Chapter-10

Extending the limit of Tractability

10.1 Finding small Vertex Covers

The worst case instances of NP-complete problems are not solvable in polynomial time except in a few situation where it might be possible.

In the vertex cover problem there are two natural size parameters. Size of the graph and size of the vertex cover. If we have n vertices in the graph and we want to find out vertex cover of size K, there are ${}^{n}C_{K}$ subsets.

Each takes time $O(K \times n)$ to check whether it is a vertex cover and $\forall\ {}^{n}C_{K}$ subsets the total time becomes $O(K \times n^{K+1})$.

Theorem 10.1 - If $G = (V, E)$ has n nodes the maximum degree of any node is atmost 'd' and there is a vertex cover of size atmost 'K' then G has atmost 'K x d' edges.

Proof - Let S be a vertex cover in G of size $K' \leq K$ every edge in G has atleast 1 end in S but each node in S can cover atmost 'd' edges. Thus there can be atmost $K'd \leq Kd$ edges.

To search for a K node vertex in G

1) If G contains no edges then empty set is a vertex cover.
2) If G contains $> K|V|$ edges then it has no K-node vertex cover else.

3) for every edge $e = (u, v)$ recursively check if either of $G - \{u\}$ or $G - \{v\}$ has a vertex cover of size $K-1$.

Chapter-11 Approximation Algorithm

→ Approximation algorithms are most suitable when polynomial time complexity is an unattainable goal.

→ Define solution guaranteed to be close to optimal. Thus making the execution feasible in polynomial time.

11.1 Load Balancing Problem

→ When multiple servers need to process a set of jobs or requests, the load balancing problem arises.

→ We are given a set of m machines $M_1, M_2 \ldots M_m$ and a set of n jobs, where each job j has a processing time t_j.

→ We seek to assign each job to one of the machines so that the loads placed on all machines are as balanced as possible.

→ Mathematically in any assignment if $A(j)$ is the set of jobs assigned to machine M_i, the total time that machine M_i needs to work is $T_i = \sum_{j \in A(i)} t_j$ called load on machine M_i.

→ The maximum load on any machine is called makespan given by $T = \max_i T_i$

→ The goal of load balancing problem is to minimize the makespan.

→ Its an NP-hard problem.

Greedy Balance

→ Start with no jobs assigned.

→ Set $T_i = 0$ and $A(i) = \phi$ for all M_i

→ for $j = 1$ to n let M_i be a machine that achieves the minimum $\min_k T_k$

→ Assign job j to machine M_i

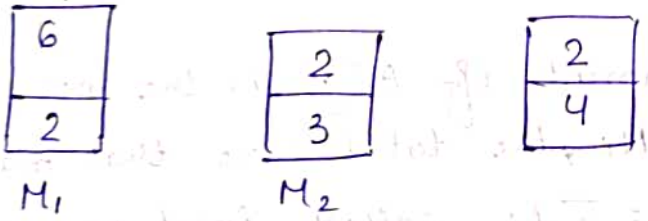
$A(i) \leftarrow A(i) \cup \{j\}$

$\leftarrow T_i \leftarrow T_i + T_j$

end for

* For a sequence of 6 jobs which sizes 2, 3, 4, 6, 2, 2 and 3 machines. Calculate make-span using above algorithm.

Makespan = $\max_i T_i$ $\quad T_i = \sum_{j \in A(i)} t_j$



Max load on any machine ← makespan

So here M_1 has max load i.e. 8

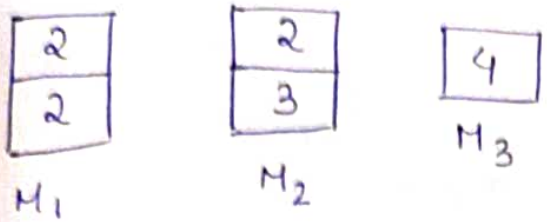
Dry run

$j = 1$ $\quad A(i) = \{2\}$ $\quad T_i = 2$

$j = 2$ $\quad T_2 = 3$

$j = 3$ $\quad T_3 = 4$

→ For 2, 3, 4, 2, 2, 6



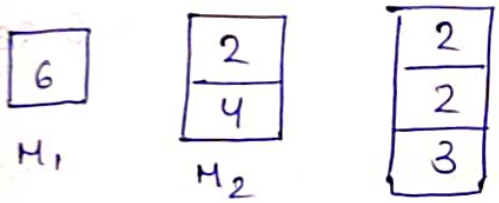
$T_1 = \phi$ $T_i = \phi$ $T_i = 0$

$2+2$ $3+2$

$= 4$ $= 5$

While placing 6 we have M1 & M_3 with same load so we can place it any where makespan will be 10.

→ For 6, 4, 3, 2, 2, 2



$T_i = \phi$ $T_i = \phi$ $T_i = \phi$

6 4 3

6 $+2$ $+2$

6 7

Everytime we have to choose machine with minimum load.

Theorem 11.1 The optimal makespan is atleast

$$T^* \geq \frac{1}{m} \sum_j t_j$$

For above prob → $T^* \geq \frac{1}{3} \sum 6+4+3+2+2+2 \geq 6.3$

Theorem 11.2 The optimal makespan is atleast $T^* \geq \max_j t_j$

→ Algorithm greedy balance produces an assignment of jobs to machines with makespan $T \leq 2T^*$

Q 2, 3, 4, 4, 3, 2, 7 No. of machines = 4

m_1	m_2	m_3	m_4
3 / 2	2 / 3	7 / 4	4
$T_i = \phi$	$T_i = \phi$	$T_i = \phi$	$T_i = \phi$
2 + 3 = 5	3 + 2 = 5	4 + 7 = 11	4

But according to, $T^* \geq \frac{1}{m} \sum_j t_j$

$$\frac{1}{4}(2+3+4+4+3+2+7) = \frac{25}{4}$$

$$= 6.25$$

Sorted Balance

1. Start with no jobs assigned
2. Set $T_i = 0$ & $A(i) = \phi \; \forall M_i$
3. Sort jobs in decreasing order of processing times t_j
4. Assume that $t_1 \geq t_2 \geq \dots \geq t_n$
5. For $j = 1 \dots n$
6. Let M_i be the machine that achieves the minimum $\min_k T_k$
7. Assign job j to machine M_i
8. Set $A(i) \leftarrow A(i) \cup \{j\}$
9. Set $T_i \leftarrow T_i + t_j$
10. End for.

Theorem 11.5

Sorted balance algorithm produces an assignment T^* with makespan

$$T \leq \frac{3}{2} T^*$$

Q Under which condition related to no. of jobs will the greedy balance always be optimal.

Soln If no. of jobs equal to no. of machines or less than no. of machines.

Theorem 11.4

If there are more than m jobs then $T^* \geq 2t_{m+1}$

Proof - Consider only the first m+1 jobs in the sorted order. They each take atleast t_{m+1} time. There are m+1 jobs and only m machines. So there must be a machine that gets assigned two of these jobs. This machine will have processing time atleast $2t_{m+1}$

11.3 Set Cover

Greedy Set Cover

We maintain the set R of remaining elements with R = U and no sets selected while $R \neq \emptyset$

Select set S_i that minimises $w_i / |S_i \cap R|$.

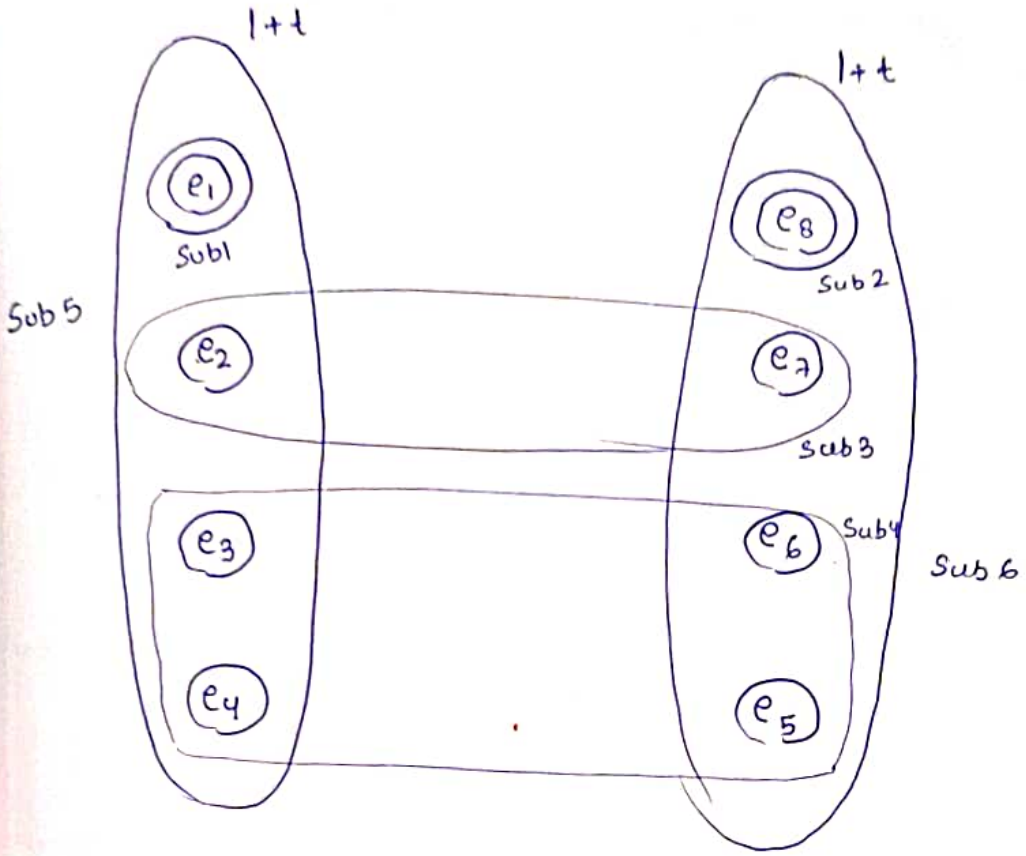
Delete set S_i from R

End while

Return selected sets.

Cost per element covered -

When the greedy algorithm builds the solution one set at a time to indicate progress it uses the criteria of choosing sets with small weight and maximum no. of elements. The measure $\frac{w_i}{|S_i|}$ gives the cost of selecting S_i i.e. we cover $|S_i|$ elements at a cost of w_i since that would be of overlapping elements between sets, we compute the cost by only considering elements common to the set S_i and set R of uncovered elements.



	Sub 1	Sub 2	Sub 3	Sub 4	Sub 5	Sub 6
wi	1	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1+t}{4}$	$\frac{1+t}{4}$
Si∩R	= 1	= 1	= 0.5	= 0.25	= > 0.25	= > 0.25
Itn 2	1	1	0.5	x	$\frac{1+\epsilon/2}{}$ > 0.5	$\frac{1+\epsilon/2}{}$ > 0.5
Itn 3	1	1	x	x	> 1	> 1

Optimal → 4

According to greedy → $2+2\epsilon = 2+2\times 0.5$ [let $\epsilon = 0.5$]

$= 3$

Theorem 11.9

If C is the set cover obtained by greedy set cover then $\sum_{S_i \in C} w_i = \sum_{\phi \in U} c_\phi$

Define $c_\phi = \frac{w_i}{|S_i \cap R|}$ for all $\phi \in S_i \cap R$

c_ϕ is computed for every element.

Theorem 11.10 For every set S_k the sum $\sum_{\phi \in S_k} c_\phi$ is atmost $H(|S_k|) \cdot w_k$ where H is a harmonic function bounded as $H(n) = \Theta(\ln n)$

Theorem 11.11 The set cover C selected by greedy set cover has weight atmost $H(d^*)$ times optimal weight w^*.

Let C^* denote the optimum set cover, so that $w^* = \sum_{S_i \in C^*} w_i$

For each of the sets in C^* using 11.10 we can write. $w_i \geq \frac{1}{H(d^*)} \sum_{\phi \in S_i} c_\phi$

Because these sets form a set cover, we have

$$\sum_{S_i \in C^*} \sum_{\phi \in S_i} c_\phi \geq \sum_{\phi \in U} c_\phi$$

Combining this with 11.9 we get,

$$w^* = \sum_{S_i \in C^*} w_i \geq \sum_{S_i \in C^*} \frac{1}{H(d^*)} \sum_{s \in S_i} c_s \geq \frac{1}{H(d^*)} \sum_{s \in U} c_s =$$

$$\frac{1}{H(d^*)} \sum_{S_i \in C} w_i$$

11.4 Pricing Method

Theorem 11.12 The set cover approximation algorithm can be used to give an $H(d)$ approximation algorithm for the weighted vertex cover Problem, where d is the maximum degree of the graph.

Problem statement of weighted vertex cover

Each vertex $i \in V$ has a weight $w_i \geq 0$ and the weight ~~of a~~ s of a is given by

$$w(S) = \sum_{i \in S} w_i$$

The goal is to find a vertex cover S for which $w(S)$ is minimum.

Using pricing method or primal dual method we consider each edge e as a separate agent who is willing to pay something to the node that covers it. The edge will determine the prices.

$\boxed{p_e \geq 0}$ $\forall$ edge $e \in E$, is that if each edge $e \in E$ pays the price p_e. The cost of S will be covered approximately.

Theorem 11.13 For any vertex cover S^* and any non-negative and fair prices p_e $\sum_{e \in E} p_e \leq w(S^*)$.

Pricing-Method algorithm for finding vertex cover:

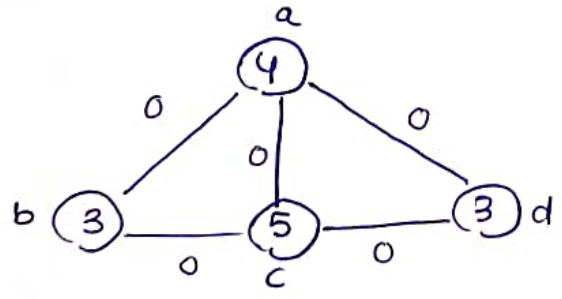
The algorithm greedily sets the prices and uses them to drive the way it selects nodes for the vertex cover.

We say a node i is paid for $\sum_{e=(i,j)} p_e = w_i$

Vertex-Cover-Approx (G, w)

1. Set $p_e = 0$ $\forall$ $e \in E$
2. while there is an edge $e = (i,j)$ s.t neither i nor j is ~~weight for~~ paid for
3. Select edge e
4. Increase p_e
5. end while
6. Let S be the set of all weight for nodes
7. return S.

Consider a graph G

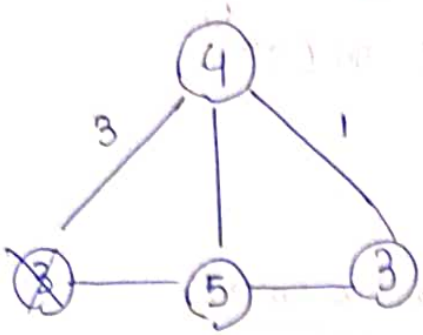


$e=(a,b)$

$p_e = \min(w_a, w_b)$

$= 3$

The prices p_e are said to be fair if for each vertex i the edges adjacent to i do not have to pay more than the cost of the vertex.



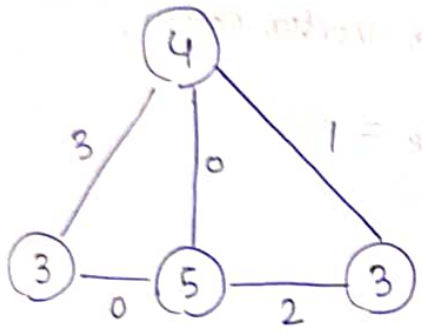
e = (a, d)

$P_e = \min(w_a, w_d)$

$= 1$

$S = \{a, b, d\}$

$w(S) = 4 + 3 + 3 = 10$



e = (c, d)

$P_e = \min(w_c, w_d)$

$= 2$

$S^* = \{a, c\}$

$w(S^*) = 4 + 5 = 9$

11.14 (Theorem) - The set S and prices p returned by the algorithm satisfiable inequality

$$w(S) \leq 2 \sum_{e \in E} P_e$$

Theorem 11.15- The set S returned by the algorithm is the vertex cover and it's cost is atmost twice the minimum cost of any vertex cover.

11.2 The Center Selection Problem

→ Given a set S of n size (n places or cities) we have to select K centers for building large shopping malls.

→ People in each of the n cities should shop at one of the malls so all the K malls should be central.

→ Dist. between any two sides is given by the standard eucledian distance formula.

→ Mathematically if C is the set of centers, we say C forms an r cover if each side is within distance atmost r from one of the centers i.e. if dist(s, c) ≤ r ∀ sides s ∈ S

→ dist(s, C) can be defined as the dist of a side s to all centers C = dist(s, c) = mid dist(s, c) ∀ c ∈ C

→ The minimum r for which C is an r cover is called covering radius of C and is denoted by r(c).

→ The goal of the center selection problem is to find out a set of K center C for which r(c) is as small as possible.

Q For the given set of sides a to i where each side is a point in the cartesian coordinate plane. Calculate C for each of the three cases.

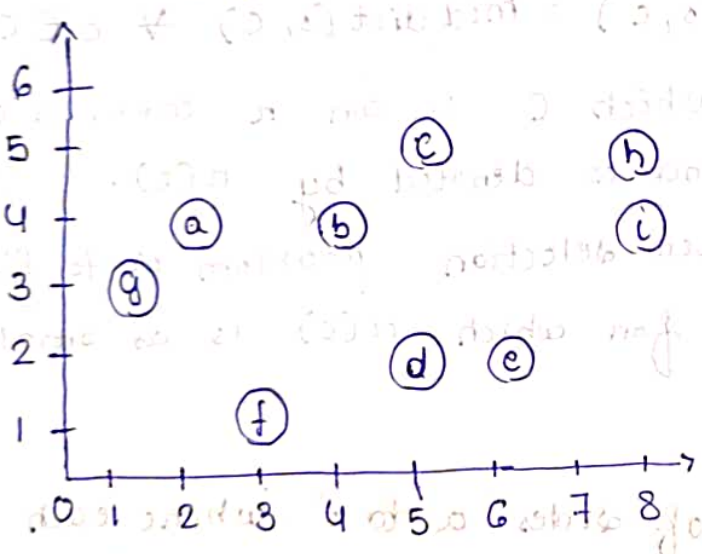
Case a) r=1, K=3 b) r=2, K=3 c) r=2, K=2

a: (2,4) c: (5,5) e: (6,2) g: (1,3) i: (8,4)

b: (4,4) d: (5,2) f: (3,1) h: (8,5)

1st Greedy Algorithm based on optimal radius

→ Let S' represent all sites that still need to be covered
→ Initialise S' = S
→ Let C = ϕ
→ while S' ≠ ϕ
→ Select any site s ∈ S' and add s to C
→ Delete all sites from S' that are at a dist atmost 2r from s.
→ Endwhile
→ If |C| ≤ K
→ return C as set of selected sites
→ else answer doesn't exist (there is no set of K centers with covering radius atmost (r).



S' = S = {a, b, c, d, e, f, g, h}

C = ϕ, while S' ≠ ϕ

a) r = 2, K = 3 s = {d}, C = {d}

2r = 2 × 1 = 2

dist (d, a) = $\sqrt{13}$ dist (d, c) = 3 dist (d, f) = $\sqrt{5}$

dist (d, b) = $\sqrt{5}$ dist (d, e) = 1 dist (d, g) = $\sqrt{17}$

dist (d, h) = $\sqrt{18}$

dist (d, i) = $\sqrt{13}$

$\pi = 2, \quad K = 3$

$\phi = d$

Delete till 2π from d

$2\pi = 2 \times 1 = 2$

$S' = \{a, b, c, d, e, f, g, h, i\}$

$S' = \{a, b, c, f, g, h, i\}$

$\phi = h$

dist (h, c) = 3

dist (h, i) = 1

dist (h, f) = $\sqrt{41}$

$S' = \{c, f\}$

$K = 3 \quad \{d, a, b\}$

b) $\pi = 2, \quad K = 3$

$S' = \{a, b, c, d, e, f, g, h, i\}$

$\phi = d \qquad 2\pi = 2 \times 2 = 4$

$S' = \{a, g, h, i\}$

$\phi = a$

$S' = \{h, i\}$

$\phi = h$

$\phi = a$

$2\pi = a$

dist (a, b) = 2

dist (a, c) = $\sqrt{10}$

dist (a, f) = $\sqrt{10}$

dist (a, g) = $\sqrt{2}$

dist (a, h) = $\sqrt{37}$

dist (a, i) = 6

$S' = \{c, f, h, i\}$

c) $\pi = 2, \quad K = 2$

$S' = \{a, b, c, d, e, f, g, h, i\}$

$\phi = a$

dist (a, b) = 2

dist (a, g) = $\sqrt{2}$

dist (a, c) = $\sqrt{10}$

dist (a, f) = $\sqrt{10}$

dist (a, d) = $\sqrt{13}$

dist (a, e) = $\sqrt{20}$

$S' = \{e, h, i\}$

$\phi = h$

dist (h, e) = $\sqrt{13}$

dist (h, i) = 1

$C = \{a, h\}$

Q a) $\pi = 1$, $K = 3$

b) $\pi = 2$, $K = 3$

c) $\pi = 2$, $K = 2$

a) $\pi = 1$, $K = 3$

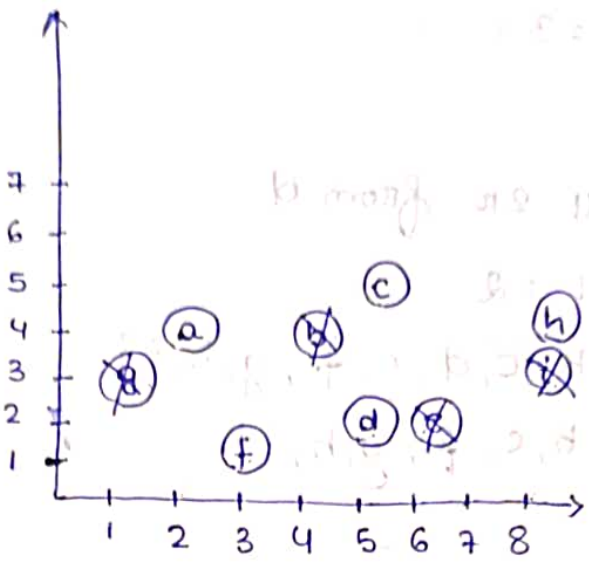
a(2,4) e(6,2)

b(4,4) f(3,1)

c(5,5) g(1,3)

d(5,2) h(8,5)

i(8,4)



dist(d,a) = $\sqrt{13}$

dist(d,b) = $\sqrt{5}$

dist(d,c) = 3

dist(d,e) = 1 ✓

dist(d,f) = $\sqrt{5}$

dist(d,g) = $\sqrt{17}$

dist(d,h) = $\sqrt{18}$

dist(d,i) = $\sqrt{13}$

$\phi = d$

Delete fill 2π from d

$2\pi = 2 \times 1 = 2$

$S' = \{a, b, c, d, e, f, g, h, i\}$

$S' = \{a, b, c, f, g, h, i\}$

$\phi = a$

$2\pi = 2$

dist(a,b) = 2 ✓

dist(a,c) = $\sqrt{10}$

dist(a,f) = $\sqrt{10}$

dist(a,g) = $\sqrt{2}$ ✓

dist(a,h) = $\sqrt{37}$

dist(a,i) = 6

$S' = \{c, f, h, i\}$

$\phi = h$

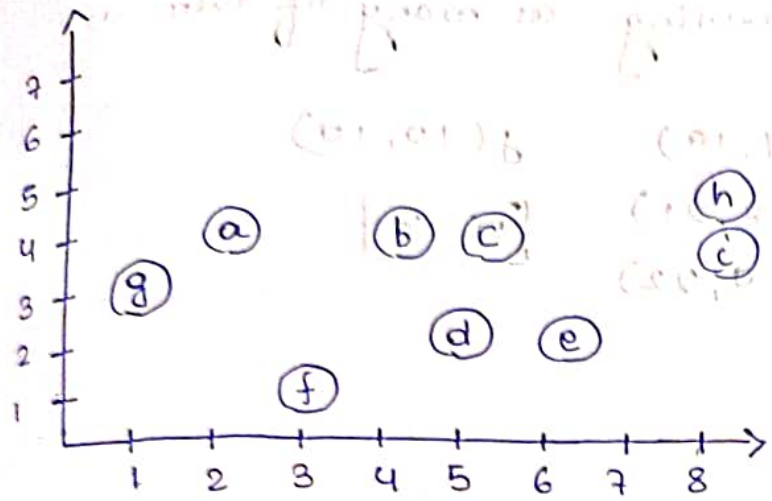
dist(h,c) = 3

dist(h,i) = 1 ✓

dist(h,f) = $\sqrt{41}$

$S' = \{c, f\}$

b) $\pi = 2$, $K = 3$



$S' = \{a, b, c, d, e, f, g, h, i\}$

$\phi = d$

$2\pi = 2 \times 2 = 4$

$S' = \{a, g, h, i\}$

$\phi = a$

$S' = \{h, i\}$

$\phi = h$

c) $\pi = 2$, $K = 2$

$S' = \{a, b, c, d, e, f, g, h, i\}$

$\phi = a$

dist (a, b) = 2
dist (a, g) = $\sqrt{2}$
dist (a, c) = $\sqrt{10}$ = 3
dist (a, f) = $\sqrt{10}$ = 3
dist (a, d) = $\sqrt{13}$ = 3
dist (a, e) = $\sqrt{20}$ = 4

$S' = \{e, h, i\}$

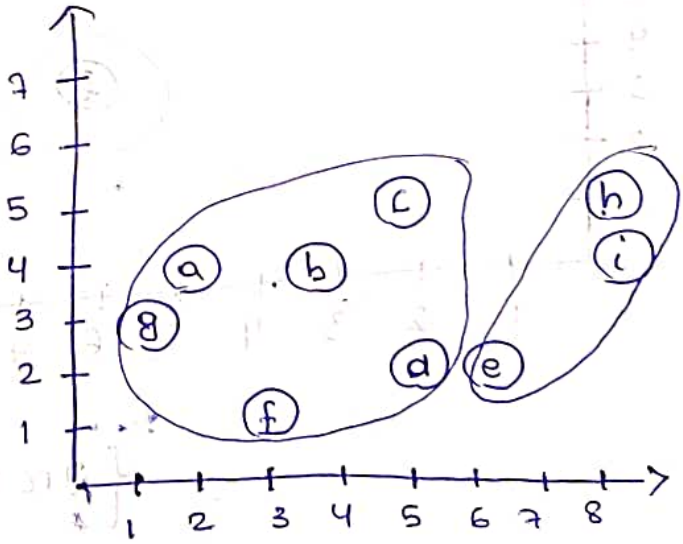
$\phi = h$

dist (h, e) = $\sqrt{13}$ = 3
dist (h, i) = 1

$C = \{a, h\}$

$K = 2$

$\pi = 2$

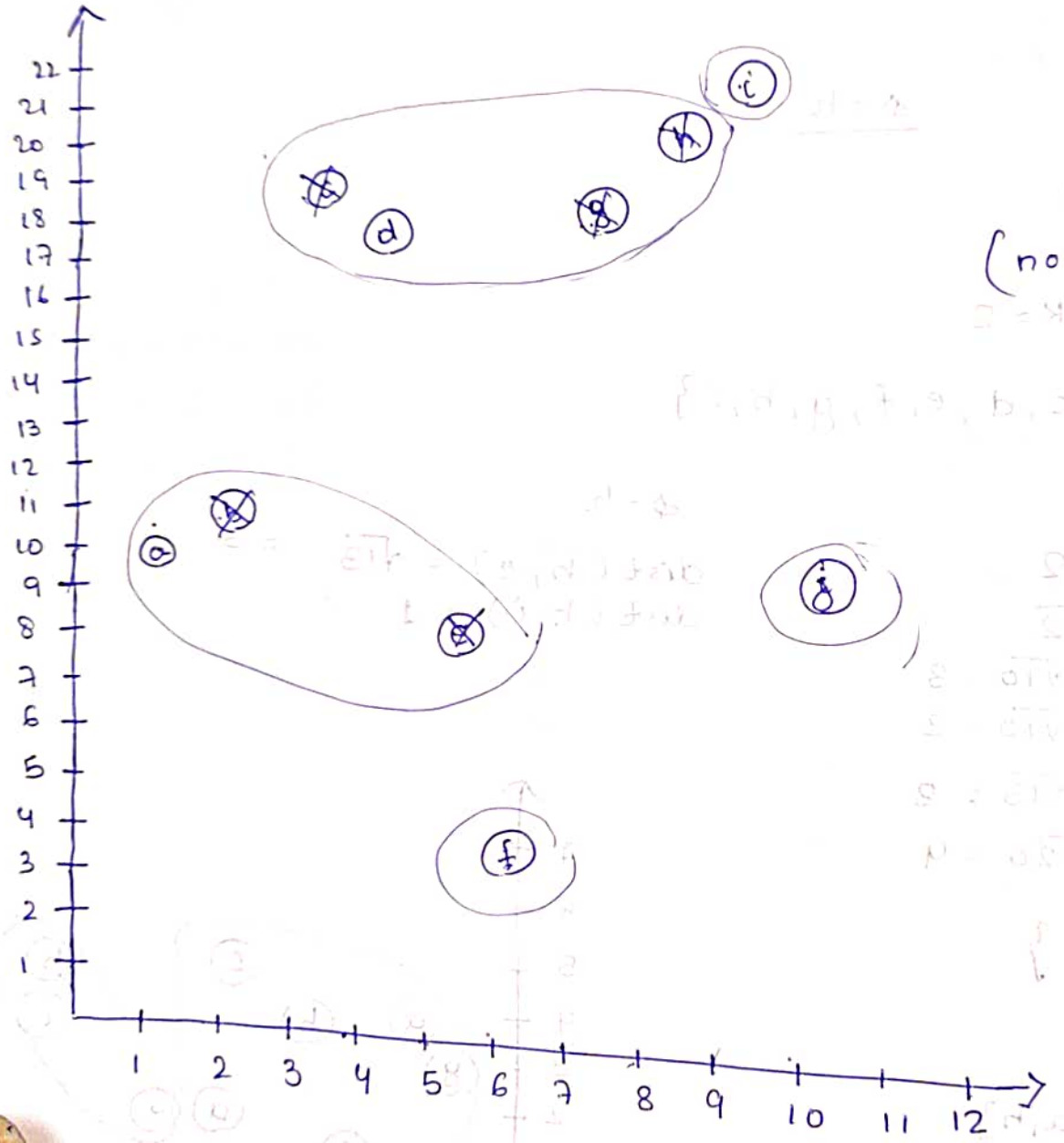


Q For the given set of points find the optimal no. of centers required for covering as many of them as possible

a (1, 10)
b (2, 11)
c (3, 9)
d (4, 18)
e (5, 8)
f (6, 4)
g (7, 19)
h (8, 21)
i (9, 22)
j (10, 10)

r = 3

Find K and C.



(not optimal)

$\therefore \{2r = 2 \times 3 = 6\}$

$S' = \{a, b, c, d, e, f, g, h, i, j\}$

$\underline{d = d}$

$d(d, a) = \sqrt{73}$

$d(d, b) = \sqrt{53}$

$d(d, c) = \sqrt{2}$ ✓

$d(d, e) = \sqrt{101}$

$d(d, f) = \sqrt{148}$

$d(d, g) = \sqrt{10} = 3$ ✓

$d(d, h) = \sqrt{25} = 5$ ✓

$d(d, i) = \sqrt{41} = 6.4$

$d(d, j) = \sqrt{100}$

$S' = \{a, b, e, f, i, j\}$

$\underline{S = a}$

$d(a, b) = \sqrt{2}$ ✓

$d(a, e) = \sqrt{20} = 4$ ✓

$d(a, f) = \sqrt{61} =$

$d(a, i) = \sqrt{208}$

$d(a, j) = \sqrt{81}$

$S' = \{f, i, j\}$

$\underline{S = i}$

$d(i, f) = \sqrt{333}$

$d(i, j) = \sqrt{145}$

$S' = \{f, j\}$

$S = f$ $\quad\quad$ $S = j$

$d(f, j) = \sqrt{52}$

2nd greedy algorithm

1. Assume $K <= |S|$
2. Select any site ϕ and include in C
3. while $|c| < K$
4. Select a site $\phi \in S$ that maximises distance ϕC.
5. Add site ϕ to C.
6. end while
7. return C as the selected set of sites.

$\phi = \{a\}$

$C = \{a\}$

$\phi = i$

$C = \{a, i\}$ > optimal ?

dist (a, j) = 9

dist (a, f) = $\sqrt{61}$ = 7.—

dist (i, c) = $\sqrt{45}$ = 6.—

dist (i, d) = $\sqrt{41}$ = 6.—

$C = \{g, f\}$ → optimal ✓

{Q.3 exercise (Pg. 658) bit a, b both}

Q.3 Suppose you are given a set of +ve integers $A = \{a_1, a_2, \ldots, a_n\}$ and a +ve integer B. A subset $S \subseteq A$ is called feasible if the sum of the nos. in S does not exceed B: $\sum_{a_i \in S} a_i \leq B$

The sum of the nos. in S will be called the total sum of S.

You would like to select a feasible subset S of A whose total sum is as large as possible.

Example. If $A = \{8, 2, 4\}$ and $B = 11$, then the optimal solution is the subset $S = \{8, 2\}$.

a) Here is an algo

Initially $S = \phi$

Define $T = 0$

For $i = 1, 2, \ldots, n$

If $T + a_i \leq B$ then

$S \leftarrow S \cup \{a_i\}$

$T \leftarrow T + a_i$

End if

End for

Give an instance in which the total sum of the set S returned by this algo is less than half the total sum of some other feasible subset of A.

11.8 The Knapsack Problem

Arbitrarily good approximation

→ Given n items where each item $i = 1, 2, \ldots, n$ has 2 integer parameters a weight w_i and a value v_i

→ Given knapsack capacity W the goal of the knapsack problem is to find a subset S of items of maximum value subject to the restriction that the total weight of the set should not exceed W.

→ Mathematically, we aim to maximize $\sum_{i \in S} v_i$ subject to the condition $\sum_{i \in S} w_i \leq W$

→ The Dynamic Programming algorithm for this problem will run in polynomial time for most of the cases with the condition $\sum_{i \in S} v_i$ is atmost $(1+\epsilon)$ factor below the optimal answer and ϵ is the fixed value > 0 and ϵ does not depend on n.

→ Such an algorithm is called polynomial time approximation scheme.

When values are very small

→ Let $V^* = \max_i v_i$

If the values are all small integers then V^* is small and the problem can be solved in polynomial time. However if values are large we use a rounding parameter B and consider values rounded to an integer multiple value of B

→ For each item i calculate its rounded value as

$$\tilde{v}_i = \lceil v_i / b \rceil b$$

11.34 For each item i, $v_i \le \tilde{v}_i \le v_i + B$

Knapsack-Approx (ϵ)

Set $b = (\epsilon/(2n)) \max_i v_i$

Solve the knapsack problem with $\hat{v}_i$ or $\tilde{v}_i$

Return S

Caret $\hat{v}_i = \tilde{v}_i / b = \lceil v_i / b \rceil$ → Scaled values for Knapsack problem

$\tilde{v}_i = \lceil v_i / b \rceil \cdot b$

Current DP soln

if $n \notin O$, $OPT(n, \omega) = OPT(n-1, \omega)$

if $n \in O$, $OPT(n, \omega) = v_n + OPT(n-1, \omega - \omega_n)$

⇒ If $\omega < \omega_i$, $OPT(i, \omega) = OPT(i-1, \omega)$

else $OPT(i, \omega) = \max(OPT(i-1, \omega), v_i + OPT(i-1, \omega - \omega_i))$

11.35 The Knapsack problem with the value $\tilde{v}_i$ and the scaled problem with value $\hat{v}_i$ have the same set of optimal solutions, The optimal values differ exactly by a factor of b and the scaled values are integral.

The new dynamic programming algorithm for the Knapsack Problem

→ The subproblem is now defined in terms of value instead of weight $\overline{OPT}(i, V)$ is the smallest knapsack capacity w so that we can obtain a solution using a subset of items $\{1 \dots i\}$ with value atleast V.

→ For all values $i = \{0 \dots n\}$ and values $V = 0, \dots, \sum_{j=1}^{i} v_j$ $\forall$ values we have sub problem each if v^* denotes $\max_i v_i$ the largest value V is $\sum_{j=1}^{n} v_j \leq n v^*$.

→ If $n \notin O$ then $\overline{OPT}(n, V) = \overline{OPT}(n-1, V)$

If $n \in O$ then n is the only item in O, $\overline{OPT}(n, V) = w_n$

If $n \in O$ is not the only item in O, $\overline{OPT}(n, V) = w_n + \overline{OPT}(n-1, V - v_n)$

⇒ If $n \in \Theta$, $\overline{OPT}(n, V) = w_n + \overline{OPT}(n-1, \max(0 \dots V - v_n))$

Q.1 $W = 6$, $n = 3$

$w_1 = w_2 = 2$, $w_3 = 3$, $v_1 = 2$, $v_2 = 4$, $v_3 = 3$

$OPT(3, 6) = 5$

$\overline{OPT}(i, V)$ smallest w

	0	v_1	$v_1 + v_2$	$v_1 + v_2 + v_3$
0	0	0	0	0
1	0	2	2	2
2	0	2	4	4
3	0	2	4	

i	1	2	3
w	2	2	3
V	2	4	3

6

$\min(4, 3 + M[2, \ldots 4])$

Knapsack (n)

Array $M[0 \ldots n, 0 \ldots V]$

for $i = 0$ to n

$M[i, 0] = 0$

end for

for $i = 1, 2, \ldots n$

for $v = 1, \ldots, \sum_{j=1}^{i} v_j$

If $v > \sum_{j=1}^{i-1} v_j$ then $M[i, v] = w_i + M[i-1, v]$

else $M[i, v] = \min(M[i-1, v] w_i + M[i-1, \max(0, v - v_i)]$

end if

end for

end for

return the max value v s.t. $M[n, v] \leq W$

Q

i	1	2	3
w	2	2	3
V	2	4	3

$\sum_{j=1}^{i} v_j$

$i = 1$ | 2

$i = 2$ | 6

$i = 3$ | 9

	0	1	2	3	4	5	6	7	8	9
0	0									
1	0	2	2							
2	0	2	2	2	2	2	2			
3	0	2	2	2	2	2	2	3	3	3

$M[i, v] = w_i + M[i-1, v]$

$M[i, v] = \min(M[i-1, v], w_i + M[i-1, \max(0, v-v_i)])$

$M[1, 2] = 2$

$M[2, 1]$, $i = 2$, $v = 1$ $\quad 2 + 2 = 4$

$\sum_{j=1}^{2-1} v_j = \sum_{j=1}^{1} v_j = 2$

for $i = 3$

$v = 1$ to $\sum_{j=1}^{i} v_j = 1$ to $\sum_{j=1}^{3} v_j = 1$ to $(v_1 + v_2 + v_3) = 1$ to 9

$v = 1 \quad v > \sum_{j=1}^{i-1} v_j \Rightarrow v > v_1 + v_2 \Rightarrow v > 6 \rightarrow F$

$M[3,1] = \min(M[2,1], w_3 + M[2, \max(0, 1-3)])$

$v = 1 \quad v > \sum_{j=1}^{i-1} v_j \rightarrow T$

$M[3,7] = w_3 + M[2,7] = 3 + 0 = 3$

$v = 8$ T, $M[3,8] = w_3 + M[2,8] = 3$

Q

○	60	5
□	50	3
△	70	4
◇	30	2

W = 5

	0	10	20	30	40	50	60	70	80
ϕ	0	0	0	0	0	0	0	0	0
○	0	5	5	5	5	5	5	0	0
□	0	3	3	3	3	3	5	3	3
△	0								
◇	0								

i = 1 to 4

i = 1 :- V = 10 to $\sum_{j=1}^{i} v_j$ = 1 to 6

$V > \sum_{j=1}^{i=1} v_j$ ⇒ $V > \sum_{j=1}^{0} v_j$ → T

M[1,10] = w_1 + M[0,10]

= 5

V = 2 M[1,2] = w_1 + M[0,2] = 5

$i=2$ - $v = 1$ to $\sum_{j=1}^{i} v_j$ = 1 to 11

$v = 1$ to 6 - $M[2, v] = \min(M[1, v],\ w_2 + M[1, \max(0, v-5)])$
$= 3$

$v = 7$ to 11 :- $M[2, v] = w_2 + M[1, v]$

Chapter - 13

13.5 Randomized Divide and Conquer

Median Finding:

$S = \{a_1, a_2, \ldots, a_n\}$

$K = n/2$ if n is even

$(n+1)/2$ if n is odd

Splitters :

Select (S, K)

1. Choose a splitter $a_i \in S$
2. For each element a_j of S
3. Put a_j in S^- if $a_j < a_i$
4. Put a_j in S^+ if $a_j > a_i$
5. End for
6. If $|S^-| = K-1$ then a_i is the desired answer
7. Else if $|S^-| > K$ then
8. The K^{th} largest element lies in S
9. Else suppose $|S^-| = \ell < K-1$

[The K^{th} largest element lies in S^+

Recursively call select $(S, K-1-\ell)$

10. End if

Q $S = \{-100, 207, 18, 7, -31\}$

$n = 5, \quad K = (5+1)/2 = 3$

qSelect (5, 3)

Let $a_i = 18$

$S^- = \{-100, 7, -31\}$

$S^+ = \{207\}$

$|S^-| = 3 = K - 1$ (false)

$\Rightarrow |S^-| \geq 3$ (True)

qSelect (5, 3)

$S = \{100, 7, -31\}$

Let $a_i = -31$

$S = \{-100\}$

$S^+ = \{T\}$

$|S^-| = 1 = K - 1$ (false)

$|S^-| \geq 3$ (false)

$\Rightarrow |S^+| = 1 - l < K - 1$ (True)

$\Rightarrow$ qSelect $(S^+, 3)$

$S^+ = \{T\}$

↑

Single element

$\Rightarrow$ ret 7.

RANDOMISED QUICK SORT

```
QuickSort (A, p, r)
    if p < r
        q = PARTITION (A, p, r)
        Quicksort (A, p, q-1)
        QuickSort (A, q+1, p);

PARTITION (A, p, r)
    x = A[r]
    i = p-1
    for j = p to r-1
        if A[j] ≤ x
            i = i+1
            exchange A[i] with A[j]
    exchange A[i+1] with A[r]
    ret i+1.

RANDOMIZED-PARTITION (A, p, r)
    i = RANDOM (p, r)
    exchange A[r] with A[i]
    return PARTITION (A, p, r)

RANDOMIZED-QUICKSORT (A, p, r)
    if p<r
        q = RANDOMIZED-PARTITION (A, p, r)
        RANDOMISED-QUICKSORT (A, p, q-1)
        RANDOMIZED-QUICKSORT (A, q+1, r)
```

Expected running time → $O(n \log n)$

Q

	1	2	3	4	5	6	7
A =	10	8	-8	7	-70	81	99

q = R_P (A, 1, 7)

i = R (1, 7)

Let i = 4, A[i] = 7

10	8	-8	99	-70	81	7

→ pivot

1st → | 10 | 8 | -8 | 99 | -70 | 81 | 7 |

i = 1, j = 1, pivot p = 7

p < A[i] | i = 2

i = 2 → | 10 | 8 | -8 | 99 | -70 | 81 | 7 |

j i

p < A[i], i = 3

i = 3 → | 10 | 8 | -8 | 99 | -70 | 81 | 7 |

j i

p > A[i],

swap (A[i], A[j])

i = 4, j = 2

i = 4, | -8 | 8 | 10 | 99 | -70 | 81 | 7 |

p < A[i], i = 5

i=5,

-8	8	10	99	-70	81	7

j i

p > A[i]

Swap(A[i], A[j]), i=6, j=3

i=6,

-8	-70	10	99	8	81	7

j i

i=7 Swap(A[i], p)

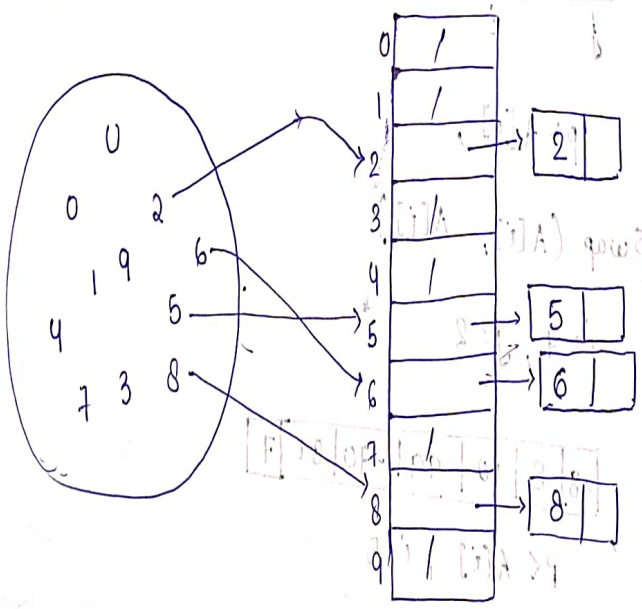
8	-70	(7)	99	8	81	10

13.6 Hashing

→ Dictionary Data Structures

Hash Tables are used for implementing dictionary data structures for storing key value pairs with the objective of performing dictionary operations very efficiently.

Direct Address Tables



→ U is the universe of key values to be stored. It consists of values from 0, 1, ..., m-1 where m is not a very large number.

→ Direct address table is an array of T(0, ... m-1) in which each position or slot corresponds to a key in universe U.

→ At any given point of time, the actual keys stored in the direct address Table are given by a set K.

DIRECT ADDRESS SEARCH (T, K)

return T[K] → O(1)

DIRECT ADDRESS INSERT (T, x)

T[x.key] = x → O(1)

DIRECT ADDRESS DELETE (T, x)

T[x.key] = NIL → O(1)

Disadvantage

→ Only suitable for small values of m

→ Wastage of space due to unallocated free slot across the table that can't be otherwise accessed.

Hash Functions $h : U \to \{0, 1, \ldots, m-1\}$

→ If a size of U is large, storing a table t of |u| may be impractical

→ The set K of keys actually stored might be very small relative to |u| leading to a lot of wastage of space.

To reduce the storage requirement to $\Theta(|K|)$ from $\Theta(|U|)$ we can use a hash function h to compute the slot h(k) which is a no. b/w 0 to m-1.

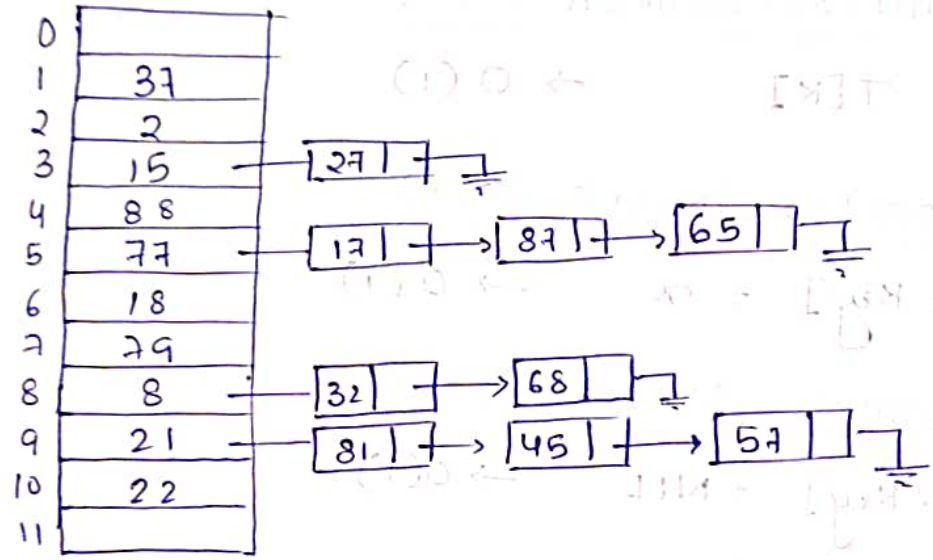
We say that if element with key k hashes to slot h(k) or h(k) is the hash value of key k.

The division method

$h(k) = k \bmod m$ $\quad K = \{21, 18, 8, 2, 22, 32, 81, 77, 45, 15, 17, 27, 37, 57, 88, 89, 79, 68, 65\}$

$m = 12$

$h(k) = 21 \bmod 12 = 9$ and so on...



Collision resolution by chaining

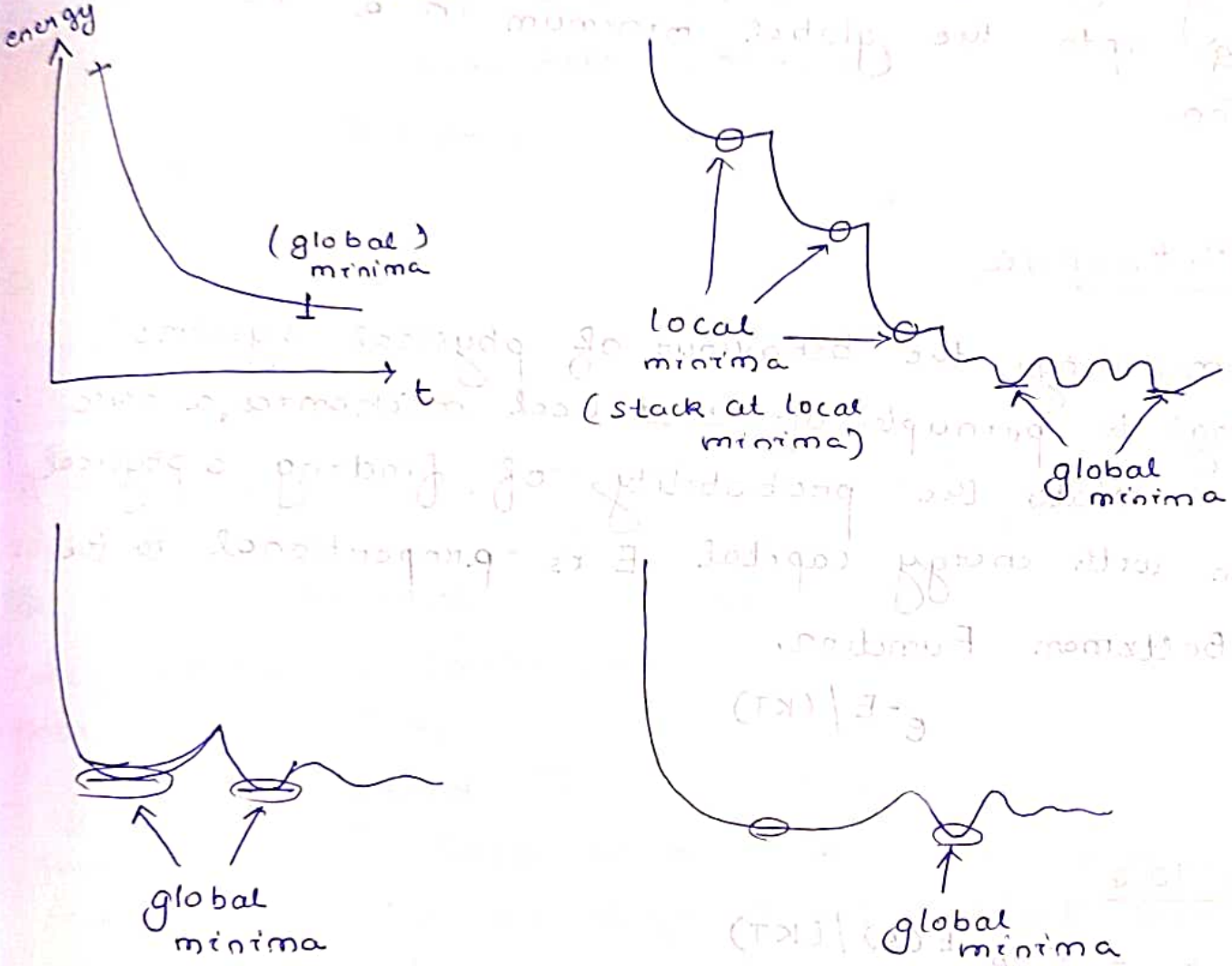
→ All physical systems tend to exhibit high level of potential energy at higher altitudes.

→ They are always working towards reducing their potential energy as they fall down to lower altitudes.

→ When metals and alloys are cooled down post chemical treatments, there is a steep reduction in the amount of energy that the metal is giving out.

Local Search (12.1)

All physical systems tend to exhibit high level potential energy and higher altitudes. They are always working towards reducing their potential energy as they fall down to lower altitudes. When metals and alloys are cooled down post chemical treatments there is a steep reduction in the amount of energy that the metal is giving out.



In computational minimisation problems we have a large set C of possible solutions we also have a cost function 'c'. that measures the quality of a solution.

For a solution set C, its cost is given by C(s) the goal of computational minimisation problem is to find a solution $s^* \in C$ for which. $C(s^*)$ is as small as possible.

The solution 'S' can be obtained by a small modification to analysis of S given as $S \rightsquigarrow S'$

S and S' are called neighbouring solution of each other.

N(s) is the neighbourhood of S consisting of all solution S' such that $S \rightsquigarrow S'$. A local search algorithm is responsible for taking in this setup including algorithms relation and choosing new solution leading up to the global minimum in e iterative function.

12.2 Metropolis

For stimulating the behaviour of physical systems according to principles of statistical mechanics, a basic model states the probability of finding a physical system with energy capital E is proportional to the Gibbs-Boltzman Function.

$$e^{-E/(KT)}$$

Theorem 12.2

Let $Z = \sum_{S \in e} e^{-E(S)/(KT)}$

For a state S, Let $f_S T$ denotes the fraction of the first T state in which the state of simulation is in S then the limit of $f_S T$ as T approaches infinity $= \frac{1}{Z} e^{-E(S)/}$

Metropolis Algorithm

Start with S_0 and constants k, T.

In one step:

Let S be the current solution

Let S' be the chosen solution from neighbour relations

if $C(S') \leq C(S)$

update $S \leftarrow S'$

else with prob $e^{-(C(S')-C(S))/(kT)}$

update $S \leftarrow S'$

otherwise

Leave S unchanged

End if

12.3 Hoffield NNs

A hoffield network is an undirected graph with each edge A configuration S of the network is an assignment of values $-1/+1$ to each node v.

One particular configuration is called the state S_v of node v.

→ If v is joined to v by an edge of (-)ve weight, then u and v tend to have the same state. If u is joined to v by an edge of +ve weight, then u & v tend to have opposite states.

→ The Absolute value $|w_e|$ is called strength of requirement.

→ An edge $E = (u, v)$ is good if the requirement is satisfied by the states of its two end points i.e

$w_e < 0$ and $S_u = S_v$

OR $w_e > 0$ and $S_u \neq S_v$

E is bad.

→ A node v is satisfied in a given configuration. If the total absolute weight of all good edges incident to v is atleast as large as the total absolute weight of all bad edges incident to v

i.e $$\sum_{e=(u,v)\in E} w_e s_u s_v \leq 0$$

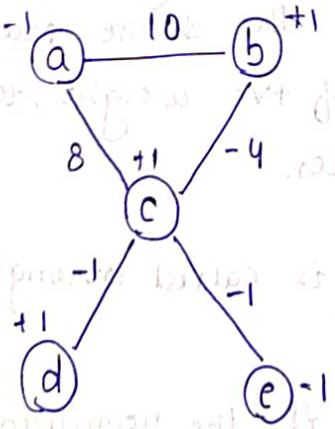
A configuration is called stable if all nodes are satisfied. The objective of local search in this scenario is to find stable configurations in Hoffield networks.

State flipping algorithm to find stable configuration

While the current configuration is not stable, choose an unsatisfied node v.

Flip the state of v

End while



Su	a	b	c	d	e
S_1	1	+1	+1	+1	-1
S_2	+1	+1	+1	+1	-1
S_3	+1	+1	-1	+1	-1
S_4					

We	a-b	a-c	b-c	c-d	c-e
	-10	8	-4	-1	-1
	bad	good	good	good	bad
	good	bad	good	good	good
	good	good	bad	bad	good