approximation ratio.

$\rho(n) \geq \max\left\{ \frac{C}{C^*}, \frac{C^*}{C} \right\}$, C: cost of n. C^*: n 的 optimal 做法 的 cost

we call it $\rho(n)$-approximation algorithm.

$\rho(n)$ 要取尽量接近 1 的越好. ($\rho(n)=2$ 可以的话 取 3 也可以, 但无意义)

approximation scheme. $(1+\varepsilon)$-approximate algorithm

PTAS: polynomial-time approximation scheme 多项式时间的近似模式

对 fixed ε 都 run polynomial time

如 $O(n^{\frac{2}{\varepsilon}})$. $O((\frac{1}{\varepsilon})^2 n^3)$ → FPTAS. fully PTAS

ε 很小, $O(n^{\frac{2}{\varepsilon}})$ 会很快.

Binpack Problem NP Hard

Input: n items with size $s_1, s_2, \cdots, s_n$ $(0 < s_i \leq 1)$

Output: packing the items using fewest bins with unit capacity.

```
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.

1. Next Fit

$B_1, B_2, \cdots, B_k$, 每个 B_i 不一定能装满

但 $s(B_1) + s(B_2) = 1$ $\Rightarrow s(B_1) + 2\sum_{i=2}^{k-1} B_i + B_k > k-1$

$s(B_2) + s(B_3) = 1$ $\quad\quad \therefore \sum_{i=2}^{k-1} B_i > \frac{k-1}{2}$

$\vdots$ $\quad\quad \Rightarrow OPT > \frac{k-1}{2}$

$s(B_{k-1}) + s(B_k) = 1$ $\quad\quad NF = k \begin{cases} k = 2m & OPT \geq m \\ k = 2m+1 & OPT \geq m+1 \end{cases} \Rightarrow \frac{NF}{OPT} \leq 2$

$\therefore$ it has an appro ratio of (at most) 2.

Given A. if for any instance I, $\max\left\{\frac{A(I)}{OPT(I)}, \frac{OPT(I)}{A(I)}\right\} \leq \rho(|I|)$

we say A is a $\rho(n)$-approx alg.

↳ abosolute approx ratio.

2. First Fix

↓速度N=?,

```
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.

3. Best Fit

在 First fit 之基础上找 tightest bin

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

上述3种 algorithm: on-line algorithm → can't change decision.

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

off-line algorithm.

4. first (best) fit decreasing

可以看到 entire items ⟩ => 对 item 排序

trouble maker: 大 item ⟩ first (best) fit decreasing

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.

The Knapsack Problem — 0/1 version (不可拆分) NP hard.

n 个背包, m weight 限制, P_i profit, w_i weight.

=> 求取 maximum profit : approximate ratio = 2.

dynamic programming

W_{ip} = 从 $\{1, \dots, i\}$ 中取一组, 使有 minimum weight 和 total profit 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: $W_{i,p} = \infty$

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

$i = 1, \dots, n$. 但 $p = 1, \dots, nP_{max}$ => $O(n^2 P_{max})$.

=> 如果 profit 很大, 同时 ÷ 10 w (●比如) 到 small range

↓有精度损失.

$(1+\varepsilon) P_{alg} \le P$, ε : 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.

distance:

$d(x,x) = 0$ (identity)

$d(x,y) = d(y,x)$ (symmetry)

$d(x,z) \le d(x,y) + d(y,z)$ 三角形.

$dist(s_i, c)$ = distance from s_i to the closest center.

$r(C)$ = smallest covering radius.

1. greedy

先找 best possible, 然后逐渐 add, 来减少 covering radius. ✗.

→ C^*: optimal $r(C^*) \le r$ ($r(C^*)$ 已知)

```
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);
}
```

选一个点做center. 在它周边 2r 范围内 remain 的点删去 (均可覆盖)

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

但 $r(C^*)$?

设 $0 < r < r_{max}$. 猜, 2分法 $r = (0 + r_{max})/2$

$r_{max} < \infty$

=> { Yes : K centers found with $2r$, ↓
No : r 太小, ^ ↑ }

最后 $r_0 \le r \le r_1$, $r = \frac{r_0 + r_1}{2}$

Solution radius $= 2r_1$ => 2 approximation

=> be far away (smarter)

```
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;
}
```

找越远越好.

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

—— 2-approximation

unless P=NP. 不存在近似比<2 in solution.

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 .