2.1 Single machine scheduling.

n jobs need to be scheduled over time

job j {
- cannot start before time r_j $(j=1,\cdots,n)$
- has a duration of p_j $(j=1,\cdots,n)$
- has a due date d_j $(j=1,\cdots,n)$

Say that job j is **available** at time t if $r_j \le t$ and job j has not been scheduled yet.

machine is **idle** at time t if no job is processed at time t.

Earliest Due Date (EDD) Algo.

At any moment that machine is idle, start the job that has the earliest due date among the jobs that are available at the moment.

Theorem 2.1 EDD is a 2-approximation algo.

EDD is a greedy type of algo. since it always make the choice that seems best at the moment.

Let's consider a wider class of algo.

Greedy algo. At any moment that the machine is idle, start some job among the jobs that are available at that moment.

Theorem 2.2 Any Greedy algo. is a 2-approximation algo. if all due dates are negative (≤ 0).

Proof 1. Polynomial. ✓

2. Feasible. ✓ since no job start before due date and no overlap of jobs.

3. $\le$ 2-OPT. Let's denote

L^*_{max}, C^*_{max} = max. lateness / completion time of opt.

L_{max}, C_{max} = max. lateness / completion time of EDD./Greedy

Since due date d_j are negative and completion times C_j are positive, we have.

$L^*_{max} = \max_j (C^*_j - d_j) \ge \max_j C^*_j = C^*_{max}$ and

$L^*_{max} = \max_j (C^*_j - d_j) \ge \max_j (-d_j)$

Also clearly that a greedy algo. always produces a schedule of minimal length, thus

$$C_{max} \le C^*_{max}.$$

Therefore,

$$L_{max} = \max_j (C_j - d_j) \le C_{max} + \max_j (-d_j) \le C^*_{max} + \max_j (-d_j) \le 2L^*_{max}$$

2.2 The k-center problem.

Greedy Algo. Pick the first center arbitrarily. Next, choose the ~~point that~~ centers one by one until k centers are chosen, and always choose the point that is furthest away from the set of centers already chosen. Let S be the chosen centers.

$k=2$. 直线上距离相同的五个点. Greedy = {1, 5}. OPT = {2, 4}

It's easy to see that Greedy algo. is 2-approximation.

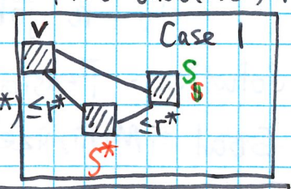
Theorem 2.3 Greedy algo. is a 2-approximation algo. for k-center.

Proof. Let S^* be an optimal set of centers, r^* be the optimal value.

Take any $v \in V$, Let $s^* \in S^*$ be its nearest center. Then $dist(v, s^*) \le r^*$

Case 1: There is an $s \in S$ with $dist(s, s^*) \le r^*$

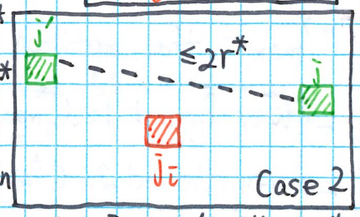
By triangle inequality, $d(v,s) \le d(v,s^*) + d(s,s^*) \le r^* + r^* \le 2r^*$.



Case 2: There is NO $s \in S$ with $dist(s, s^*) \le r^*$

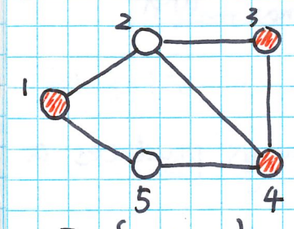
$j_i \in S^*$, $j, j' \in S$, $d(j, j') \le d(j, j_i) + d(j, j_i) \le 2r^*$

假设先选 j', 再选 j. When j' was chosen, it was the point with max. dis from the chosen center so far. Thus, dis from v to nearest center is $\le d(j, j') \le 2r^*$.



Remarkably, Greedy is the best we can do:

Theorem 2.4 There is no α-approximation algo. (for k-center) for any $\alpha<2$, unless P=NP.



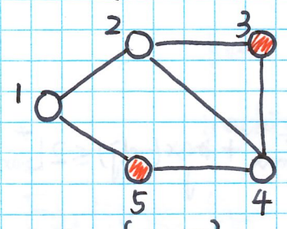
Vertex Cover

Instance: Graph $G=(V,E)$

Solution: $I \subseteq V$, s.t. every edge has an endpoint in I

Goal: minimize the number of vertices in I.

$I=\{1,3,4\}$



Dominating Set

Instance: Graph $G=(V,E)$

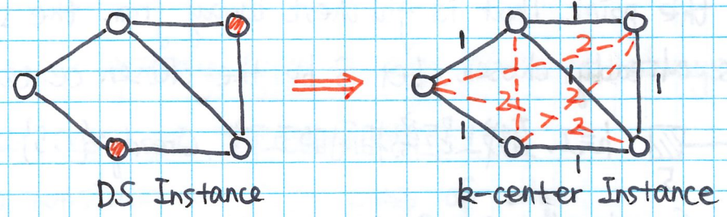
Solution: $I \subseteq V$, s.t. every vertex is in I or has a neighbour in I

Goal: minimize the number of vertices in I

$I=\{3,5\}$

Fact: Dominating Set is NP-complete

Proof Given $G=(V,E)$ and a number k. Is there a DS with at most k vertices?



DS Instance ⟹ k-center Instance

$OPT^{DS} \le k \Rightarrow OPT^{kc}=1 \Rightarrow$ Algo. $\le \alpha OPT^{kc} < 2 \Rightarrow$ Algo. $=1$

$OPT^{DS} > k \Rightarrow OPT^{kc}=2 \Rightarrow$ Algo. $=2$.

2.3 Parallel machine scheduling

Instance: m identical machines

n jobs with lengths $p_j \ge 0$ $(j=1,\cdots,n)$

Solution: Feasible schedule (assignment of jobs to machines)

Goal: Minimize the makespan C_{max} (the largest completion time)

C^*_{max} = the optimal makespan, $p_{max} = \max_j p_j$ (the largest job length)

Lower Bound 1: $C^*_{max} \ge p_{max}$: the length of any schedule $\ge$ the largest job length

Lower Bound 2: $C^*_{max} \ge (p_1+p_2+\cdots+p_n)/m$

Algo. 1 List Scheduling (Greedy Algo.)

Assign the jobs one by one (in arbitrary order) to the machines. At any step, choose the machine with the smallest load so far.

Algo. 2 Local Search

Start with any schedule

Repeat as long as:

如果将某工作移在其他机器的最后能减少其完成时间, do it.

To ensure poly. running time, we always choose the machine with smallest load at that moment.

Algo. 3 Longest Processing Time First Rule (LPT)

Order jobs by their processing time ($p_1 \ge p_2 \ge p_3 \ge \cdots \ge p_n$) and apply the list scheduling algo. with this order.

Theorem 2.5 Algo. 1 & 2 are 2-approximation algo.

1. Polynomial

Proof Clearly, algo. 1 has the poly. running time.

For algo. 2 (Local Search), Let L_k be the smallest machine load after iteration k.

Observe: By moving a job, the smallest machine load does not decrease. $\rightarrow L_0 \le L_1 \le L_2 \le \cdots$

Claim: No job moves more than once.

Assume job j moves more than once. Let L / L' be the smallest machine load before the 1st / 2nd move of job j.

$\rightarrow L > L' \rightarrow\leftarrow$ observation.

Thus, each job moves at most once. $\Rightarrow$ At most n iterations.

2. Feasible. Algo. 1 & 2. ✓.

3. Ratio: ≤ 2. Let l be the job that completes last, S_l be its start time, and p_l be its length $\Rightarrow C_{max} = S_l + p_l$. ① Algo. 1 $(p_1+\cdots+p_n)-p_l \ge mS_l$ ②

Observe: no machine has a load $< S_l$, otherwise LS could start l earlier and algo. 2 would have moved l to another machine

$$C_{max} \overset{①}{=} S_l + p_l \overset{②}{\le} \frac{1}{m}\sum_{j\neq l} p_j + p_l = \frac{1}{m}\sum_{j} p_j + (1-\frac{1}{m})p_l \overset{LB1}{\le} C^*_{max} + (1-\frac{1}{m})p_l \overset{LB2}{\le} (2-\frac{1}{m})C^*_{max}$$

$+\frac{1}{m}(p_l - p_l)$

Theorem 2.6 Algo.3 (LPT) is a $4/3$ − approximation algo.

Proof Poly. & Feas. are obvious.

Assume that it holds for any instance with at most $n' < n$ jobs.

Consider an instance I of n jobs.

Let σ be the LPT schedule and denote its length by C_{max}

Let l be the job that completes last.

1. $l = n$, Let S_n be the start time of job n. From the analysis of list scheduling, we know that

$$C_{max} = S_n + p_n \le \frac{1}{m}\left(\sum_j p_j\right) + p_n \le C^*_{max} + p_n$$

if $p_n \le C^*_{max}/3$, then done.

if $p_n > C^*_{max}/3$, then $C^*_{max} < 3p_n$. Since p_n is the smallest processing time, the optimal schedule has at most 2 jobs per machine.

Fact: If the processing time of each job is more than $1/3 \cdot C^*_{max}$, then LPT produces the optimal schedule.

i.e. $C_{max} = C^*_{max}$

2. $l < n$. Remove job n from σ and denote the remaining schedule by σ'. Since job n was added last but is not the only job that completes last, C_{max} remains.

let I' be the instance of the jobs $1, \cdots, n-1$. Note that σ' is exactly the schedule that are obtained by applying LPT to I'

By induction, C_{max} is at most $4/3$ times the optimal length of I', which is at most $4/3$ times of the optimal length of I.

$$C_{\sigma'} = C_\sigma = C_{max} \le \frac{4}{3} C^*_{I'max} \le \frac{4}{3} C^*_{Imax} = \frac{4}{3} C^*_{max}$$

2.4 The traveling salesman problem.

Instance: Complete graph with a cost c_{ij} for every pair i, j.

Solution: A cycle that goes through each point exactly once.

Cost: The length (sum of the edges costs) of the cycle

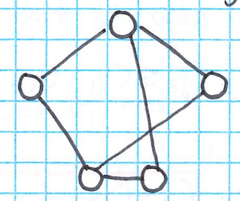
Goal: Find a solution of minimum cost.

Theorem 2.7 For TSP without the triangle inequality assumption, there does not exist an α-approximation algo for any $\alpha \ge 1$, provided $P \ne NP$.

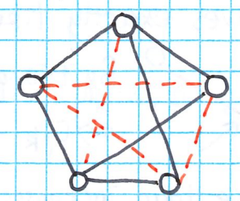
Proof Fact: The Hamiltonian Cycle problem is NP-complete.

Given an instance $G=(V,E)$ of HC, form an instance of TSP by defining $c_{ij} = 1$ $\forall (i,j) \in E$ and $c_{ij} = \alpha n$ $\forall (i,j) \notin E$.

If there is no HC in G, then any TSP should use at least one of the edges of length αn, and other edges on the tour have length at least 1. The total length of the optimal TSP tour is at least $\alpha n + (n-1) \ge \alpha n + 1$ (for $n \ge 2$). Hence,



$G=(V,E), |V|=n$



TSP − instance.

$c_{ij} = 1 \quad \forall (i,j) \in E$

$c_{ij} = \alpha n \quad \forall (i,j) \notin E$

G has a HC $\Rightarrow OPT^{TSP} = n \Rightarrow ALGO \le \alpha n$.

G has no HC $\Rightarrow OPT^{TSP} \ge \alpha n + 1 \Rightarrow ALGO \ge \alpha n + 1$.

Three algorithms for metric symmetric TSP

Algo. 1 Double Tree: 1. Find an MST
2. Double the edges
3. Find an Euler tour
4. Cut short

Algo. 2 Nearest Addition: 1. Start with a tour on 2 points
2. Keep adding the nearest point.

Algo. 3 Christofides' Algo.: 1. Find an MST T
2. Find a minimum matching M for the odd-degree vertices in T
3. Add M to T
4. Find an Euler tour
5. Cut short.

minimum cost perfect matching

an example of a maximum matching

注意已經有 1.

Theorem 2.8 Double tree is a 2-approximation algo. for TSP.

Proof: Poly. & Feas. are OK.

Lemma Let T be an MST and OPT the value of the smallest TSP tour, then cost(T) ≤ OPT

(≤ |MST| ≤ |TSP|)

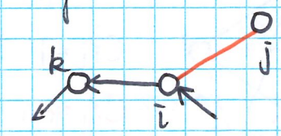
Proof Removing an arbitrary edge for the optimal TSP tour gives a spanning tree, and the cost of this tree is no more than cost(T).

Cost of the Eulertour is 2cost(T) < 2OPT.

Shortcutting does not increase the length since tricangle inequality holds.

Theorem 2.9 Nearest addition is a 2-approximation aglo. for TSP.

Proof: The algo. behaves exactly like Prim's MST algo to find an MST.



By triangle inequality: $C_{jk} \le C_{ij} + C_{ik}$

$\Rightarrow C_{jk} - C_{ik} \le C_{ij}$

Insert j in the tour increases its length by $C_{ij} + C_{jk} - C_{ik} \le 2C_{ij}$

Thus, total cost ≤ 2 cost(MST) ≤ 2opt.

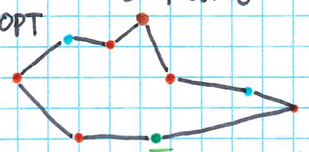
Also, Poly. & Feas. are OK.

Theorem 2.10 Christofides' algo is a 3/2 - approximation algo. for TSP.

Proof: 1. Polynomial: MST & Matching can be found in Polynomial time

2. Feasible: √.

3. Ratio ≤ 1.5. Know: cost(T) ≤ OPT.



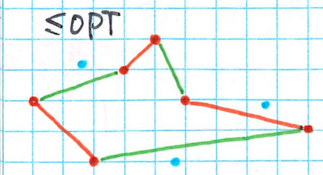
Two matchings.

Both have cost ≥ cost(M)

→ OPT ≥ 2cost(M)

claim: cost(M) ≤ OPT/2.

since the optimal TSP tour has length at least that of an optimal tour on these odd-degree vertices, which can be partitioned into two matchings by taking every other edge.



red + green ≤ OPT

red ≤ green.

Thus, cost(T) + cost(M) ≤ 3/2 · OPT

The short cutting step at the end of the algo. does not increase the length of the tour, since the triangle inequality holds.

Chapter 3. Rounding Data and Dynamic Programming.

A polynomial-time approximation scheme (PTAS) is a family of algo. A_ε, such that for any $\varepsilon > 0$:

1. A_ε is a $(1+\varepsilon)$-approximation for min / a $(1-\varepsilon)$-approximation for max.
2. The running time is polynomial, assuming ε is a constant.

Two standard ingredients of a PTAS

1. Rounding data: simplifies the instance but reduces the precision.
2. Dynamic Programming: when instance is simplified enough, then straightforward DP may be enough to solve the simplified instance in Polynomial time.

3.1 The knapsack problem.

Instance: A set of items $I = \{1, 2, \cdots, n\}$, a capacity B and for each item i a value v_i and size $s_i \le B$

Solution: $S \subseteq I$ s.t. $\sum_{i \in S} s_i \le B$

Value: $\sum_{i \in S} v_i$

Goal: Find solution of maximum value.

A pseudopolynomial time algo. by Dynamic Programming.

Define: A_j as a set of all pairs (t, w), s.t. there is a subset of items in $\{1, \cdots, j\}$ with size $t \le B$ and value w.

DP Algo.:

step1: $A_1 = \{(0,0), (s_1, v_1)\}$

step2: For $j = 2 : n$:

$A_j \leftarrow A_{j-1}$

For all $(t, w) \in A_{j-1}$:

If $t + s_j \le B$ then add $(t + s_j, w + v_j)$ to A_j

Return largest value w in A_n.

eg.

i	1	2	3
s_i	2	3	6
v_i	3	3	5

B = 7.

$A_1 = \{(0,0), (2,3)\}$

$A_2 = \{(0,0), (2,3), (3,3), (5,6)\}$

$A_3 = \{(0,0), (2,3), (3,3), (5,6), (6,5), (8,8), (9,8), (11,11)\}$

(6,5) is dominated by (5,6), because size is less, value is more

OPT = 6.

$S^* = \{1, 2\}$

Lemma. The knapsack problem can be solved (exactly) in O(nBV) time.

Proof The # of pairs (t,w) in each A_j is no more than BV where $V = \sum_{i=1}^{n} v_i$. $t \le B$, $w \le V$.