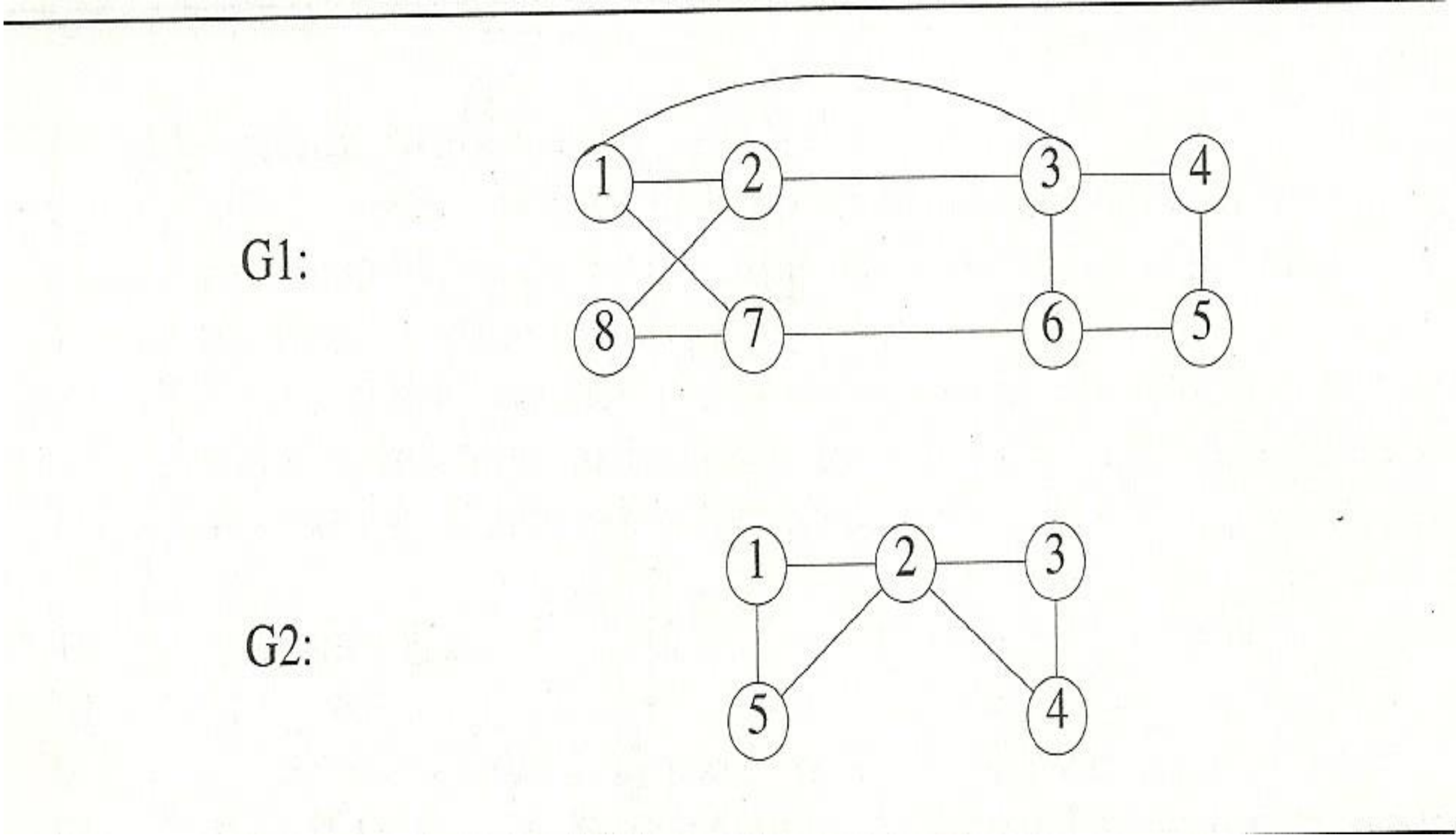
- The requirement of bounded performance tends to categorize other algorithms
- that usually work well as being bad.
- Some algorithms with unbounded performance may in fact almost always either solve the problem exactly
- or generate a solution that is exceedingly close in value to the value of an optimal solution

- Such algorithms are good in a probabilistic sense.
- If we pick a problem instance I at random,
- then there is a very high probability that the algorithm will generate a very good approximate solution.

Finding Hamiltonian Cycle in Undirected Graph

- Let G = (V, E) be a connected graph with n vertices.
- A Hamiltonian cycle (suggested by Sir William Hamilton) is a
- round-trip path along n edges of G
- that visits every vertex once and returns to its starting position.

- Hamiltonian cycle begins at some vertex
- and the vertices of G are visited in the order $v_1, v_2, .. . , v_{n+1}$
- then the edges (v_i, v_{i+i}) are in E, $1 <i <n$,
- and the vi are distinct except for v_1 and v_{n+1}
- which are same.



- The graph G1 contains the Hamiltonian cycle 1, 2, 8, 7, 6, 5, 4, 3, 1.
- The graph G2 contains no Hamiltonian cycle.

- This algorithm proceeds as follows.
- First, an arbitrary vertex (say vertex 1) is chosen as the start vertex.
- The algorithm maintains a simple path P starting from vertex 1 and ending at vertex k.
- Initially P is a trivial path with k = 1 there are no edges in P.
- At each iteration of the algorithm an attempt is made to increase the length of P.
- This is done by considering an edge(k, j)incident to the end point k of P.
- When edge(k, j) is being considered one of three possibilities exist

- 1] j = 1 and path P includes all the vertices of the graph.
- In this case a Hamiltonian cycle has been found and the algorithm terminates.

- 2] j is not on the path P. In this case the length of path P is increased
- by adding (k, j) to it. Then j becomes the new end point of P.

- 3] j is already on path P. Now there is an unique edge e = (j, m) in P such that
- the deletion of e from and the inclusion of (k, j) to P result in a simple path.
- Then e is deleted and (k, j) is added to P.
- P is now a simple path with end point m

Analysis

- The algorithm is constrained so that case 3 does not generate two paths of the same length having the same end point.
- With a proper choice of data representations this algorithm can be implemented to run in time $O(n^2)$
- Where n is the number of vertices in graph G
- This algorithm does not always find a Hamiltonian cycle in a graph that contains such a cycle.
- It works fine with certain probability.

Thank You

Email:- suhas.bhagate@gmail.com