Metaheuristics

Examples of Popular Combinatorial Optimization Problems

Fernando Lobo

Universidade do Algarve

MAXSAT

- The optimization version of the Boolean Satisfiability Problem (SAT) problem
- Formula F is given in Conjunctive Normal Form (CNF): F is the AND of several clauses.
  - Each clause is the OR of several literals
  - A literal is a variable or its negation
- The objective is to maximize the number of satisfied clauses (or alternatively, to minimize the number of unsatisfied clauses)

Weighted MAXSAT

- A variation that assings a positive weight to each clause
- The objective becomes to maximize the total weight of the satisfied clauses (or alternatively, to minimize the total weight of the unsatisfied clauses)
- MAXSAT is a particular case with an equal weight for all clauses

TSP

- We have already seen it in class
- Two common variations:
  - Symmetric TSP: if $w(u, v) = w(v, u)$, for all edges (u, v) in the TSP Graph
  - Asymmetric TSP: otherwise

Vehicle Routing Problem

- A generalization of the TSP
- Objective: find optimal routes traveled by a fleet of vehicles to serve a set of customers
  - TSP is a particular case when there is only 1 vehicle
- Vehicles are located in one or more depots, and may may have a certain maximum capacity
- Application example 1: collecting trash in a city
- Application example 2: deliver products located at a central depot to customers who have placed orders for such products

Knapsack Problem

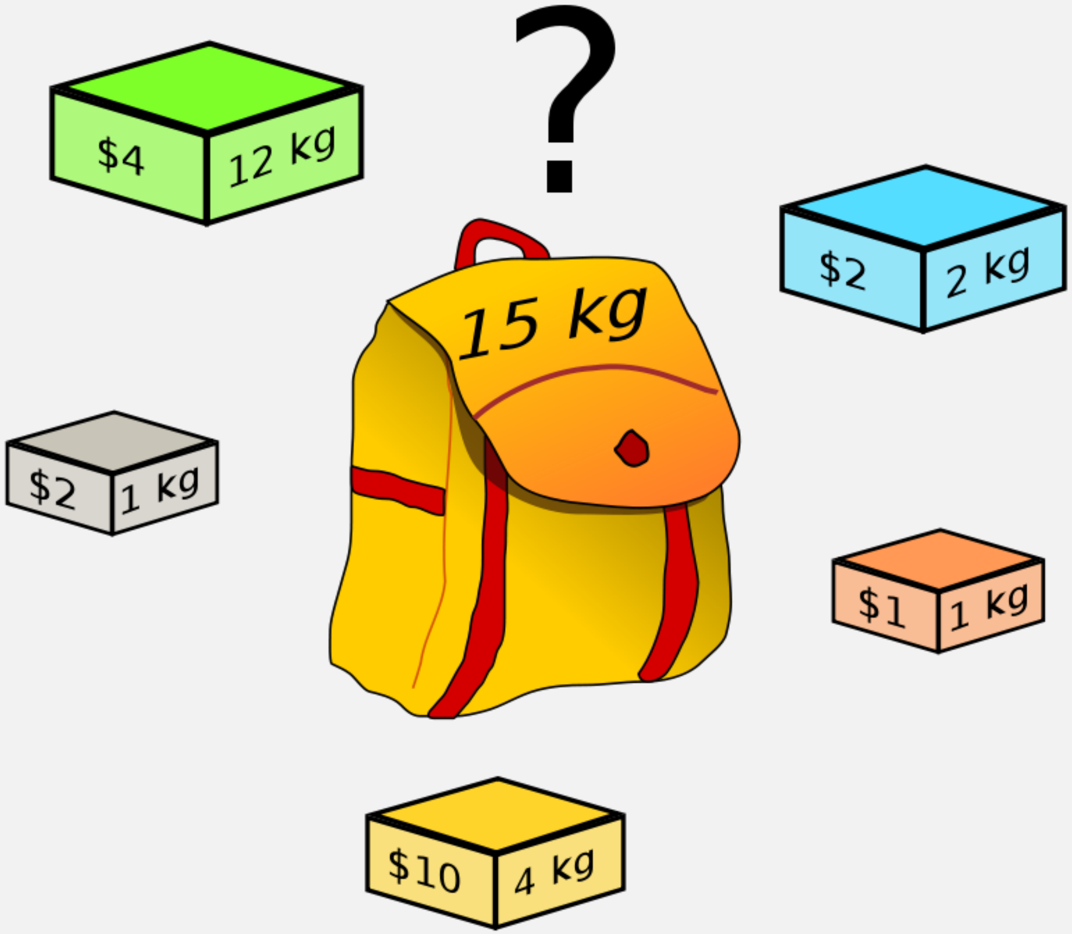


Image taken from `https://en.wikipedia.org/wiki/Knapsack_problem`

Knapsack Problem

- Given a set of n items $x_1 \ldots x_n$, each with a given weight w_i and profit p_i, find a subset of items such that the total profit is maximized, and such that the total weight does not exceed a maximum capacity C
- If weights are integers, the problem can be solved with Dynamic Programming (DP).
- If weights are real numbers, DP is not applicable

Graph Coloring Problem (GCP)

- We are given an undirected graph $G = (V, E)$
- A k-coloring of G is a mapping that assigns a positive integer from $\{1, \ldots, k\}$ to each vertex of G, such that the vertices incidents on every edge are assigned a different integer
  - an integer represents a color
- Optimization version is the Minimum-Cost GCP
  - Objective: find a minimum integer k such that a k-coloring exists

Graph Coloring Problem (GCP)

- Example of a 3-coloring. It's the minimum for this graph

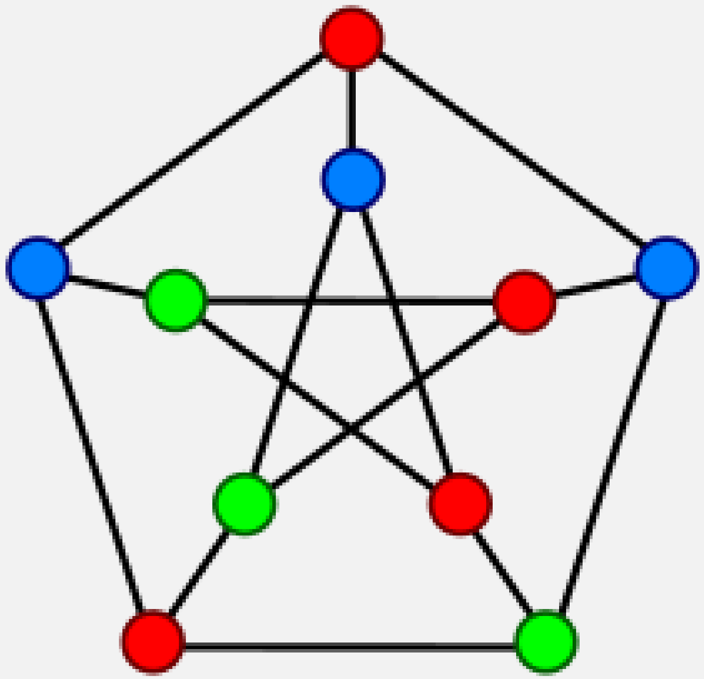


Image taken from https://en.wikipedia.org/wiki/Graph_coloring

Graph Coloring Problem (GCP)

- Example: US states. Vertices correspond to states. There is an edge between 2 states iff they are neighbours

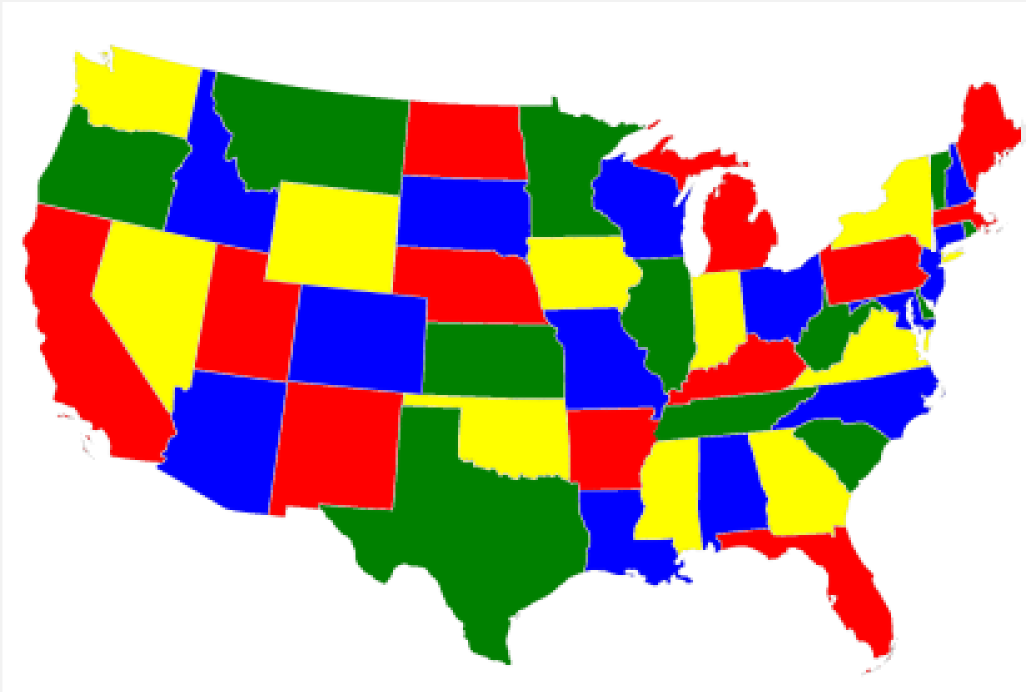


Image taken from `https://docs.ocean.dwavesys.com/en/stable/examples/map_kerberos.html`

Graph Coloring Problem (GCP)

- Application example: Register Allocation in Compilers
  - A compiler needs to allocate temporary values to registers
  - Temporaries t_1 and t_2 can use the same register r as long at any point in the program's execution, at most one of the temporaries is live
    - Compilers build a *register interference graph* (RIG)
    - a node for each temporary
    - an edge between t_1 and t_2 if they are both live at some point in the program
    - two temporaries can be allocated the same register if there is no edge between then in the RIG

Graph Coloring Problem (GCP)

- Application example: Register Allocation in Compilers
  - Colors = registers
  - Let k = number of machine registers
  - If RIG is k-colorable then there is a register assignment that uses no more than k registers

Weighted Graph Coloring Problem (GCP)

- A generalization for weighted undirected graphs
- Objective becomes to minimize the cost of the coloring
- The cost of a coloring is the sum of the weights of all edges whose incident vertices are assigned the same color

Set Covering Problem (SCP)

- Given finite set $A = \{a_1, \ldots, a_m\}$
- and given a collection F of subsets of A that cover A
  - that is: $F = \{A_1, \ldots, A_n\}$
  - and $\bigcup_{i=1}^{n} A_i = A$
- Find a subset of F that also covers A, and whose size is minimal

Set Covering Problem (SCP)

- Example taken from Hoos and Stützle's book
  - $A = \{a, b, c, d, e, f, g\}$
  - $F = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7\}$, with
    - ★ $A_1 = \{a, b, f, g\}$
    - ★ $A_2 = \{a, b, g\}$
    - ★ $A_3 = \{a, b, c\}$
    - ★ $A_4 = \{e, f, g\}$
    - ★ $A_5 = \{f, g\}$
    - ★ $A_6 = \{d, f\}$
    - ★ $A_7 = \{d\}$
  - Optimal solution: $\{A_3, A_4, A_6\}$ or $\{A_3, A_4, A_7\}$

Weighted Set Covering Problem

- A weighted version of the SCP
- Each element of F is assigned a weight
- The objective is to find a set cover with minimum total weight

Airline Crew Scheduling as a Weighted Set Covering Problem

- Given timetable of flight legs
- Assign a crew to each flight leg so that the overall cost of the assignment is as low as possible
  - Each crew has a home base
  - A schedule for a crew is a sequence of flight legs that start and end at its home base
  - Schedules may have additional constraints: rest times, maximum working times, etc.

Airline Crew Scheduling as a Weighted Set Covering Problem

- Flight legs correspond to elements of A
- Each feasible schedule for a crew is represented by a set A_i
- The weight $w(A_i)$ is the cost of that specific crew schedule (include salaries of the crew members, hotel expenses, etc)

Airline Crew Scheduling as a Weighted Set Covering Problem

- To solve the problem:
  - Start by generating a large number of possible schedules for each crew, and compute their costs
    - typically each of these schedules (A_i) has a relatively small number of elements
  - Then obtain a subset cover to minimize the total cost
- The cover guarantees that all flight legs have a crew
- In a real application, we can have thousands of flight legs and hundred thousands of possible crew schedules

Scheduling Problems

- Many combinatorial problems falls into this category
- Involves the allocation of resources (usually called “machines”) and time slots to perform a given set of tasks (or jobs), subject to several constraints and optimization criteria
- Example: Landing and takeoffs at an airport
  - Resources: runways
  - Jobs: incoming and outgoing flights
  - A schedule must assign a runway (and a starting time) to each flight (regardless of being landing or departing)
  - Objective could be to minimize total tardiness

Airport Runway Schedule

- This example is taken from Hoos and Stützle's book (page 421)
- We have 2 runways, and 7 flights (jobs: $J_1, \ldots, J_7$)

Job	J_1	J_2	J_3	J_4	J_5	J_6	J_7
Processing time	10	15	15	10	20	15	10
Due date	15	20	20	30	40	50	50
Release date	0	0	5	10	15	30	35
Weight	1	2	3	2	1	1	3

Airport Runway Schedule

Job	J_1	J_2	J_3	J_4	J_5	J_6	J_7
Processing time	10	15	15	10	20	15	10
Due date	15	20	20	30	40	50	50
Release date	0	0	5	10	15	30	35
Weight	1	2	3	2	1	1	3

Processing time : Time required for airplane to enter runway, and take-off. (During this time, no other plane is allowed to enter the runway)

Due date : Scheduled departure/arrival time

Release date : Time when flight is ready to takeoff (or land)

Weight : Relative importance of flights (ex: connecting flight could have a higher weight)

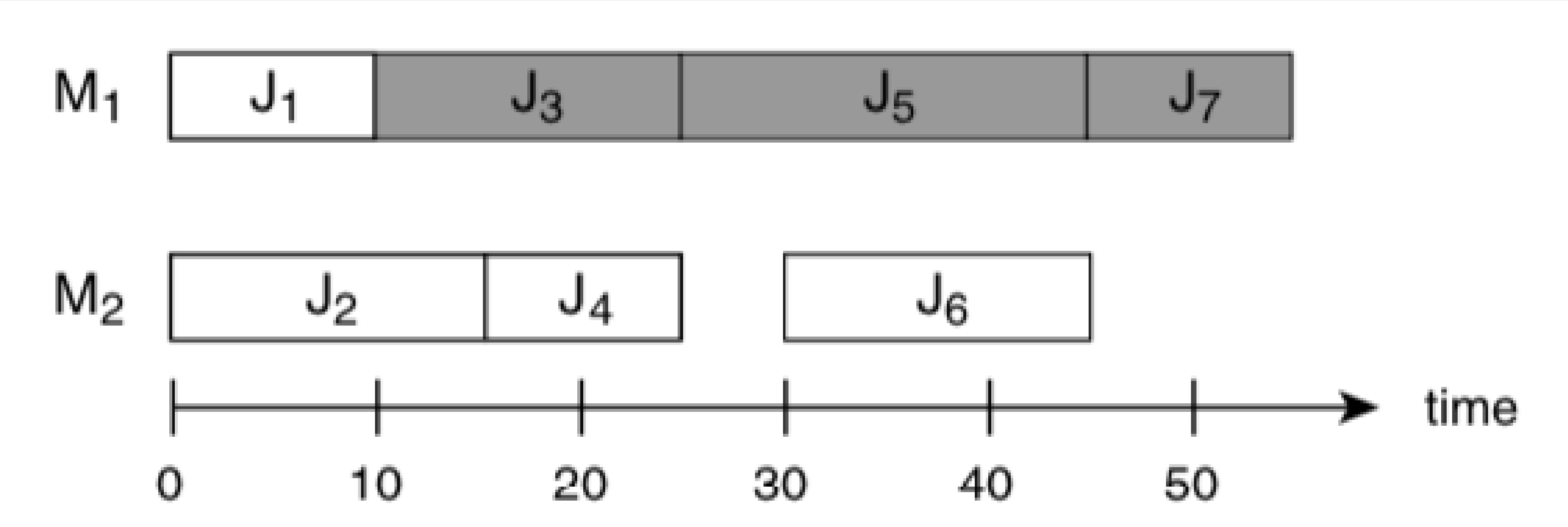
Airport Runway Schedule

Job	J_1	J_2	J_3	J_4	J_5	J_6	J_7
Processing time	10	15	15	10	20	15	10
Due date	15	20	20	30	40	50	50
Release date	0	0	5	10	15	30	35
Weight	1	2	3	2	1	1	3

- A possible schedule could be:
  - Assign the sequence of flights J_1, J_3, J_5, J_7 to runway 1
  - Assign the sequence of flights J_2, J_4, J_6 to runway 2
  - Doing so would make flights J_3, J_5, J_7 go over their due dates. The total cost of this would be $3 \times 5 + 5 + 3 \times 5 = 35$ (see next slide)

Airport Runway Schedule

Job	J_1	J_2	J_3	J_4	J_5	J_6	J_7
Processing time	10	15	15	10	20	15	10
Due date	15	20	20	30	40	50	50
Release date	0	0	5	10	15	30	35
Weight	1	2	3	2	1	1	3



More complex scheduling problems

- The previous example is an single-stage scheduling problem
- It's called single-stage because a job is done atomically
- On multi-stage problems, a job can consist of several operations to be processed on a number of different machines

Permutation Flow Shop Problem (PFSP)

- An example of a multi-stage problem
- Given a set of m machines: $M_1, \ldots, M_m$,
- and a set of n jobs: $J_1, \ldots, J_n$,
  - where each job J_i consists of m operations $o_{1,i}, \ldots, o_{m,i}$, that have to be performed on machines $M_1, \ldots, M_m$, in that order, with processing times $p_{j,i}$ for operation $o_{j,i}$
- Objective is to find a job sequence that minimizes the completion time of the last job
  - this completion is often called the *makespan*

Permutation Flow Shop Problem (PFSP)

- Common assumptions:
  - All jobs are available at time 0 (i.e. release dates = 0)
  - Pre-emption is not allowed: An operation cannot be interrrupted once it starts
  - Each machine can process at most one job at a time, and a job can only be processed by one machine at a time

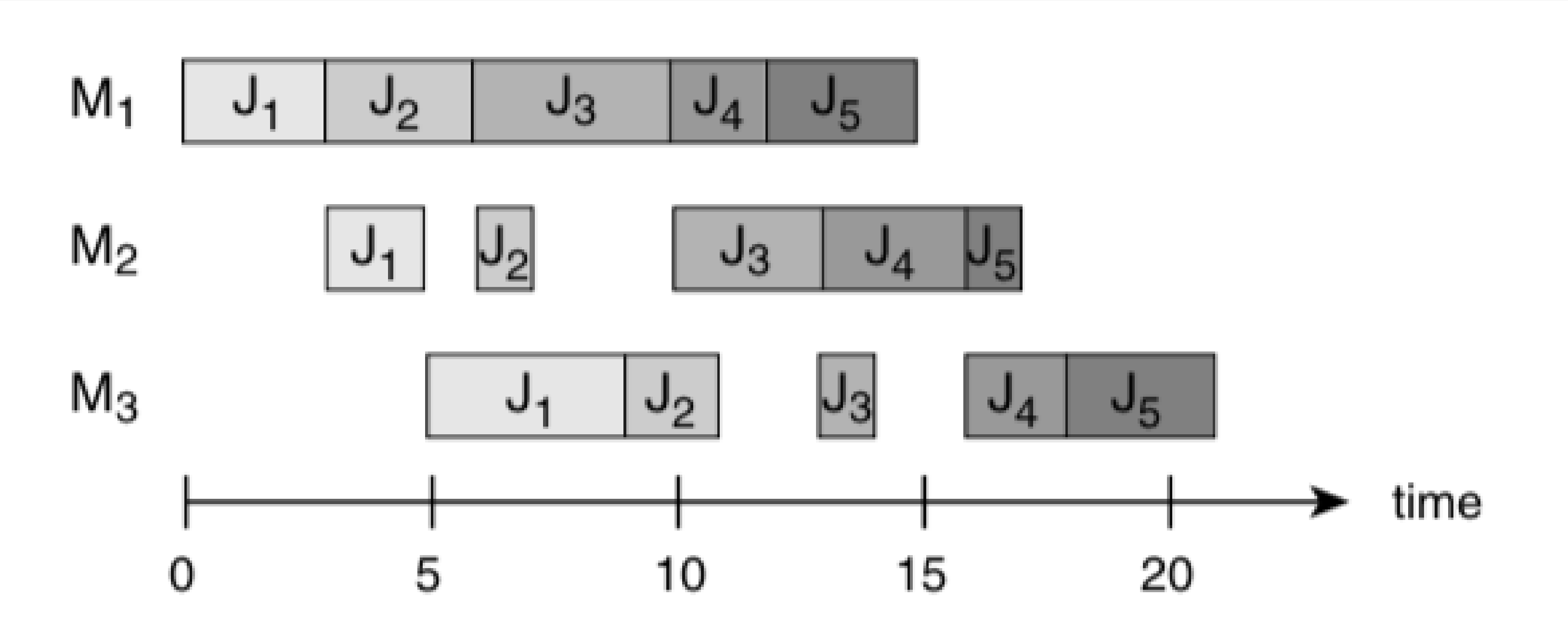
Permutation Flow Shop Problem (PFSP)

- Example taken from Hoos and Stützle's book (page 440, 441)
- We have 3 machines and 5 jobs

Job	J_1	J_2	J_3	J_4	J_5
p_{j1}	3	3	4	2	3
p_{j2}	2	1	3	3	1
p_{j3}	4	2	1	2	3

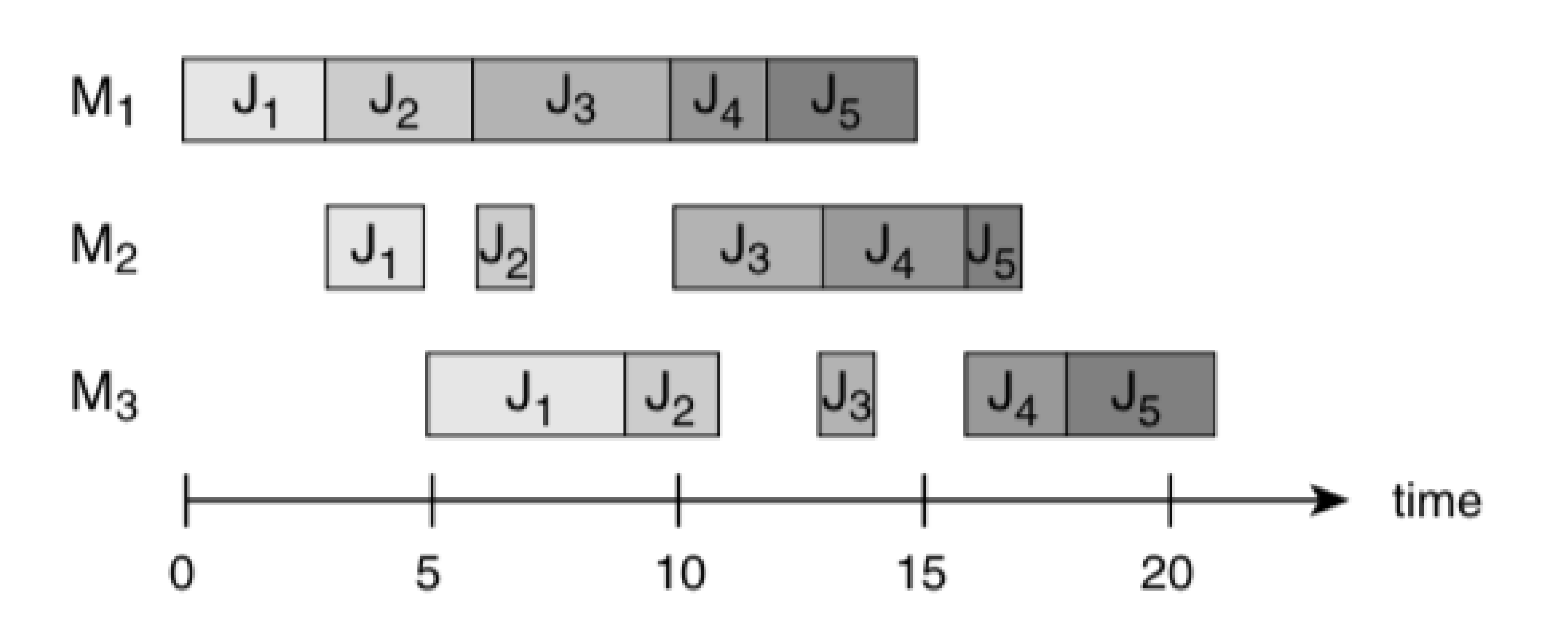
Permutation Flow Shop Problem (PFSP)

- A schedule corresponds to an ordering of the jobs. For example, the schedule J_1, J_2, J_3, J_4, J_5, yields a *makespan* of 21



Permutation Flow Shop Problem (PFSP)

Job	J_1	J_2	J_3	J_4	J_5
p_{j1}	3	3	4	2	3
p_{j2}	2	1	3	3	1
p_{j3}	4	2	1	2	3



Quadratic Assignment Problem (QAP)

- Given n objects and n locations
- and given 2 positive real-valued $n \times n$ matrices A and B, where
  - $A_{i,j}$ is the distance between locations i and j
  - $B_{r,s}$ is the flow between objects r and s
- Objective is to find a one-to-one assignment of objects to locations so that the overall cost is minimized
  - The cost for a pair of locations is defined as the product of the distance between these locations and the flow between the objects assigned to these locations
  - The overall cost is the sum of the individual costs for all pairs of locations

Quadratic Assignment Problem (QAP)

- A one-to-one mapping can be represented as a permutation ψ of the integers $1 \dots n$
- We want to minimize:
  - $f(\psi) = \sum_{i=1}^{n} \sum_{j=1}^{n} B_{i,j} A_{\psi(i),\psi(j)}$

Assigning people to offices, as a QAP

- Objects correspond to people
- Locations correspond to offices
- Flow between two persons could be the "affinity" between them.
  - $B_{i,j}$ is the affinity between person i and person j. The higher the value, the higher the affinity is
- Distance between two offices measures in distance or time
  - $A_{k,\ell}$ could be the time needed to go from office k to office ℓ
- Optimization objective is to minimize the total amount of walking distance for the people

Keyboard layout, as a QAP

- Objects correspond to letters of the alphabet
- Locations correspond to keys of the keyboard
- Flow between two letters could be the empirical frequency of the corresponding combination
  - $B_{w,e}$ is the empirical frequency of letter 'w' being followed by letter 'e'
- Distance between two keys could be the empirically measured time required for pressing them in sequence
  - $A_{i,j}$ is time needed to press key i followed by key j
- Optimization objective is to minimize total time of typing a typical text

Other problems

- Search for NP-complete or NP-hard problems on the Internet
- Wikipedia list for NP-complete problems:
  - `https://en.wikipedia.org/wiki/List_of_NP-complete_problems`