

Approximation

What for?

— Dealing with HARD problems

Getting around NP-completeness

👉 If N is small, even $O(2^N)$ is acceptable

👉 Solve some important special cases in polynomial time

✓👉 Find *near-optimal* solutions in polynomial time
— *approximation* algorithm

Approximation Ratio

【Definition】 An algorithm has an *approximation ratio* of $\rho(n)$ if, for any input of size n , the cost C of the solution produced by the algorithm is within a factor of $\rho(n)$ of the cost C^* of an optimal solution:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$$

If an algorithm achieves an approximation ratio of $\rho(n)$, we call it a *$\rho(n)$ -approximation algorithm*.

【Definition】 An *approximation scheme* for an optimization problem is an approximation algorithm that takes as input not only an instance of the problem, but also a value $\varepsilon > 0$ such that for any fixed ε , the scheme is a **$(1 + \varepsilon)$ -approximation algorithm**.

We say that an approximation scheme is a *polynomial-time approximation scheme (PTAS)* if for any fixed $\varepsilon > 0$, the scheme runs in time polynomial in the size n of its input instance.

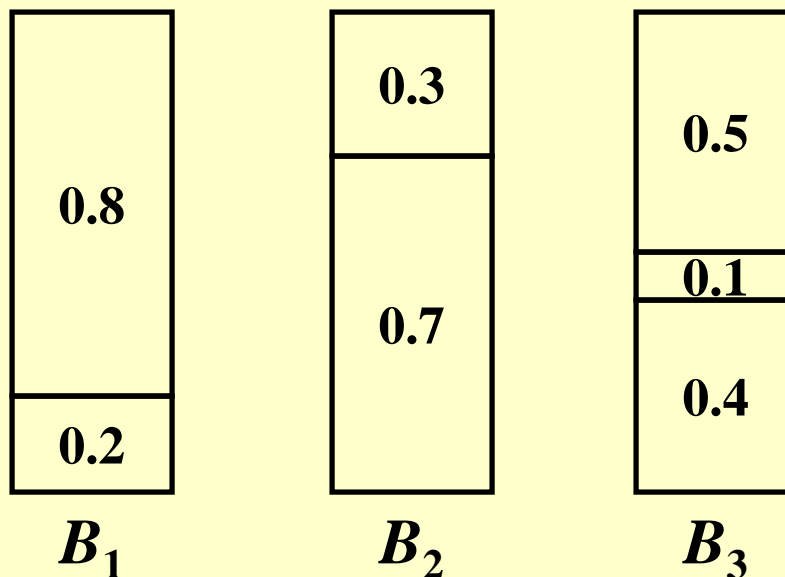
$$O(n^{2/\varepsilon}) \quad O((1/\varepsilon)^2 n^3)$$

*fully polynomial-time
approximation scheme
(FPTAS)*

Approximate Bin Packing

Given N items of sizes S_1, S_2, \dots, S_N , such that $0 < S_i \leq 1$ for all $1 \leq i \leq N$. Pack these items in the **fewest** number of bins, each of which has **unit capacity**.

[[Example]] $N = 7$; $S_i = 0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8$



An Optimal Packing



❖ Next Fit

```
void NextFit ( )
{  read item1;
  while ( read item2 ) {
    if ( item2 can be packed in the same bin as item1 )
      place item2 in the bin;
    else
      create a new bin for item2;
    item1 = item2;
  } /* end-while */
}
```

【Theorem】 Let M be the optimal number of bins required to pack a list I of items. Then *next fit* never uses more than $2M - 1$ bins. There exist sequences such that *next fit* uses $2M - 1$ bins.

A simple proof for Next Fit:

If Next Fit generates $2M$ (or $2M+1$) bins, then the optimal solution must generate at least $M+1$ bins.

Let $S(B_i)$ be the size of the i th bin. Then we must have:

$$S(B_1) + S(B_2) > 1$$

$$S(B_3) + S(B_4) > 1$$

.....

$$S(B_{2M-1}) + S(B_{2M}) > 1$$

$$\Rightarrow \sum_{i=1}^{2M} S(B_i) > M$$

The optimal solution needs at least $\lceil \text{total size of all the items} / 1 \rceil$ bins

$$\Rightarrow \lceil \text{total size of all the items} / 1 \rceil = \left\lceil \sum_{i=1}^{2M} S(B_i) \right\rceil \geq M + 1$$

❖ First Fit

```

void FirstFit ( )
{ while ( read item ) {
    scan for the first bin that is large enough for item;
    if ( found )
        place item in that bin;
    else
        create a new bin for item;
} /* end-while */
}

```

Can be implemented
in $O(N \log N)$

【Theorem】 Let M be the optimal number of bins required to pack a list I of items. Then *first fit* never uses more than $17M / 10$ bins. There exist sequences such that *first fit* uses $17(M - 1) / 10$ bins.

❖ Best Fit

Place a new item in the **tightest** spot among all bins.

$T = O(N \log N)$ and bin no. $\leq 1.7M$

〔Example〕 $S_i = 0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8$

Next Fit

First Fit

Best Fit

Discussion 14: Please show the results.

〔Example〕 $S_i = 1/7+\varepsilon, 1/7+\varepsilon, 1/7+\varepsilon, 1/7+\varepsilon, 1/7+\varepsilon, 1/7+\varepsilon,$
 $1/3+\varepsilon, 1/3+\varepsilon, 1/3+\varepsilon, 1/3+\varepsilon, 1/3+\varepsilon, 1/3+\varepsilon,$
 $1/2+\varepsilon, 1/2+\varepsilon, 1/2+\varepsilon, 1/2+\varepsilon, 1/2+\varepsilon, 1/2+\varepsilon$

where $\varepsilon = 0.001$.



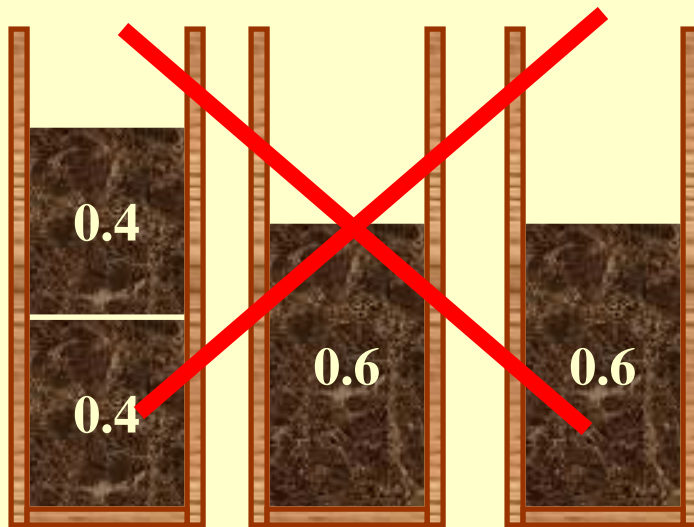
The optimal solution requires **6** bins.

However, all the three on-line algorithms require **10** bins.

👉 On-line Algorithms

Place an item before processing the next one, and can **NOT** change decision.

【Example】 $S_i = 0.4, 0.4, 0.6, 0.6$



You never know
when the input might end.
No on-line algorithm
can always give
an optimal solution.

【Theorem】 There are inputs that force any on-line bin-packing algorithm to use at least **5/3** the optimal number of bins.



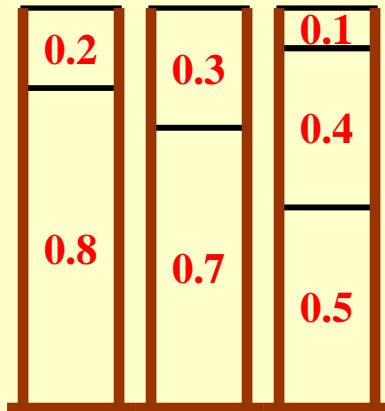
☞ Off-line Algorithms

View the **entire** item list before producing an answer.

Trouble-maker: The large items

Solution: Sort the items into non-increasing sequence of sizes. Then apply first (or best) fit – *first* (or *best*) *fit decreasing*.

[[Example]] $S_i = 0.8, 0.7, 0.5, 0.4, 0.3, 0.2, 0.1$



【Theorem】 Let M be the optimal number of bins required to pack a list I of items. Then *first fit decreasing* never uses more than $11M / 9 + 6/9$ bins. There exist sequences such that *first fit decreasing* uses $11M / 9 + 6/9$ bins.

Simple greedy heuristics can give good results.

The Knapsack Problem — fractional version

A knapsack with a capacity M is to be packed. Given N items. Each item i has a weight w_i and a profit p_i . If x_i is the percentage of the item i being packed, then the packed profit will be $p_i x_i$.

An **optimal packing** is a feasible one with **maximum profit**. That is, we are supposed to find the values of x_i such that $\sum_{i=1}^n p_i x_i$ obtains its maximum under the constraints

$$\sum_{i=1}^n w_i x_i \leq M \quad \text{and} \quad x_i \in [0, 1] \text{ for } 1 \leq i \leq n$$

Q: What must we do in each stage?

A: Pack one item into the knapsack.

Q: On which criterion shall we be greedy?

① maximum profit ② minimum weight

③ maximum profit density p_i / w_i

Example:

$n = 3, M = 20,$

$(p_1, p_2, p_3) = (25, 24, 15)$

$(w_1, w_2, w_3) = (18, 15, 10)$

Solution is...?

(0, 1, 1/2)

P = 31.5

The Knapsack Problem — 0-1 version

NP-hard

or 0

Example: $n = 5, M = 11,$ $(p_1, p_2, p_3, p_4, p_5) = (1, 6, 18, 22, 28)$ $(w_1, w_2, w_3, w_4, w_5) = (1, 2, 5, 6, 7)$

The greedy solution is

Solution is...? $(0, 0, 1, 1, 0)$
 $P = 40$ $(1, 1, 0, 0, 1)$
 $P = 35$

What if we are greedy on taking the **maximum profit** *or* **profit density**?

The approximation ratio is **2**.

Proof: $p_{\max} \leq P_{\text{opt}} \leq P_{\text{frac}}$

$$p_{\max} \leq P_{\text{greedy}} \quad \longrightarrow \quad P_{\text{opt}} / P_{\text{greedy}} \leq 1 + p_{\max} / P_{\text{greedy}} \leq 2$$

$$P_{\text{opt}} \leq P_{\text{greedy}} + p_{\max}$$

👉 A Dynamic Programming Solution

$W_{i,p}$ = the minimum weight of a collection from $\{1, \dots, i\}$ with total profit being exactly p

① take i : $W_{i,p} = w_i + W_{i-1,p-p_i}$

② skip i : $W_{i,p} = W_{i-1,p}$

③ impossible to get p : $W_{i,p} = \infty$

$$W_{i,p} = \begin{cases} \infty & i = 0 \\ W_{i-1,p} & p_i > p \\ \min\{ W_{i-1,p}, w_i + W_{i-1,p-p_i} \} & \text{otherwise} \end{cases}$$

$i = 1, \dots, n; p = 1, \dots, n p_{max} \rightarrow O(n^2 p_{max})$

👉 What if p_{max} is LARGE?

Item	Profit	Weight
1	134,221	1
2	656,342	2
3	1,810,013	5
4	22,217,800	6
5	28,343,199	7

$M = 11$

Item	Profit	Weight
1	2	1
2	7	2
3	19	5
4	223	6
5	284	7

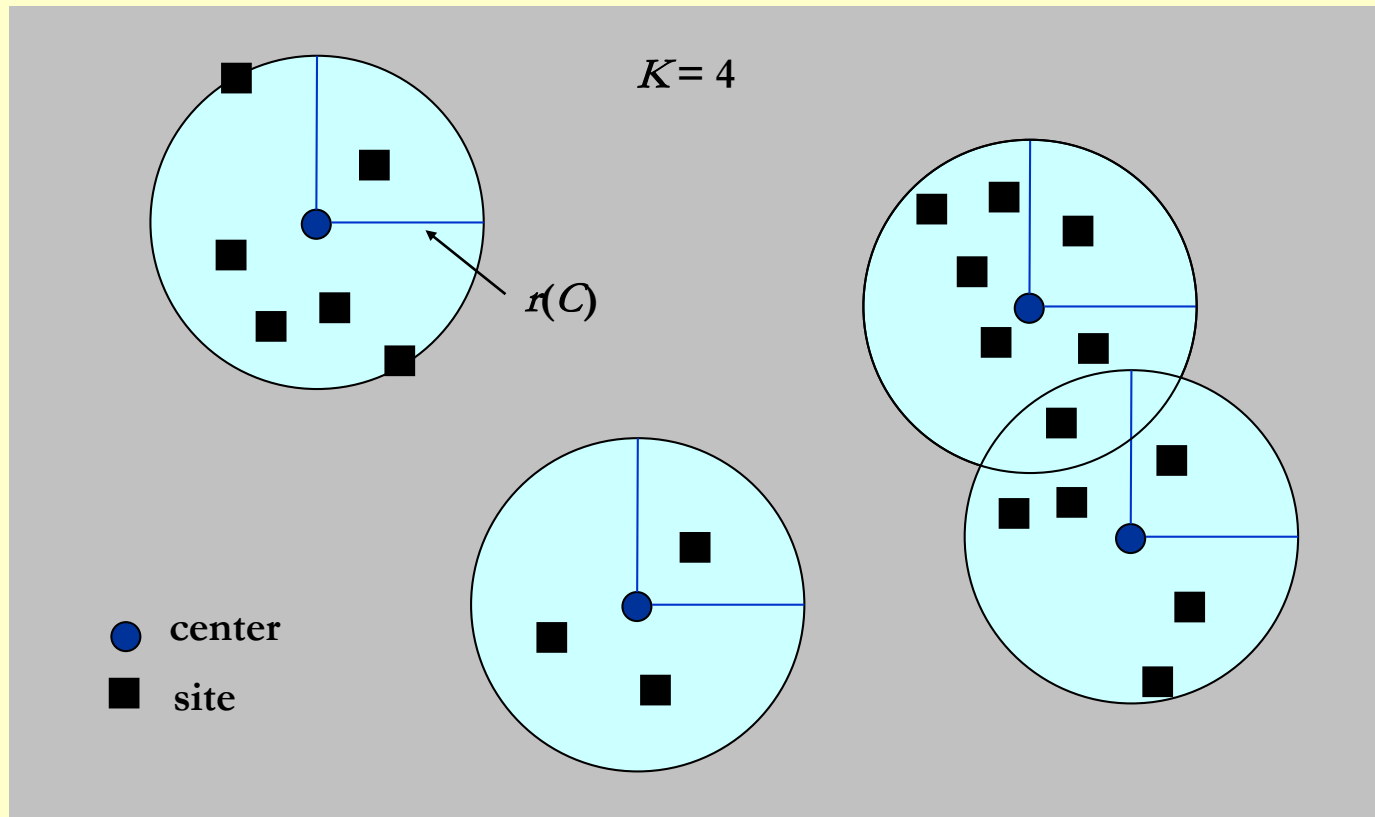
$M = 11$

👉 Round all profit values up to lie in smaller range!

$$(1+\varepsilon) P_{\text{alg}} \leq P \quad \text{for any feasible solution } P$$

precision parameter

The K -center Problem



Input: Set of n sites s_1, \dots, s_n

Center selection problem: Select K centers C so that the maximum distance from a site to the nearest center is minimized.

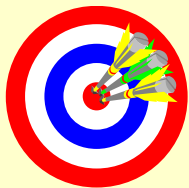
👉 What is a *distance*?

- ✓ $\text{dist}(x, x) = 0$ (identity)
- ✓ $\text{dist}(x, y) = \text{dist}(y, x)$ (symmetry)
- ✓ $\text{dist}(x, y) \leq \text{dist}(x, z) + \text{dist}(z, y)$ (triangle inequality)

$$\text{dist}(s_i, C) = \min_{c \in C} \text{dist}(s_i, c)$$

= distance from s_i to the closest center

$$r(C) = \max_i \text{dist}(s_i, C) = \text{smallest covering radius}$$

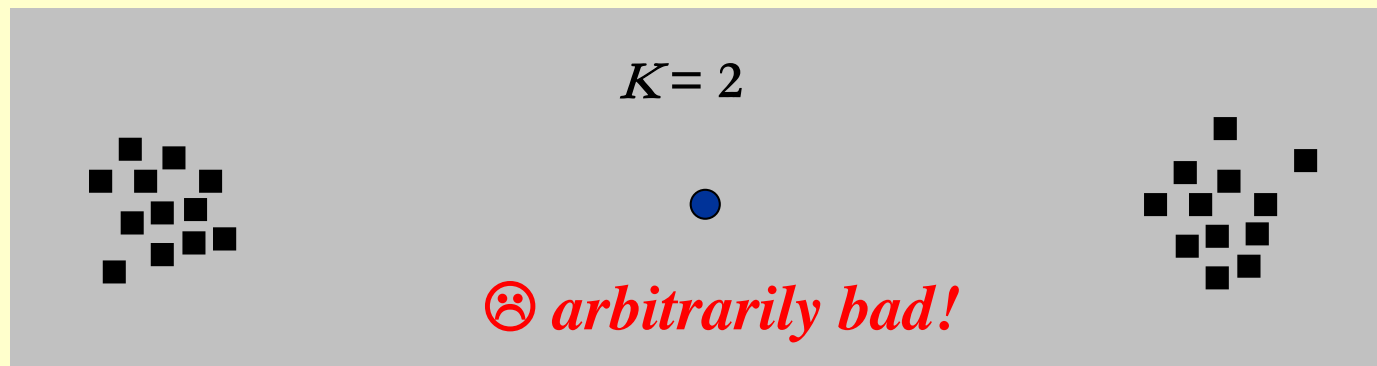
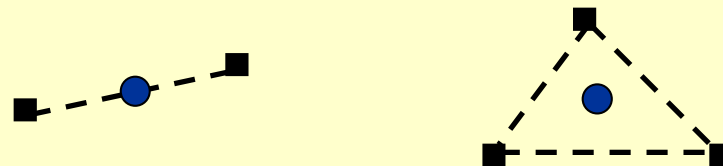


Find a set of centers C that minimizes $r(C)$,
subject to $|C| = K$.

Number of candidate centers = ∞

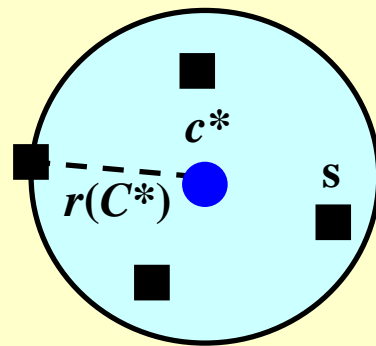
👉 A Greedy Solution

Put the first center at the *best possible* location for a single center, and then keep adding centers so as to *reduce the covering radius* each time by as much as possible.



👉 A Greedy Solution — try again ...

What if we know that $r(C^*) \leq r$ where C^* is the optimal solution set?



Discussion 15:

Take s to be the center, how can we select r so that s can cover all the sites that are covered by c^* ?

```

Centers Greedy-2r ( Sites S[ ], int n, int K, double r )
{
  Sites S'[ ] = S[ ]; /* S' is the set of the remaining sites */
  Centers C[ ] =  $\emptyset$ ;
  while ( S'[ ] !=  $\emptyset$  ) {
    Select any s from S' and add it to C;
    Delete all s' from S' that are at  $\text{dist}(s', s) \leq 2r$ ;
  } /* end-while */
  if ( |C|  $\leq$  K ) return C;
  else ERROR(No set of K centers with covering radius at most r);
}

```

【Theorem】 Suppose the algorithm selects more than K centers. Then for any set C^* of size at most K , the covering radius is $r(C^*) > r$.

Do we really know $r(C^*)$?





Binary search for r




$$0 < r \leq r_{max}$$

Guess: $r = (0 + r_{max}) / 2$



 {

 Yes: K centers found with $2r$ 

 or

 No: r is too small 

$$r_0 < r \leq r_1 \quad r = (r_0 + r_1) / 2$$

 Solution radius = $2r_1$ ——— **2**-approximation



A smarter solution — be far away

```
Centers Greedy-Kcenter ( Sites S[ ], int n, int K )
{
  Centers C[ ] =  $\emptyset$ ;
  Select any s from S and add it to C;
  while ( |C| < K ) {
    Select s from S with maximum dist(s, C);
    Add s it to C;
  } /* end-while */
  return C;
}
```

【Theorem】 The algorithm returns a set C of K centers such that $r(C) \leq 2r(C^*)$ where C^* is an optimal set of K centers.

—— **2**-approximation

👉 Is there a hope of a $3/2$ -approximation? Or $4/3$?

【Theorem】 Unless $P = NP$, there is **no** ρ -approximation for center-selection problem for any $\rho < 2$.

Sketch of the proof: By contradiction.

If we can obtain a $(2-\epsilon)$ -approximation in polynomial time, then we can solve DOMINATING-SET (which is NP-complete) in polynomial time.

Dominating set problem has a solution of size K iff there exists K centers C^* with $r(C^*) = 1$.

Then a $(2-\epsilon)$ -approximation must give the optimal solution since all the distances involved are integers.

Three aspects to be considered:

A: Optimality -- *quality of a solution*

B: Efficiency -- *cost of computations*

C: All instances

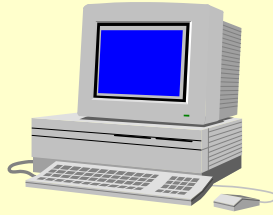
Researchers are working on

A+C: Exact algorithms for all instances

A+B: Exact and fast algorithms for special cases

B+C: Approximation algorithms

*Even if $P=NP$, still we cannot guarantee **A+B+C** .*



Research Project 6

Texture Packing (26)

Texture Packing is to pack multiple rectangle shaped textures into one large texture. The resulting texture must have a given width and a **minimum height.**

You are to design and analyze an approximation algorithm that runs in polynomial time.

**Detailed requirements can be downloaded from
<https://pintia.cn/>**

Reference:

Data Structure and Algorithm Analysis in C (2nd Edition):
Ch.10, p.359-366; *M.A.Weiss 著、陈越改编, 人民邮电出版社, 2005*

Introduction to Algorithms, 3rd Edition: Ch.35, p.1106-1140; *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. The MIT Press. 2009*