Lecture11_Approximation

1.introduction

Getting around NP-Completeness

 If N is small, even O(2N) is acceptable

 Solve some important **special cases** in polynomial time (加一些限制)

 Find **near-optimal** solutions in polynomial time

——approximation algorithm （和最优算法进行比较）

Approximation Ratio

definition：

【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:

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

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

C*： min cost

approximation scheme

definition

【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+ ε)-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.

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

$O((1/\varepsilon)^2 n^3)$:fully polynomial-time approximation scheme (FPTAS) 因为 1/ e 和 n 都是线性的

2.bin packing

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

NP Hard !!!

can we pack them ? **NPC**

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 */
}
```

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.) 严格的上界

说明，p(n) = 2;

证明：

A simple proof for Next Fit:

第一步，假设

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 ith bin. Then we must have:

$$S(B_1) + S(B_2) > 1$$
$$S(B_3) + S(B_4) > 1$$
$$\cdots\cdots$$
$$S(B_{2M-1}) + S(B_{2M}) > 1$$

第二步，根据假设推论 最优解相邻两个bin 不可以加在一起

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

第三步，可以推出，size之和 >M 也就是>=M+1

也就是说至少需要 M+1个bin, 矛盾！！！

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

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

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 */
}
```

O(N log N)

Then first fit never uses more than 17M / 10 bins.   ratio =1.7

best fit

Place a new item in the **tightest**(放入东西后 剩余空间最小) spot among all bins.

O(N log N)bin no. <= 1.7M   ratio =1.7

eg1:最优解显然是3

$S_i = 0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8$

next fit   0.2 0.5   5

下一个能不能和前一个放一起

0.4

0.7 0.1

0.3

0.8

first fit   0.2 0.5 0.1   4

每次从头遍历找到能放的

0.4 0.3

0.7

0.8

best fit   0.2 0.5 0.1   4

所有能放的中找剩余空间最小的

0.4

0.7 0.3

0.8

eg2:竖着看，最优解是6，但是next fit, first fit ,best fit 结果都是10

【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 $\varepsilon = 0.001$.

online algorithm

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

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.

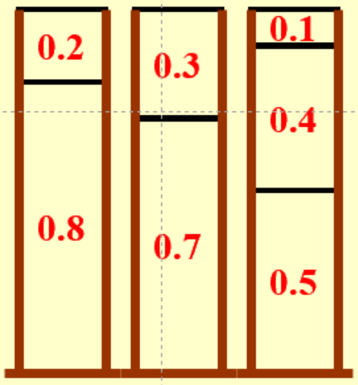
off - line algorithms

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.

3.The Knapsack Problem — fractional version

背包问题

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 constrains

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

Q: what must we do in each stage?

A: Pack one item into the knapsack.

Q: On which criterion shall we be greedy?

A: maximum profit density pi / wi

过程：计算所有item 的 density 尽可能选择最大的

The Knapsack Problem — 0-1 version

take it or not take

Proof: ① $P_{max} \le \underline{P_{opt} \le P_{frac}}$

② $P_{max} \le P_{greedy}$

③ $P_{\text{profit density}} \le P_{greedy}$

$\rightarrow P_{opt} \le \underbrace{P_{frac} \le P_{\text{profit density}} + P_{max}}_{\text{最后一个 item} < P_{max}.} \le P_{greedy} + P_{max}$

$$\Rightarrow P_{opt} / P_{greedy} \le 1 + p_{max} / P_{greedy} \le 2$$

A Dynamic Programming Solution

考虑三个部分：subset profit weight

$W_{i,p}$ = the minimum weight of a collection from $\{1, \ldots, i\}$ 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 : $W_{i,p} = \infty$

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

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

worst case 每一个p = p max

不是polynomial time,因为 p max不是定值 而是指数级的O(2^sizeof(pmax))

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

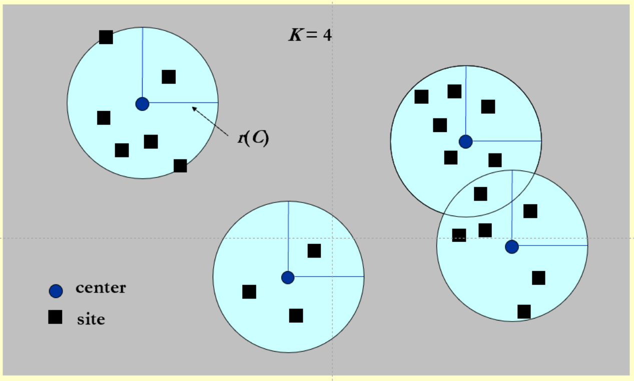
M = 11

统一缩小，减小数据范围

Round all profit values up to lie in smaller range!

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

4.K-center problem



Input: Set of n sites $s_1, \ldots, s_n$

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

补充定义：

✓ $\text{dist}(x, x) = 0$ (identity)

✓ $\text{dist}(x, y) = \text{dist}(y, x)$ (symmetry)

✓ $\text{dist}(x, y) \le \text{dist}(x, z) + \text{dist}(z, y)$ (triangle inequality)

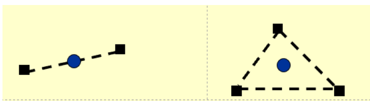
$$\text{dist}(s_i, C) = \min_{c \in C} \text{dist}(s_i, c)$$
$$= \text{distance from } s_i \text{ to the closest center}$$

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

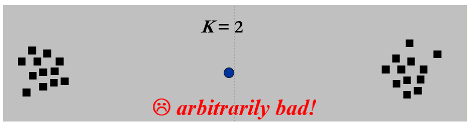
number of candidate centers are infinite

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.



第二个点怎么放？应该左右各一个center合适，这个算法有问题



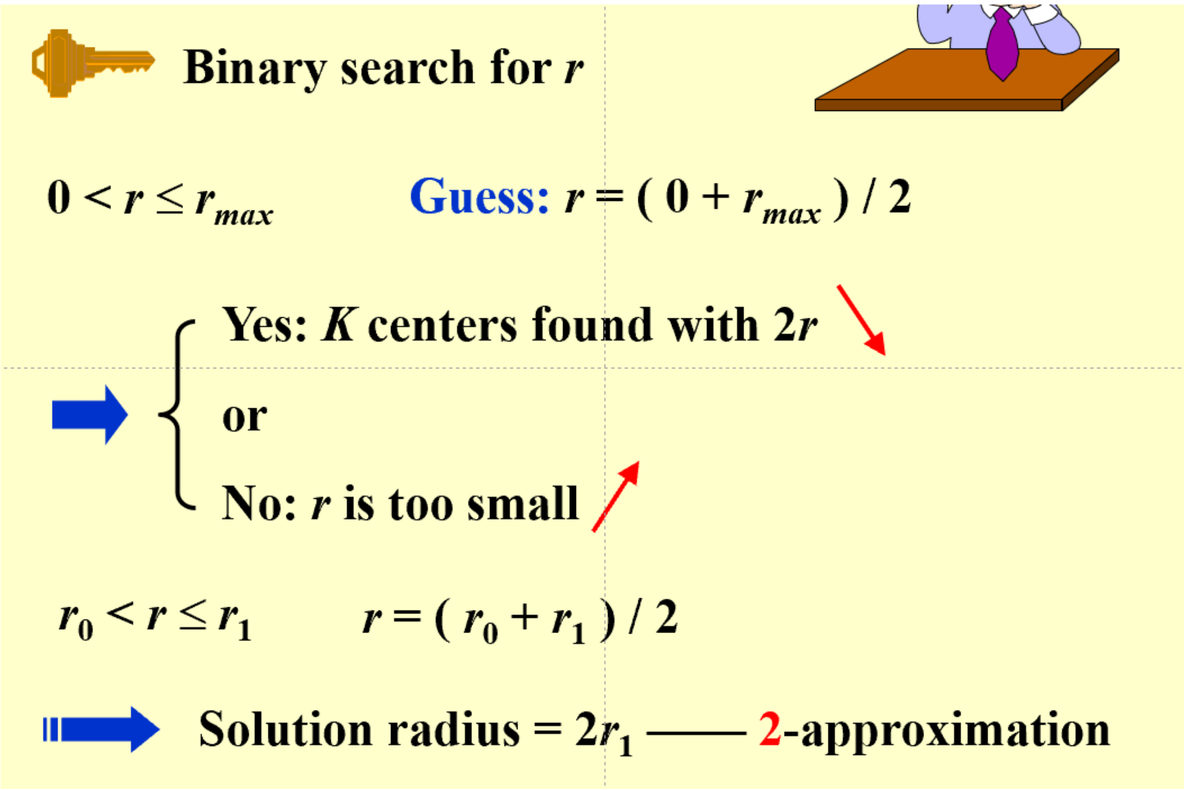
A Greedy method try again

如果已知 r(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[ ] = ;
    while ( S’[ ] !=  ) {
        Select any s from S’ and add it to C;
        Delete all s’ from S’ that are at dist(s’, s)  2r;
    } /* end-while */
    if ( |C|  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.

怎么找r?




```
Centers  Greedy-Kcenter ( Sites S[ ], int n, int K )
{   Centers  C[ ] = ;
    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) <= 2r(C *) where C* is an optimal set of K centers.

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

总结：

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 .**