Cooperative and Adaptive Algorithms

Simulated Annealing (SA)
Examples

SA for the Graph Coloring Problem, [1]

- Given a graph with n nodes and m edges. Allocate a color to each node such that adjacent nodes are not give the same color.
- The objective is to find an allocation which uses the smallest number of colors.
- Another variant is given a number of colors, k, and a graph, can we answer the question: does the graph have k-coloring?
- NP-hard problem

Applications that can be represented as graph coloring [2]

- Register assignment in compiler design in order to execute programs efficiently.
- K registers, n variables, can’t assign two variables currently in use to the same register.
- Represent variables as node and two values will have an edge if they are currently in use.

The Graph Coloring Problem

Wavelength assignment

- Wavelength assignment in wireless communication (or frequency assignment for TV or radio channels)
  - K wavelengths or (frequencies), n devices, no close devices (or channels) should be assigned the same wavelength (frequency)
  - Devices become nodes of the graph with edges if they shouldn't be assigned the same wavelength, colors represent the wavelengths

Scheduling with conflict constraints

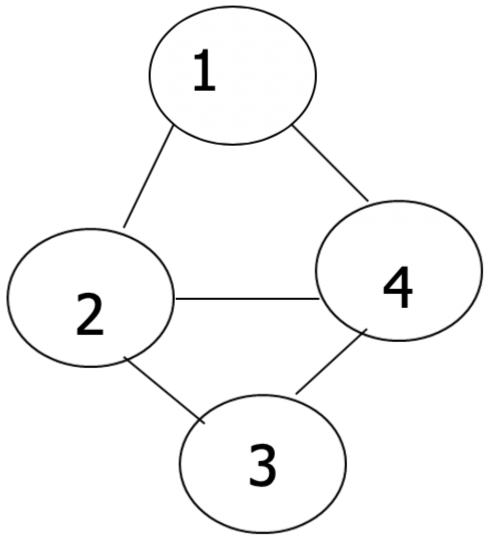
Scheduling of jobs on processors with some jobs requiring the same resource at the same time

- Jobs are nodes of the graph with edges if they have conflict in demand of resource
- Coloring of the nodes will be a feasible schedule.

Graph representation

- Adjacency matrix

Adjacency List



$$\begin{matrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{matrix}$$

L[1]= {2,4}

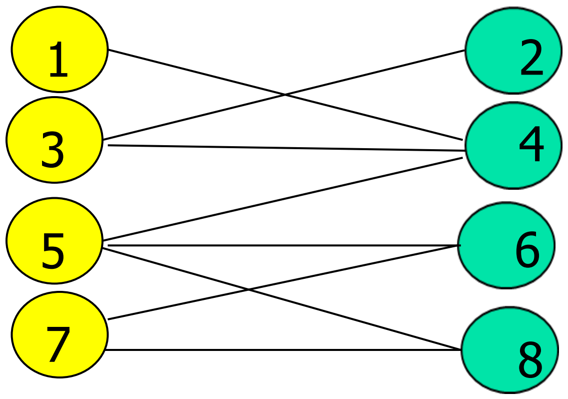
L[2]={1,3,4}

L[3]={2,4}

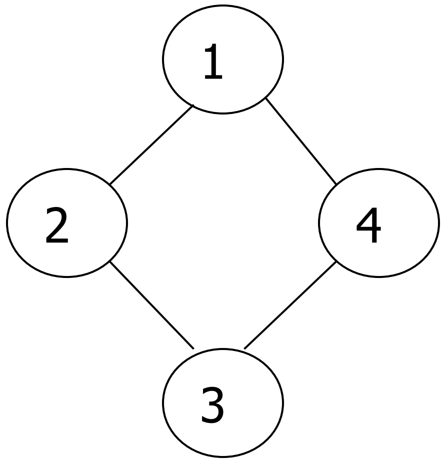
L[4]={1,2,3}

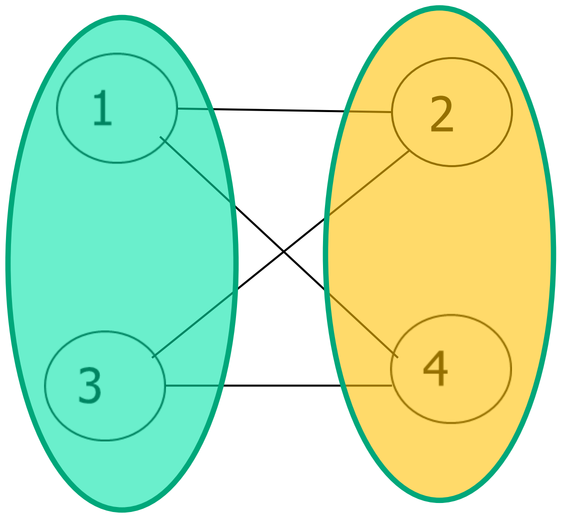
2-coloring problem

- A graph is 2-colorable if it is bipartite
- Bipartite graph is a graph that its nodes can be partitioned into sets x and y in such a way that every edge has one end in x and the other in y.
- X and Y are called independent sets.



Bipartite Graphs?



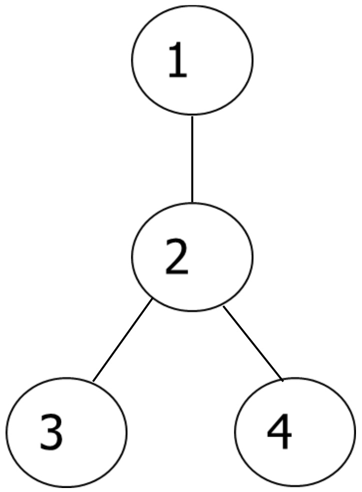


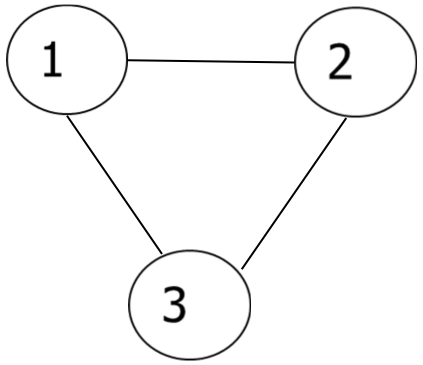
Yes {1,3}, {2,4}

Bipartite Graphs?

A bipartite graph can't contain an odd cycle.

One way to check that is to use BFS (or DFS) of the graph and check for odd cycles. Or one can alternate colors between the levels of the BFS and then check if an edge has the same color on both ends

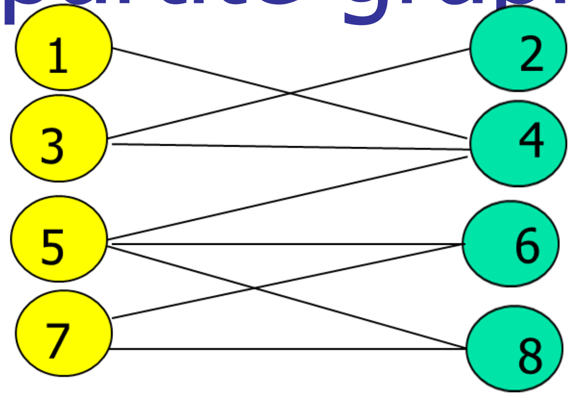


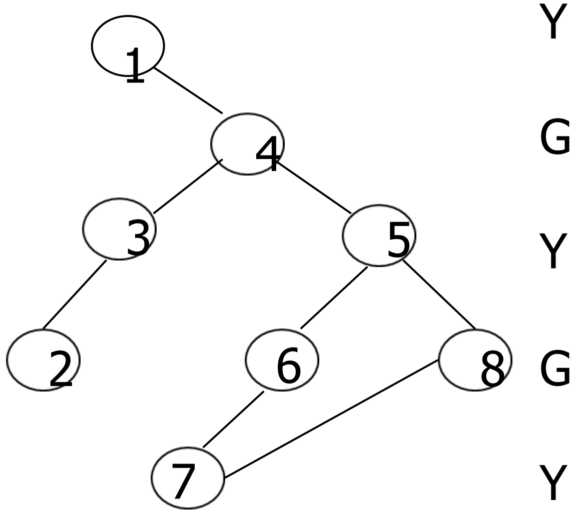


Yes {1,3,4}, {2}

No, it has odd length cycle

Bipartite graph



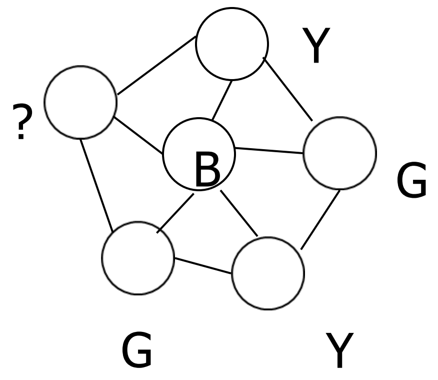


O (n+m)

2- coloring can be determined in O(n+m)

3-coloring

- One may think that any graph that is not 3-colorable will contain 4 nodes that are mutually adjacent and hence require 4 colors. But this not true . It is possible to have graphs that doesn't satisfy this and also not 3 -colorable

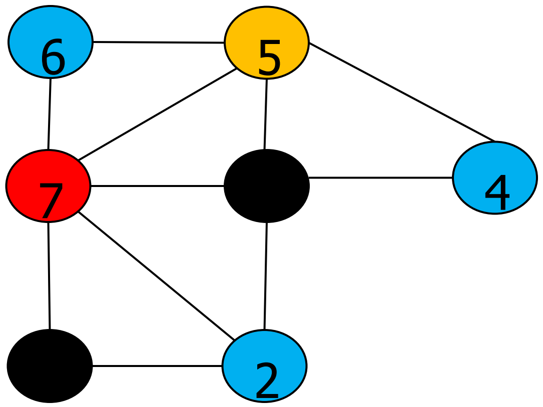


It is NP-complete
Generalizes to K>3

Finding the min number of colors to color a graph K =3 or larger using SA

- Start with no-optimal solution and then try to improve it.
- Example

4 colors can we do better?



No same colours are adjacent.

Node coloring

- Independent set correspond to nodes of the same color- they are not connected.
- For the example with this coloring we have

```
{1,3}, {2,4,6},{5},{7}
  G         B       Y    R
```

- One can define a neighborhood operator as changing the color of some nodes between 2 subsets, but it has still to be feasible (satisfying the constraint no edge between nodes of the same color).

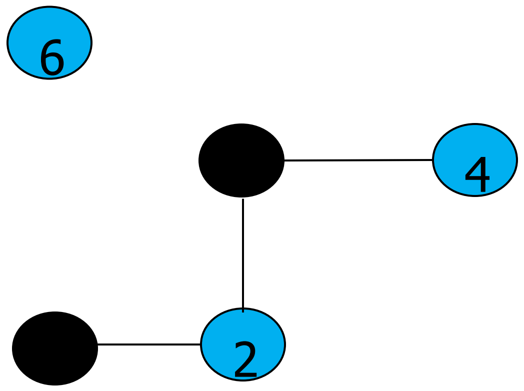
{1,3}, {2,4,6}, {5}, {7}
G B Y R

{1,3},{2,4,6} we can look at part of the graph involving these nodes and these colors and see if we can change the color of some of these nodes.

we can change the color of node 6 to G
without affecting other nodes producing

{1,3,6}, {2,4}
G B

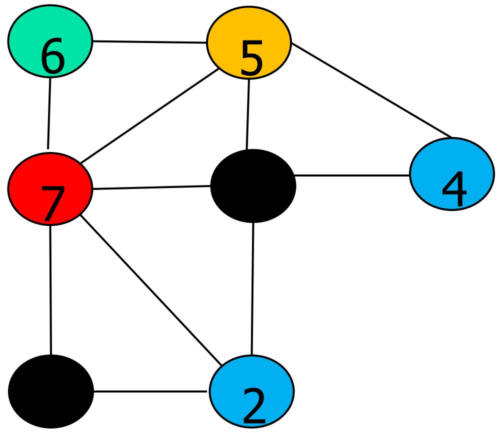
cost function is K number of colors
still the total number of colors is 4
so it is a non-improving solution,
apply the probability of acceptance.
it will be accepted (why).



{1,3,6},{2,4},{5},{7}
G B Y R

- Consider another Neighbor by looking at other 2 sets
- {2,4}, {5}
  B Y

Not feasible to make 5 B or
4 Y
It is feasible to make 2 Y
We get {4}, {2,5}
Cost still 4 so non-improving
Solution again apply acceptance probability
It will be accepted.



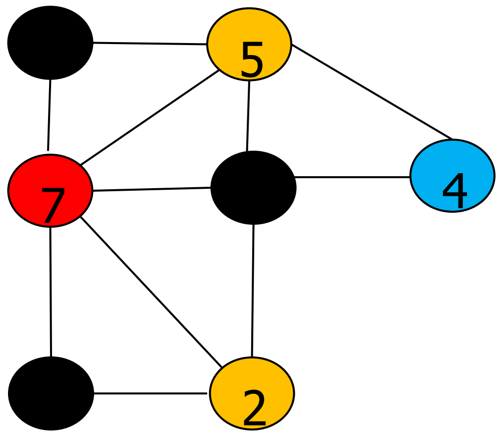
{1,3,6},{4},{2,5},{7}
G B Y R

- Consider {2,5},{7}
  Y R

Not feasible to make 7 Y
Or 2 or 5 R

- Consider {1,3,6},{7} not

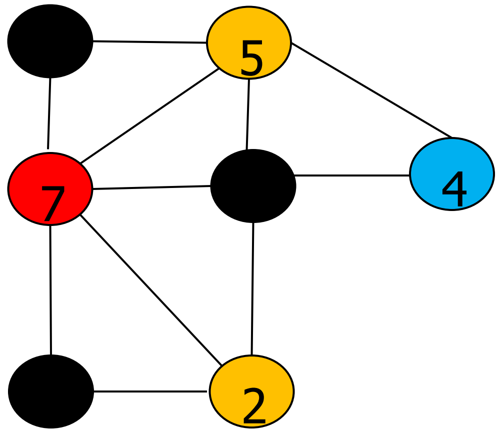
Feasible to change colors



{1,3,6},{4},{2,5},{7}
G B Y R

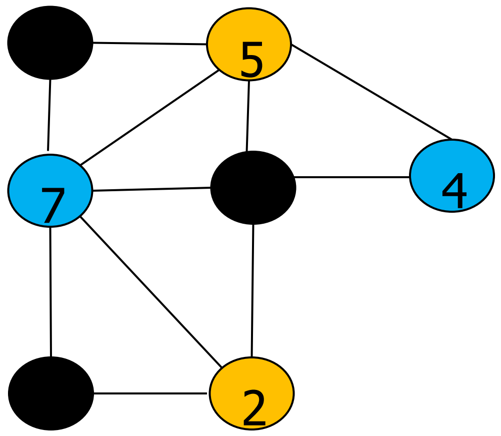
- Consider {4},{7}

7 can be B making an empty
Set which will improve the
Cost function to be 3 colors.



{1,3,6},{4},{2,5},{}
G B Y R

Final solution



Observations

- Cost function as the number of colors will not change for many moves. It only change when one subset becomes empty. Sometimes it only occurs at an optimal solution
- That makes SA wonders randomly till this happens. The cost function is not giving any guidance to the search
- To check the feasibility of changing a color of node one has to check all connected nodes to such node to see if such change will require changing the color of other nodes. This can be computationally demanding

Suggestions

- Use a cost function that can bias choices of sets that are of not of the same size (large and small rather than equal size).

One can use a penalty to reflect this bias such as

$$-\sum_{i=1}^{k} |V_i|^2, where V_i$$ is the number of nodes in set *i*

(e.g for 2 sets of size 10 each we get -200, that –(100+100), while for one of size 15 and the other of size 5 we gat -250, that is –(225+25))

- To avoid checking feasible solution we can relax the restriction and use a penalty function for the number of edges that are not satisfying the constraint.

Example of such function can be

$$\sum_{i=1}^{k} 2|V_i||E_i|,$$ where $|E_i|$ is the number of edges with

both ends in V_i . A feasible solution will have this function equal zero.

This cost function is also suitable for applications where we are give a fixed number of color and we looking for a feasible solution.

Scheduling problems

http://riot.ieor.berkeley.edu/riot/Applications/Scheduling/objectives.html

- Scheduling jobs to be processed on machine(s)
- Objective functions
- **Makespan**

Let C_i denote the completion time of the ith job in a batch of n jobs given. The makespan, C_{max}, is defined as,

$C_{max} = \max (C_1, C_2, \ldots , C_n)$

- The makespan is the completion time of the last job.
- A common problem of interest is to minimize C_{max},
- This criterion is usually used to measure the level of utilization of the machine.

Objective Functions

- **Total (Weighted) Completion Time**

Let C_i denote the completion time of the ith job in a batch of n jobs given. The total completion time is defined as,

$$\sum_{i=1}^{n} C_i$$

- The total completion time is the sum of all the completion times of the jobs. It is commonly referred to as the flow time.

Let w_i denote the weight assigned to the ith job in the a batch of n jobs given. The total weighted completion time is defined as,

$$\sum_{i=1}^{n} w_i C_i$$

A common problem is in minimizing the total (weighted) completion time. This problem allows one to find an indication to the total holding or inventory caused by the schedule.

Objective Functions

- **Lateness**

Difference between completion time and the due time. $L_i=C_i-d_i$, where C_i is the completion of job i and d_i is the due date of job i.

Some of the problems involving lateness are:

1. Minimizing the total lateness. The total lateness is defined as,

$$\sum_{i=1}^{n} L_i$$

2. Minimizing the total weighted lateness. Let w_i denote the weight assigned to the ith job. The total weighted lateness is defined as,

$$\sum_{i=1}^{n} w_i L_i$$

3. Minimizing the maximum lateness, L_{max}. L_{max} is defined as,

$L_{max} = \max (L_1, L_2, \dots , L_n)$

Objective Functions

- **Tardiness**

The tardiness of job i, T_i, is defined as $T_i=\text{Max}(0, C_i-d_i)$, where C_i is the completion of job i and d_i is the due time of job i.

Some of the problems involving tardiness are:

1. Minimizing the maximum tardiness

   Min (max (T_i))

2. Minimizing the number of tardy jobs. The number of tardy jobs is defined as,

$$N_T=\sum_{i=1}^{n} q(T_i)$$

where q(x) = 1 if x > 0

q(x) = 0 otherwise

Tardiness (cont'd

- Minimizing the total weighted tardiness. Let w_i denote the weight assigned to the ith job. The total weighted tardiness is defined as,

$$\sum_{i=1}^{n} w_i T_i$$

Machine environments

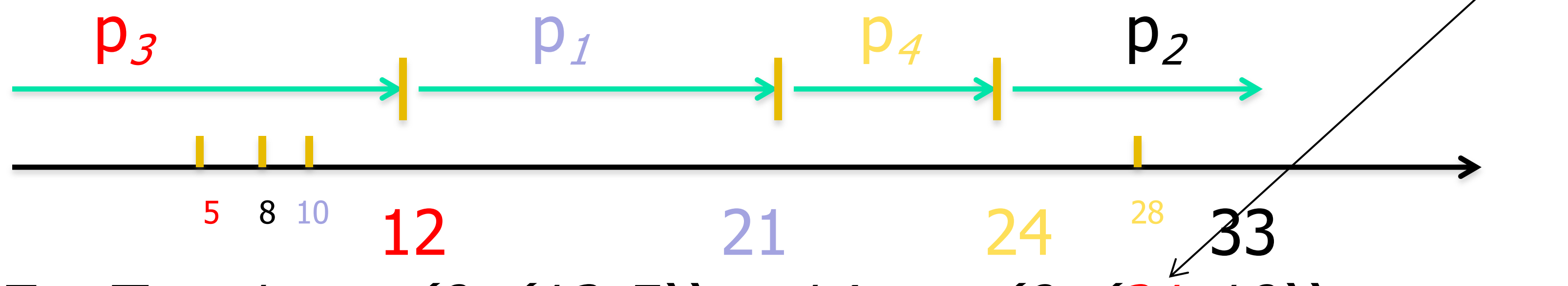
- **Single Machine**: Only one machine is available to process jobs. Each job has a single task. Every job is performed on the same machine.
- **Parallel Machines**: Multiple machines are available to process jobs. The machines can be identical, of different speeds, or specialized to only processing specific jobs. Each job has a single task.

Example

single machine, minimize total weighted tardiness

	jobs	1	2	3	4
processing	p_j	9	9	12	3
due	d_j	10	8	5	28
weight	w_j	14	12	1	12

Let an initial solution be 3, 1, 4, 2



$\Sigma w_j T_j$ = 1·max(0, (12-5)) + 14·max(0, (21-10)) + 12·max(0,(24-28))+ 12 ·max(0,(33-8)) = 461

SA

jobs	1	2	3	4
p_j	9	9	12	3
d_j	10	8	5	28
w_j	14	12	1	12

- Neighborhood by swapping the order of jobs. Select one at random
- Use geometric temperature reduction

$\alpha = .9, \; t_0 = 0.9$

- Set no of iterations at the same temperature=2

Iteration 1 Consider the following neighbor (initial 3, 1, 4, 2 @*461*)

1, 3, 4, 2

$\Sigma w_j T_j$ = *14.max(0,(9-10))+1.max(0,(21-5))+*

12.max(0,(24-28)+12.max(0,(33-8))=316

ΔC = -145 then we accept the new solution as it is improving

jobs	1	2	3	4
p_j	9	9	12	3
d_j	10	8	5	28
w_j	14	12	1	12

1, 3, 4, 2@316

Iteration 2

- Generate a neighbor (say) 1, 4, 3, 2

  $\Sigma w_j T_j$ = *14.max(0,(9-10))+12.max(0,(12-28))+ 1.max(0,(24-5)+12.max(0,(33-8)) = 319*
- ΔC *= +3 i.e not-improving, apply the acceptance criteria.*
- *Calculate* $e^{-\Delta c/t}$ = $e^{-(319-316)/.9}$ *= 0.035*
- Generate a random number x between (0,1), say it was x=0.01which is less than the acceptance probability, then we accept the solution.
- So we accept a non-improving solution and we exceeded the max number of iterations at this temperature, so decrease the temperature
- $t = t \times \alpha$, *t= 0.9x0.9=0.81*

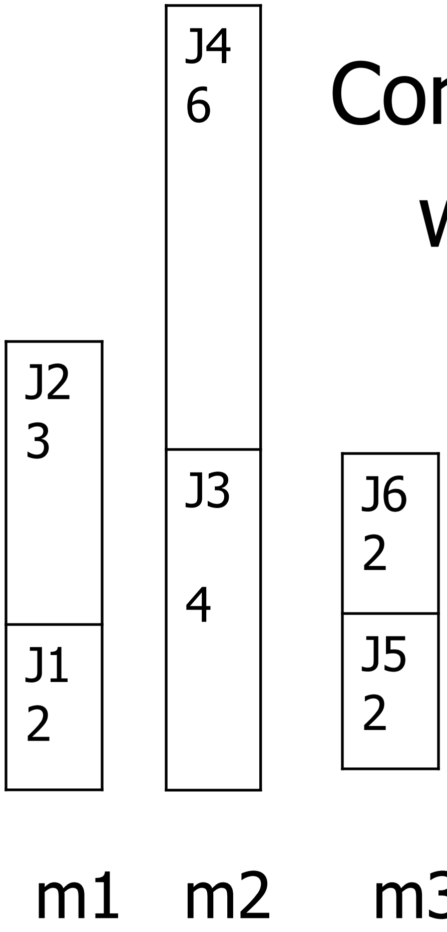
Multiple Machines Minimum Makespan Problem

- Given processing times for n jobs, $p_1, p_2, \ldots, p_n$, and an integer m, the Minimum Makespan Scheduling problem is to find an assignment of the jobs to m machines, $M_1, M_2, \ldots, M_m$ so that the final completion time (the makespan) is minimized.

Example

- 6 jobs with processing times:p1,p2,p3,p4,p5,p6= (2,3,4,6,2,2)
- 3 machines m1,m2, m3.
- All machines should be assigned jobs.

P=(2,3,4,6,2,2)



- Consider the following schedule we can code this schedule as

Jobs 1 and 2 -→Machine m_1 Jobs 3 and 4 -→Machine m_2

(1,1,2,2,3,3) indicating machine assigned to the corresponding jobs

Makespan=10

Generating Neighbors

- We can consider permutation of the machines numbers for the 6 jobs, e.g
- (1,1,1,1,2,3), (1,1,1,2,1,3),…… there are a bit less than 3^6 as we exclude unassigned machines.
- Notice that swap operation from the initial solution doesn't cover feasible solutions

SA

Initial soln (1,1,2,2,3,3)

- Select a permutation at random
- Use geometric temperature reduction

α =.9, t_0 =0.9

- Set no. of iterations at the same temperature=2

Iteration 1 Consider the following neighbor

  - (1,1,1,1,2,3) for p(2,3,4,6,2,2) will produce a makespan of 15 which is worse than the initial solution
  - Calculate $e^{-\Delta c/t} = e^{-5/0.9} = 0.0004$
  - Generate a random number x between (0,1), say it was x=0.2 which is larger than the acceptance probability so we reject the solution

SA

Iteration 2

- Generate another neighbor, say (1,2,3,1,2,3) with makespan of 8
- This is an improvement of the current solution so we accept it.
- And so on.
- Optimal makespan for this problem is 7

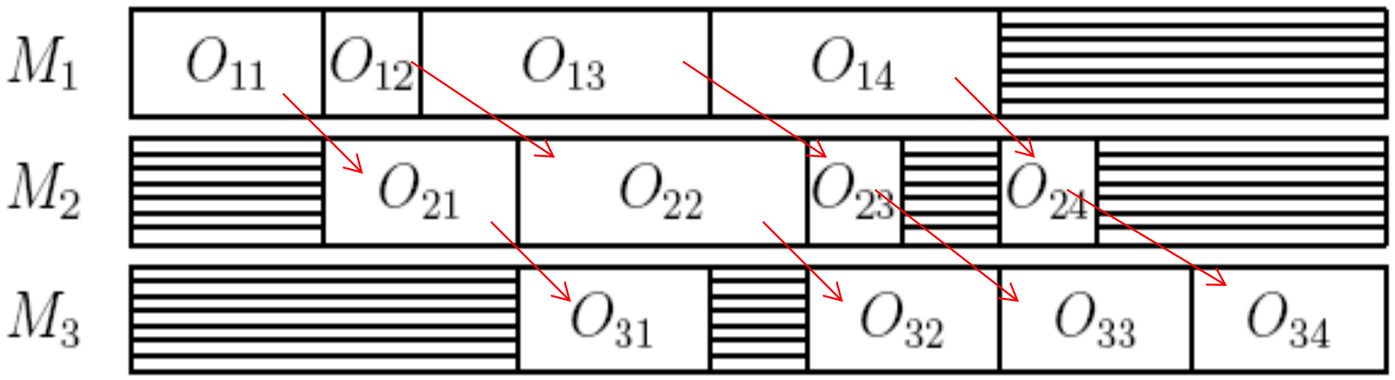
Flow Shop Scheduling

- In the general flow shop model, there are a series of machines numbered 1,2,3…m. Each job has exactly m tasks. The first task of every job is done on machine 1, second task on machine 2 and so on. Every job goes through all m machines in a unidirectional order. However, the processing time each task spends on a machine varies depending on the job that the task belongs to. The precedence constraint in this model requires that for each job, task i-1 on machine i-1 must be completed before the i^{th} task can begin on machine i.

Example

O_{kj}=Task k of job *i*

job	O1i	O2i	O3i
1	2	2	2
2	1	3	2
3	3	1	2
4	3	1	2



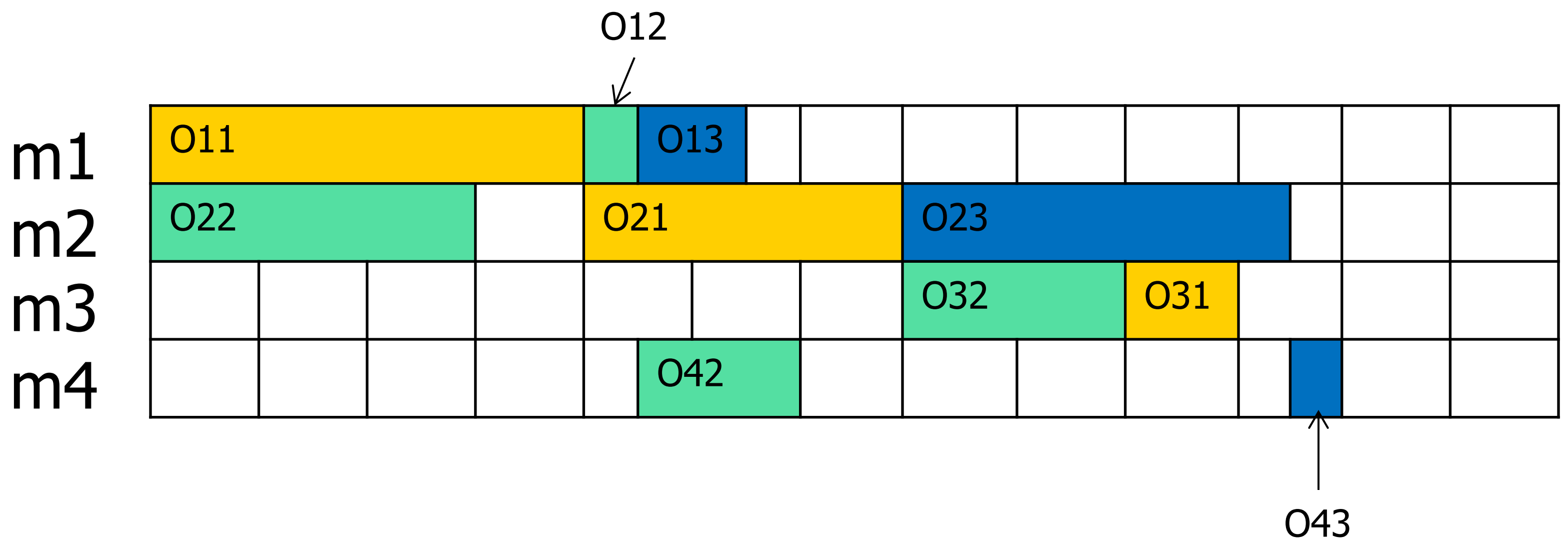
Job shop

- In the general job shop model, there are a set of m machines. Jobs have tasks (or operations) not necessarily m as in the flow shop.
- Each job uses the machines (or subset of them) in a specified sequence
- The flow of the tasks in a job does not have to be unidirectional. Each job may also use a machine more than once.

Example

$O_{\text{machine jobe}}$

Jobs	Machine Sequence	Processing Times
1	1,2,3	O_{11}=8, O_{21}=6, O_{31}=2
2	2,1,4,3	O_{22}=6, O_{12}=1, O_{42}=3, O_{32}=4
3	1,2,4	O_{13}=2, O_{23}=7, O_{43}=1



References

1. N. Metropolis, A. W. Rosenbluth, M. Rosenbluth, A. H. Teller and E. Teller. ”Equation of State Calculations by Fast Computing Machines”. *J. Chem. Phys.* 21, pp. 1087-1092, 1953.
2. S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi. ”Optimization by Simulated Annealing”. *Science* 220, 671-680, 1983.
3. V. Cerny. ”Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm”. *J. Opt. Theory Appl.*, vol. 45, no. 1, pp. 41-51, 1985.
4. L. Ingber. “Adaptive Simulated Annealing: Lessons Learned”. invited paper to a special issue of the Polish Journal *Control and Cybernetics* on “Simulated Annealing Applied to Combinatorial Optimization.”, Vol.25, No.1, pp.33-54 1996.
5. O. Wendt and W. Konig. ”Cooperative Simulated Annealing: How much Cooperation is Enough?”. Research Report 97-19, Institute of Information Systems, GoetheUniversity Frankfurt, 1997.
6. Revees, C., (ed) Modern Heuristic Techniques for Cominatorial Problems. Ch 2 by Kathryn Dowsland, 1993.

References (cont'd)

7. Miki, Hiroyasu, Wako and Yoshida Adaptive Temperature Schedule Determined by Genetic Algorithm for Parallel Simulated Annealing, CEC 2003.
8. David S. Johnson; Cecilia R. Aragon; Lyle A. McGeoch; Catherine Schevon, Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning, *Operations Research*, Vol. 39, No. 3. (May - Jun., 1991), pp. 378-406.