Chapter - 11 Approximation Algorithm

(1) Load Balancing problem.
(2) Vertex cover. (6) Knapsack problem.
(3) Set cover.
(4) Center location.
(5) Disjoint path.

Load Balancing problem

→ In load balancing problem we have m no. of identical machines say $M_1, M_2 - - - M_m$ and we have 'n' no. of jobs. Each of the job has a processing time (length / size of the job). Let job 'j' has a processing time t_j. A job must run without interruption on a single machine.

→ A single machine can process only one job at a time.

→ We have to assign each job to some machine which results in an allocation of jobs to machines and it results in each machine receiving a certain load.

→ Load of the machine is just sum of all the processing time of the jobs that have been assigned to that machine.

→ Let $J(i)$ denote the subset of jobs assigned to machine M_i. The load of machine M_i is

$$\boxed{T_i = \sum_{j \in J(i)} t_j}$$

$$\Updownarrow$$

$$\boxed{T_i = \sum_{j \in J(i)} t_j}$$

Goal :- Assign every job to a machine so that the maximum load $\max\{T_1, T_2, - - - T_m\}$ is minimized. That is we seek to minimize the maximum load on any machine.

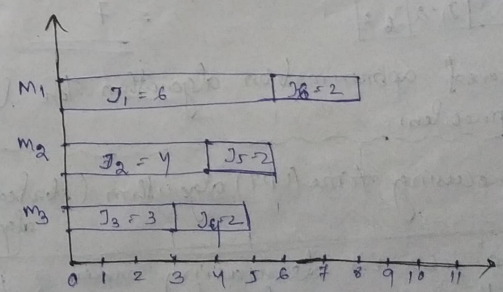
$$\boxed{\max(T_1, T_2, - - -, T_m\} = \max(T_i) \quad \text{for } i = 1, 2, 3, - - m}$$

$\downarrow$

makespan ⇒ maximum load of a machine.

Exp Let there are 3 machines m_1, m_2 and m_3. Then 6 jobs with processing time 6, 4, 3, 2, 2, 2.

job (j)	processing time (t_j)
1	6
2	4
3	3
4	2
5	2
6	2



makespan = $\max\{t_1, t_2, t_3\}$

$= \max\{8, 6, 5\} = \underline{\underline{8}}$

SHOT ON MI A2
MI DUAL CAMERA

Algorithm → Greedy Algorithm for Load balancing (List scheduling Algorithm)

Greedy-Balance:

Start with no job assigned.

Set $T_i = 0$ and $J(i) = \phi$ for all machines m_i.

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

Let m_i be a machine that achieves the minimum load $\min_k T_k$.

Assign job j to machine m_i

Set $J(i) = J(i) \cup \{j\}$

Set $T_i = T_i + t_j$

End for

$\because$ T_i = total load
$J(i)$ = subset of the jobs.

m_1 : $\boxed{J_1 = 6}$

m_2 : $\boxed{J_2 = 4}\boxed{J_5 = 2}$

m_3 : $\boxed{J_3 = 3}\boxed{J_4 = 2}\boxed{J_6 = 2}$

Make-span $= \max\{7, 6, 6\}$
$= 7$

An improved approximation algorithm for load balancing problem.

→ Longest processing time (LPT) algorithm (sorted based algorithm)

Sort the 'n' jobs in decreasing order of processing time and then run greedy-balance (list scheduling algorithm).

Algorithm

Sort-Balance:

Start with no jobs assigned.

Set $T_i = 0$ and $J(i) = \phi$ for all the machine M_i

$O(n\log n)$ Sort the jobs in decreasing order of the processing time t_j.

assume $t_1 \geq t_2 \geq t_3 \cdots \geq t_n$

For $j = 1, 2, 3 \ldots n$

Let M_i be the machine which achieve the minimum load $\min_k T_k$.

$O(n\log n)$ assign the job j to that machine M_i.

Set $J(i) = J(i) \cup \{j\}$

Set $T_i = T_i + t_j$

End for.

Time complexity → Greedy-balance → $O(n)$
Sort-balance → $O(n\log n)$

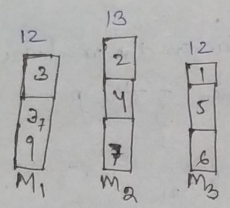
Example

Find the shedule for the jobs arriving with their processing time as 4, 3, 1, 2, 5, 6, 9, 7 on the 3 machine.

Ans

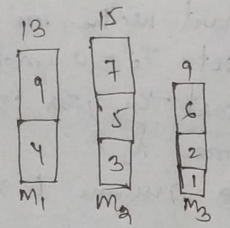
Job	processing time	sort the t_j in decreasing order	sort the jobs corresponding to the processing time t_j
1	4	9	J_7
2	3	7	J_8
3	1	6	J_6
4	2	5	J_5
5	5	4	J_1
6	6	3	J_2
7	9	3	J_4
8	7	1	

By the sorted balance



makespan = max(12, 13, 12)
= 13

By the greedy balance



makespan = max(13, 15, 9)
= 15.

Setcover: A general Greedy Heuristic.

Given a universe set U of 'n' elements, a collection of subsets of U say $S = \{S_1, S_2, S_3, \ldots S_m\}$ where every S_i has an associated weight or cost $w_i > 0$. Find a minimum weight subcollection of S that covers all elements of U. That is the goal is to find a setcover C so that the total cost or weight $\sum_{S_i \in C} w_i$ is minimized.

Example

$U = \{1, 2, 3, 4, 5\}$

then $S_1 = \{1, 3, 4\}$

$S_2 = \{2, 3, 5\}$

$S_3 = \{1, 5\}$

$S = \{S_1, S_2, S_3\}$

$w = \{5, 10, 3\}$

so $w_1 = 5$

$w_2 = 10$

$w_3 = 3$

Set cover with minimum weight cover

that is $\{S_1 \cup S_2\} = \{1, 2, 3, 4, 5\} = U$ (set cover)

and the weight $w = w_1 + w_2 = 5 + 10 = 15$

then $\{S_1 \cup S_3\} = \{1, 3, 4, 5\} \neq U$ (not a set cover)

then $\{S_2 \cup S_3\} = \{1, 2, 3, 5\} \neq U$ (not a set cover)

so there is only one set cover with minimum weight

so $\{S_1 \cup S_2\}$ is the setcover.

Exp-2

$U = \{1, 2, 3, 4, 5\}$

$S_1 = \{1, 3, 4\}$

$S_2 = \{2, 5\}$

$S_3 = \{1, 2, 3, 4\}$

$w_1 = 5$, $w_2 = 10$, $w_3 = 3$

(i) $\{S_1 \cup S_2\} = \{1, 2, 3, 4, 5\} = U$ (setcover)

weight $w = w_1 + w_2 = 5 + 10 = 15$

(ii) $\{S_1 \cup S_3\} = \{1, 2, 3, 4\} \neq U$ (not a setcover)

(iii) $\{S_2 \cup S_3\} = \{1, 2, 3, 4, 5\} = U$ (a set cover)

weight $w = w_2 + w_3 = 13$

so by the 2 setcover the minimum weighted setcover is $\boxed{\{S_2 \cup S_3\} \text{ that is } 13}$.

* our goal is to calculate the minimum weight set-cover by the greedy approach.

Algorithm ($O(i \log n)$)

Greedy-set-cover {

Start with $R = U$ and no sets selected.

While $R \neq 0$

Select set S_i that minimizes $w_i / |S_i \cap R|$

Delete set S_i from R.

Endwhile

Return the selected sets.

}

⇒ Here 'R' represents the set of remaining uncovered elements.

⇒ $\frac{w_i}{|S_i \cap R|}$ represents the cost paid for ~~one~~ covering each new element.

⇒ $\frac{w_i}{|S_i|}$ represents the cost per element cover.

Explanation

Greedy-set-cover (U, S)

{ $R = U$

$A = \phi$ (A is the set of selected subset which is the setcover)

While $(R \neq \phi)$

{

Select set S_i that minimizes $\frac{w_i}{|(S_i \cap R)|}$

Set $A = A \cup \{S_i\}$

Delete set S_i from R.

return A

}

Example

$U = \{1, 2, 3, 4, 5\}$

$S_1 = \{1, 3, 4\}$ | $w_1 = 5$

$S_2 = \{2, 5\}$ | $w_2 = 10$

$S_3 = \{1, 2, 3, 4\}$ | $w_3 = 3$

$R = U = \{1, 2, 3, 4, 5\}$

cost paid for covering each element of $S_1 = \frac{5}{3} = 1.66$

" " " of $S_2 = \frac{10}{2} = 5$

" " " of $S_3 = \frac{3}{4} = 0.75$ (min)

so $A = \{S_3, S_2\}$ $R = \{\cancel{1, 2, 3, 4, 5}\}$ delete the element of the S_2.

⇓ delete the elements of the S_3.

then $A = \{S_3, S_2\}$ is the set cover which has minimum weight.

Exp Let $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$

$S_1 = \{1, 2\}$, $S_2 = \{2, 3, 4, 5\}$, $S_3 = \{6, 7, 8, 9, 10, 11, 12, 13\}$

$S_4 = \{1, 3, 5, 7, 9, 11, 13\}$, $S_5 = \{2, 4, 6, 8, 10, 12, 13\}$

Assume all subsets are having the same weight 2.

$w(S_1 = S_2 = S_3 = S_4 = S_5) = 2$

1st iteration

cost paid for each covering element $s_1 = \frac{2}{2} = 1$

" " " " $s_2 = \frac{2}{4} = 0.5$

" " " " $s_3 = \frac{2}{8} = 0.25$ (min)

" " " " $s_4 = \frac{2}{7} = 0.28$

" " " " $s_5 = \frac{2}{7} = 0.28$

2nd iteration

$R = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13\}$

$s_1 = \frac{2}{2} = 1$

$s_2 = \frac{2}{4} = 0.5$ (min) then delete the element of s_3 from R.

$s_4 = \frac{2}{3} = 0.66$

$s_5 = \frac{2}{2} = 1$

$R = \{1, 2, 3, 4, 5\}$

$\rightarrow R = \{1\}$

3rd iteration

$s_1 = \frac{2}{1} = 2$

$s_4 = \frac{2}{1} = 2$ $\rightarrow$ min then delete the element of s_1 or s_4 from the R.

$s_5 = \frac{2}{0} = \infty$ (stop)

$R = \{\} = \{\} = \phi$

→ then the while loop terminate and the set covers with minimum weight is $\{s_3, s_2, s_1\}$ or $\{s_3, s_2, s_4\}$

→ But the actual optimal soln is $\{s_4, s_5\}$ and greedy algorithm gives almost the optimal solution.

VERTEX COVER

→ The approximation algorithm for Vertex cover problem takes as input an undirected graph G, and return a vertex cover, where size is guaranteed to be no more than twice the size of an optimal vertex cover.

Note

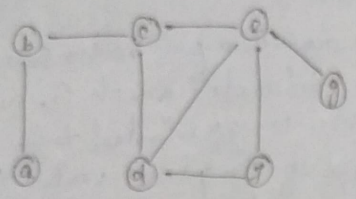
2-approximation algorithm

→ ol gives solutions that never cost more than twice that of optimal if it is a minimization problem, or never provide less than half the optimal value if it is a maximized problem.

Algorithm

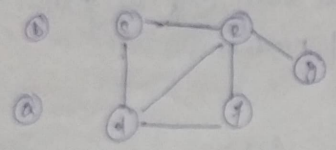
```
Approx-vertex-cover(G)
{
    C = ϕ
    E' = E(G)
    while (E' ≠ ϕ)
    {
        let (u,v) be an arbitrary edge of E'.
        C = C ∪ {u,v}
        remove from E' every edge incident on either
        u or v.
    }
    return C.
}
```

Exp



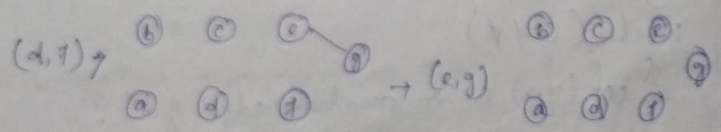
$c = \{b, c\}$

Take any arbitrary edge i.e (b, c) and remove it's adj edge.



Now take (d, f) and remove it's adj edges.

$c = \{b, c, d, f, e, g\}$



optimal soln = $\{e, d, b\}$

The pricing method : Vertex Cover

Given an undirected graph with vertex weight $w_i \geq 0$. Find a minimum weight subset S of the vertices such that each edge is incident to at least one vertex in S. minimum weight subset S means sum of the weights of vertices in S is minimum.

Pricing method

Each edge must be covered by some vertex. Each edge pays a price $p_e \geq 0$ to use a vertex. ~~Sudge~~ Edge incident to vertex i should pay at most w_i in total so, for each vertex ;

$$\boxed{\sum_{e=(i,j)} p_e \leq w_i} \rightarrow \text{fairness cond}^n.$$

Algorithm

Pricing-Vertex-cover (G)

{ For each edge $e \in E$

$p_e = 0$

while there exists an edge for which neither end point is tight pick any such edge e.

increase p_e until an endpoint become tight.

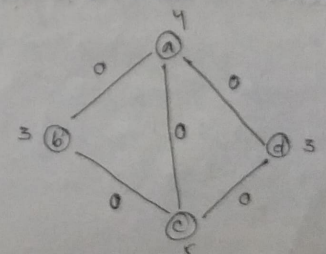
Endwhile

Return the set S of all tight vertices.

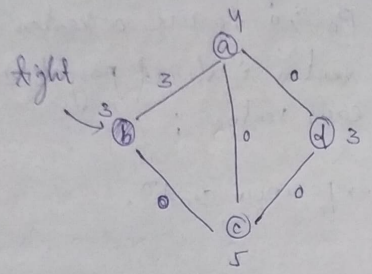
}

Tight Vertex : A vertex is said to be tight if the sum of the prices contributed by its incident edges is equal to its weight.

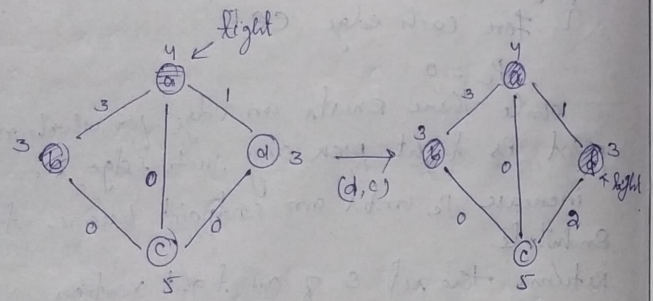
Exp



let us pick an uncovered edge (b, a) and increase the price



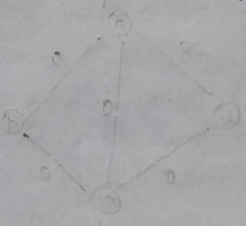
let us pick (a, d) and increase the price



Now there is no such edge with no tight vertex at both end. Hence the vertex loop gets terminated

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

→ vertex cover (all tight vertices)



Center selection problem

→ Given a set S of m sites $s_1, s_2, \ldots s_n$ select k centers c so that "maximum" distance from a site to nearest center is minimized".

C → the set of k centers

→ $C = \{c_1, c_2, \ldots c_k\}$

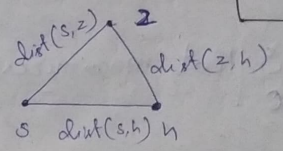
→ This problem visible as a complete graph

→ The distance is the standard Euclidian distance with following properties.

(i) $dist(s, s) = 0 \quad \forall\ s \in S$

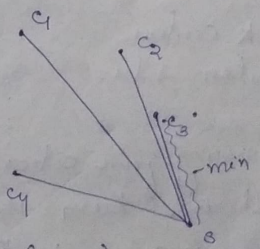
(ii) $dist(s, z) = dist(z, s) \quad \forall\ z, s \in S$ (symmentric property)

(iii) $dist(s, z) + dist(z, h) \geq dist(s, h) \quad \forall\ z, s, h \in S$ (Triangle inequality)



→ Defn :- Distance of site s to the centers is defined as $\boxed{dist(s, C) = \min_{c \in C} dist(s, c)}$

Exp:



$C = \{c_1, c_2, c_3, c_4\}$

so $dist(s, C) = \min_{c \in C} dist(s, c)$

$= dist(s, c_3)$

Defn ⇒ The set C forms an "r-cover" if each site is ~~within~~ within distance at most 'r' from one of the centers.

⇒ The minimum 'r' for which C is an r-cover will be called the "covering radius" of C and denoted by r(C).

Objective

Select set C of k centers for which r(C) is as small as possible.

Greedy Algorithm for center selection.

1. Assume k ≤ |S|
2. Select any site s and let C = {s}
3. while |C| < k
4. select a site s ∈ S that maximizes dist(s, C)
5. Add site s to C.
6. Endwhile
7. Return C as the selected set of sites.

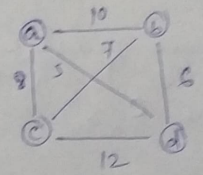
where S = set of n sites
C = set of k centers
s = a particular site.

Example

Let us consider the following four cities, {a, b, c, d} and the distance between them. How to place 3 ATM, among these 4 cities so that the maximum distance of a city to an ATM is minimized.

dist(a,b) = 10, dist(a,c) = 8, dist(a,d) = 5,
dist(b,c) = 7, dist(b,d) = 6, dist(c,d) = 12

Ans



Step 1 let C = {a}
|C| = 1
k = 3

Step 2 dist(c, a) = 8
dist(d, a) = 5
dist(b, a) = 10 (maximum)

so add the site b to the C

C = {a, b}
|C| = 2
k = 3

Step 3 then select a site s ∈ S that maximizes dist(s, C)

then check

dist(c, a) = 8
dist(c, b) = 7

$$dist(c, C) = \min_{c \in C} dist(c, c) = 7$$

⇒ dist(c, C) = 7

then dist(d, a) = 5 (min) = dist(d, C)
dist(d, b) = 6

then dist(c, C) = 7 (maximum)

dist(d, C) = 5

so we can select c because of dist(c, C) = 7 (max)

then add the site c to the C.

C = {a, b, c}

Step-1 |C| = 3

k = 3

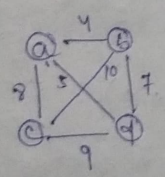
while loop terminated so the C = {a, b, c} Ans.

Exp At there are 4 sites with their distance matrix as :

$$\begin{array}{c|cccc} & a & b & c & d \\ \hline a & 0 & 4 & 8 & 5 \\ b & 4 & 0 & 10 & 7 \\ c & 8 & 10 & 0 & 9 \\ d & 5 & 7 & 9 & 0 \end{array}$$

select 3 centers so that the maximum distance of site to a center is minimized.

Ans



Step-1 k = 3

S = {a, b, c, d} ⇒ |S| = 4

C = {a} (initially selected)

while (|C| < k) ⇒ 1 < 3 (TRUE)

dist(c, a) = 8 (max)

dist(b, a) = 4

dist(d, a) = 7

so add the site c to the center C.

⇒ C = {a, c}

Step-2 k = 3

C = {a, c}

|C| = 2

while (2 < 3) (TRUE)

then check which site is selected.

so dist(b, a) = 4 (min) = dist(b, C)

dist(b, c) = 10

similarly dist(d, c) = 9

dist(d, a) = 5 (min) = dist(d, C)

then dist(b, C) = 4

dist(d, C) = 5 (minimizes dist(s, C))

then add the site d to the center C.

⇒ C = {a, c, d}

Step-3 |C| = 3

k = 3

while (3 < 3) false

the loop is terminated then

return the C = {a, c, d} Ans.

Knapsack problem

→ There is i items and each items have the weight w_i and profit/value v_i. W = knapsack capacity.

→ our objective that maximizes the profit/value.

$$\boxed{\text{Maximizes } \sum_{j=1}^{i} v_j}$$

subject to condition $\boxed{\sum_{j=1}^{i} w_j \leq W}$

Recurrence formula to get the soln

If $V > \sum_{i=1}^{n-1} v_i$, then

$$M(n,V) = w_n + M(n-1, V-v_n)$$

otherwise,

$$M(n,V) = \min\left(M(n-1,V),\ w_n + M[n-1, \max(0, V-v_n)]\right)$$

A new dynamic Algorithm for knapsack problem.

Algo Knapsack(n)

{ Array $M[0 \ldots n, 0 \ldots V]$ // n is the no. of items

for $i = 0, 1, \ldots, n$ $\qquad V$ = sum of all values $= \sum_{i=1}^{n} v_i$

$M[i,0] = 0$

EndFor

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

for $V = 1, 2, \ldots, \sum_{j=1}^{i} v_j$ {

if $\left(V > \sum_{j=1}^{i-1} v_j\right)$ then

$M[i,V] = w_i + M[i-1, V-v_i]$

endif

else

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

EndElse

End for

End for

Return maximum value V such that $M[n,V] \leq W$.

}

Example let $n = 4$, $W = 35$

$n \rightarrow$	1	2	3	4
w_i	12	12	13	14
v_i	1	2	1	1

then $V = 1+2+1+1 = 5$

so

$V \rightarrow$ / $\downarrow$	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	12				
2	0	12	12	24		
3	0	12	12	24	37	
4	0	12	12	24	37	51

step-1 when $i = 1$ then $V = 1$

if $\left(1 > \sum_{j=1}^{1} v_j\right)$ (True)

(invalid)

so $M[1,1] = w_1 + M[1-1, 1-1]$

$= 12 + 0 = 12$

step-2 when $i = 2$ then $V = \sum_{j=1}^{2} v_j = 2+1 = 3$

$V = 1, 2, 3$

so we conclude $M[2,1] = 12$

$M[2,2] = 12$

$M[2,3] = 24$

$i = 2$
$v = 1$
if $(1 > 1)$ false)
so go to the else part
then $m[2,1] = \min(m[1,1],\ 12 + m[1, \max(0,0)])$
$= \min(12,\ 12 + m[1,0])$
$= \min(12,\ 12 + 0) = 12$

→ $m[2,2]$ → if $(2 > \sum_{j=1}^{1} v_j)$ (True)
$m[2,2] = w_2 + m[2-1,\ 2-2]$
$= 12 + m[1,0] = 12 + 0 = 12$

for $m[2,3]$
if $(3 > 1)$ (True)
$m[2,3] = w_2 + m[1,\ 1] = 12 + 12 = 24$

Step-3
when $i = 3$ then $V = \sum_{j=1}^{i-1} v_j = v_1 + v_2 + v_3$
$= 1 + 2 + 1 = 4$
so $V = 1, 2, 3, 4$

→ then $m[3,1]$ → if $(1 > \sum_{j=1}^{i-1} v_j)$ → $(1 > 3)$ (false)
so else part executed
→ $m[3,1] = \min(m[2,1],\ 13 + m[3, \max(0,0)])$
$= \min(12,\ 13 + m[2,0]) = \min(12, 13)$
$= 12$

→ $m[3,2]$ → if $(2 > 3)$ (false)
→ $m[3,2] = \min(m[2,2],\ 13 + m[2, \max(0,0)])$
$= \min(12, 13) = 12$

→ $m[3,3]$
if $(3 > 3)$ (false)
$m[3,3] = \min(m[2,3],\ w_3 + m[2, \max(0, 3-1)])$
$= \min(24,\ 13 + m[2,2])$
$= \min(24,\ 13 + 12) = 24$

→ $m[3,4]$
if $(4 > 3)$ (True)
$m[3,4] = w_3 + m[3-1,\ 4-1] = 13 + m[2,3]$
$= 13 + 24 = 37$

Step-4
when $i = 4$ then $V = \sum_{j=1}^{4} v_j = 1+2+1+1 = 5$
$V = 1, 2, 3, 4, 5$

$m[4,1]$
if $(1 > \sum_{j=1}^{i-1} v_j)$ → if $(1 > 4)$ (false)
$m[4,1] = \min(m[3,1],\ w_4 + m[3, \max(0,0)])$
$= \min(12,\ 14 + 0) = 12$

$m[4,2]$
if $(2 > 4)$ (false)
$m[4,2] = \min(m[3,2],\ w_4 + m[3, \max(0,0)])$
$= \min(12,\ 14 + 0) = 12$

$m[4,3]$ if $(3 > 4)$ (false)
$m[4,3] = \min(m[3,3],\ w_4 + m[3, \max(0,2)])$
$= \min(24,\ 14 + 12) = 24$

$m[4,4]$

if $(4 > 4)$ (False)

$m[4,4] = \min(m[3,4], w_4 + m[3, \max(0,3)])$

$= \min(37, 14 + m[3,3])$

$= \min(37, 14 + 24) = 37$

$m[4,5]$

if $(5 > 4)$ (True)

$m[4,5] = w_4 + m[3,4] = 14 + 37 = 51$

Then minimum profit is 24 which is $24 \uparrow 35$.

Ans.

Trick

$v \rightarrow$, $i \downarrow$

i \ v	0	1	2	3	4	5
0	0					
1	0	12				
2	0	12	12	24		
3	0	12	12	24	37	
4	0	12	12	24	37	51

$i = \{1\}$

$w_i = 12$

$i = \{1,2\}$

$m[2,1] = 12$

$m[2,2] = 12 + 0 = 12$

$m[2,3] = 12 + 12 = 24$

$i = \{1,2,3\}$

$v = 1 \rightarrow 12$

$v = 2 \rightarrow 12$

$v = 3 \rightarrow \max(13, 24) = 24$

$v = 4 \rightarrow \max(14, 12 + 24)$

$\rightarrow \max(14, 37) = 37$

$i = \{1,2,3,4\}$

$v = 1 \rightarrow 12$

$v = 2 \rightarrow 12$

$v = 3 \rightarrow \max(13, 24) = 24$

$V_4 = \max(14, 12 + 24)$

$= 37$

$V_5 = \max(37, 51)$

$= 51$.

Q

items	w_i	v_i
1	12	100000 / 100000 = 1
2	12	200000 / 100000 = 2
3	13	100000 / 100000 = 1
4	14	100000 / 100000 = 2

Similarly process the algorithm.

Example

let $m = 5$

$w = 11$

items	w_i	v_i
1	1	934221 = 1
2	2	5956342 = 7
3	5	17819013 = 20
4	6	21217800 = 23
5	7	27,343,199 = 30

$\left[\frac{27343199}{934221}\right] = \lfloor 29.2 \rfloor = 30$

$\Longrightarrow$ original instance of knapsack problem.

Rounded the nearest multiple that is 1,000,000

so

items	w_i	v_i
1	1	1000000
2	2	6000000
3	5	18000000
4	6	22000000
5	7	28000000

Rounded instance

scale down

v_i
1
6
18
22
28

Scaling instance

Local Search

$\Rightarrow$ In computer science local search is a heuristic method for solving computationally hard optimization problem

$\Rightarrow$ a heuristic method or a heuristic technique is any approach to problem solving that employs a practical

method that is not garented to be optimal or perfect, but is sufficient for reaching an immediate short term goal or approximation.

⇒ Local search algorithm move from solution to soln in the space of candidate solution (the search space) by appling local changes until an optimal soln is found or a time bound is ~~elapsed~~ elapsed.

⇒ Local search algorithms explores the space of the possible solution in sequencial fashion moving from the current soln to a near by one.

⇒ Some problems where local search has been applied as

(1) Vertex cover problem :- in which a solution is a vertex cover of a graph, and the target is to find solution with a minimal number of vertices.

(2) Travelling Salesman problem :- in which a solution is a cycle containing all nodes of the graph and the target is to minimize the total length of the cycle.

(3) Hopefield Neural Network problem - for which finding stable configurations in Hopefield network.

(4) Simulated Annealing

(5) metropolis Algorithm and etc.

An application of vertex cover problem

Given graph $G = (V, E)$; the set C of possible soln consists of all subsets S of V that form vertex cover.

Hence, we have always $V \in C$

the cost $C(S)$ of a vertex cover S will be it's size. minimizing the cost of a vertex cover S is the same as finding one minimum size.

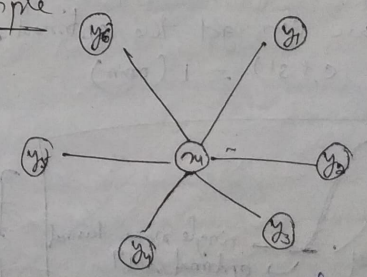
(where S is the vertex cover)

⇒ The local search algorithm uses a simple neighbor relation.

Neighbor relation

⇒ we say that $S \sim S'$, if S' can be obtained from S by adding or deleting a single node.

For example

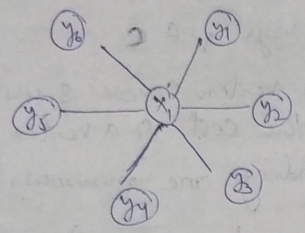


Gradient descent :- Let S denote the current soln. If there is a neighbor S' of S with strictly lower cost, then replace S with the neighbor whose cost is as small as possible. Otherwise, terminate the algorithm.

Continue the example

we know that $\{x_4\} \rightarrow$ vertex cover (optimal soln)

by the Gradient Descent $S = V = \{x_1, y_1, y_2, y_3, y_4, y_5, y_6\}$



$c(S) = 7$

step-1 if y_1 is removed

then $c(S') = 6$

so $c(S') < c(S) \Rightarrow 6 < 7$ (True)

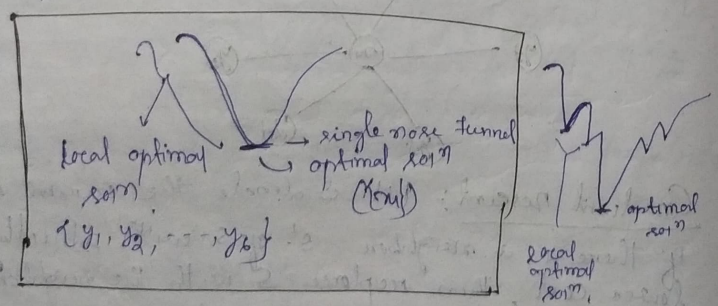
step-2 if y_2 is removed

then $c(S'') = c(S') \Rightarrow 5 < 6$ (True)

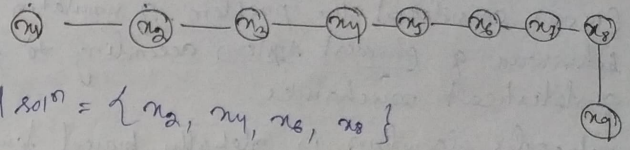
step-3 similarly we can proceed and

at last we can get the optimal soln that

is $\{x_4\}$ so $c(S') = 1$ (min)



Exp-2



optimal soln $= \{x_2, x_4, x_6, x_8\}$

local optimal soln $= \{x_1, x_3, x_5, x_7, x_9\}$

(or)

$\{x_1, x_2, x_4, x_5, x_7, x_8\}$

(or)

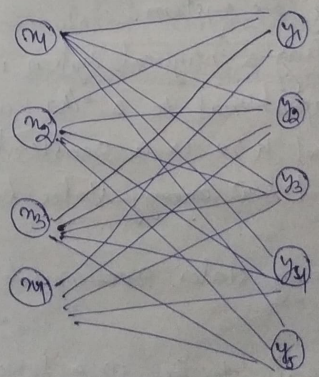
$\{x_1, x_2, x_3, x_4, x_6, x_7, x_8, x_9\}$

In vertex cover problem

Start with $S = V$

if there is a neighbor S' that is a vertex cover and has lower cardinality or cost, replace S with S'.

Q



initially

$S = V$

$= \{x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_5\} \Rightarrow |S| = 9$

$S' = \{x_1, x_2, x_3, x_4\}$

$|S'| = 4$

$|S'| < |S|$ so replace it

optimal soln $= \{x_1, x_2, x_3, x_4\}$

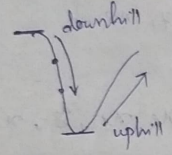
local optimal soln $= \{y_1, y_2, y_3, y_4, y_5\}$

(or) $\{x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4, y_5\}$

Metropolis algorithm

⇒ metropolis, Rosenbluth, Rosenbluth, Teller and Teller (1953) considered the problem of simulating the behaviour of physical system according to principles of statistical mechanics.

⇒ metropolis algorithm is globally biased towards "downhill" steps, but occasionally makes "uphill" steps to break out of local minima.



⇒ metropolis algorithm use Gibbs - Boltzmann function.

Gibbs - Boltzmann function

⇒ It is a probability distribution or probability measure that gives the probability that a system will be in a certain state as a function of that state's energy and the temperature of the system.

⇒ The probability of finding a physical system in a state with energy E is proportional to $e^{-E/kT}$, where $T>0$ is temperature and $k>0$ is a constant.

⇒ For T large, high and low energy states have same probability.

⇒ For T small, low energy states are much more probable.

⇒ The system is more likely to be in a lower energy state than higher one since for any temperature $T>0$, function is monotone ~~as~~ decreasing function of energy E.

Algorithm

(1) Start with an initial solution S_0 and constants k and T.
(2) Let s be the current solution.
(3) Let s' be choosen uniformly at random from neighbors of s.
(4) If $C(s') \leq C(s)$
(5) update $s \leftarrow s'$
(6) Else
(7) with probability $e^{-(C(s')-C(s))/kT}$
(8) update $s \leftarrow s'$
(9) otherwise
(10) leave s unchanged.
(11) End If

$\Delta E = E(s') - E(s) = C(s') - C(s)$

$C(s') = E(s')$, $C(s) = E(s)$

Simulated annealing

⇒ an algorithmic technique known as simulated annealing is used to mimic metropolis algorithm computationally.

⇒ simulated annealing works by running the metropolis algorithm with gradually decreasing the value of T over the course of the execution.

⇒ The exact way in which T is updated is called a cooling schedule and a number of ~~concerns~~ considerations go into the design of the cooling schedule.

(1) If T is large, then probability of accepting an uphill move is large.

(2) if T is small, then uphill moves are almost never accepted. (downhill moves are accepted)