Trajectory-based Optimization:

Tabu Search - II

Lecture 6 – Tuesday May 27, 2014

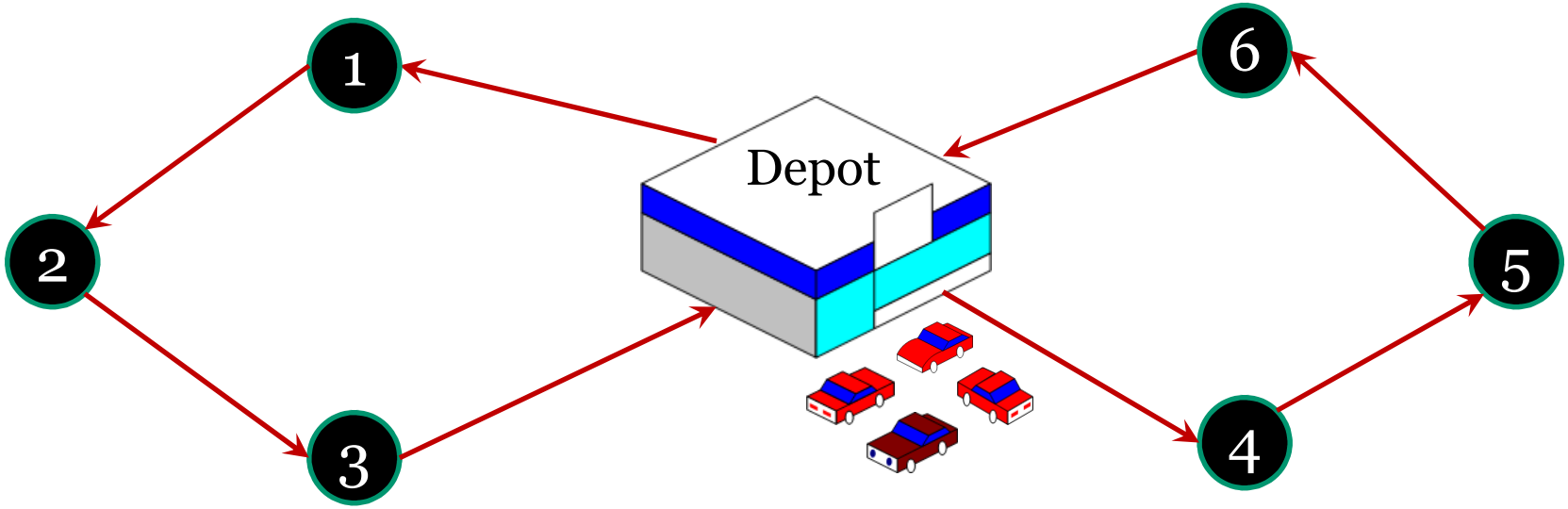
Outline

- Tabu Search
- Solving Permutation Optimization
- Solving Graph Problems
- Solving Constraint Satisfaction Problems
- **Vehicle Routing Problem (VRP)**
- Assignment Problem
- Knapsack Problem
- Adaptation
- Cooperation
- Summary

Vehicle Routing Problem

The **Vehicle Routing Problem (VRP)** is defined by having **m vehicles** at the depot that need to service **c customers**.

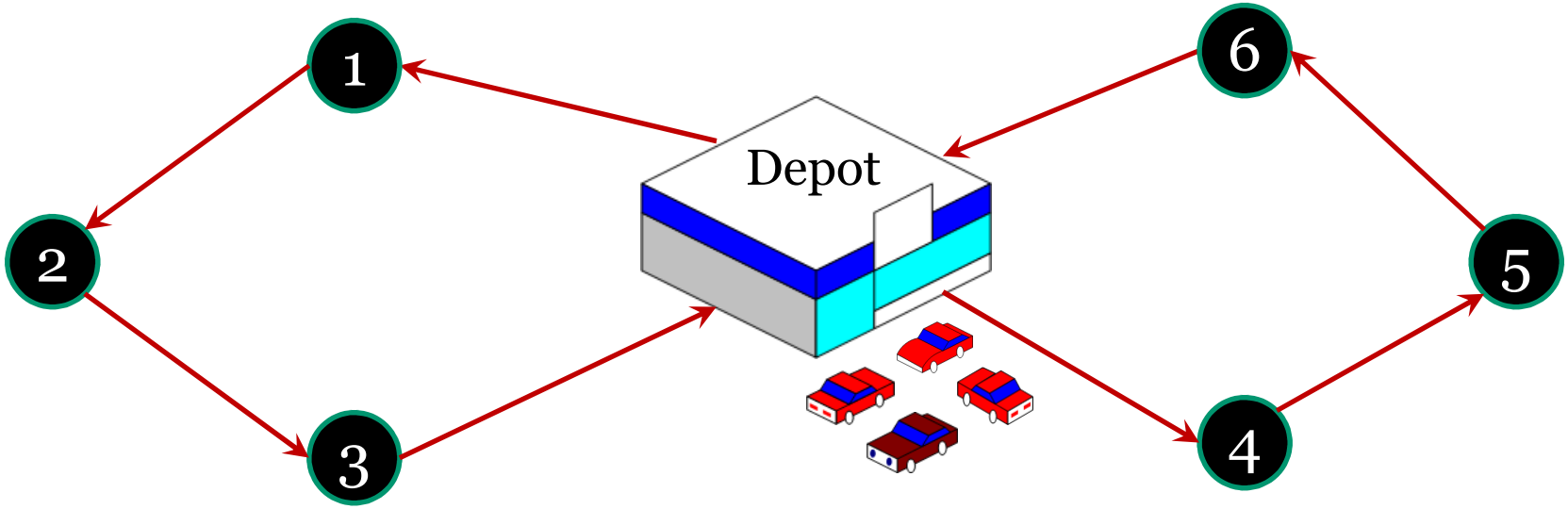
The travel distances between every two cities are defined by a matrix D. The travel distances from the depot to all the cities are given by a vector Depot.



Vehicle Routing Problem

The VRP consists of determining the routes to be taken by a set of **m vehicles** satisfying the following conditions:

- Starting and ending at the depot,
- Having minimum travel cost,
- Each customer is visited exactly once by exactly one vehicle.



Vehicle Routing Problem

- **Solution Representation**

Tabu search can be used to solve this problem by having the following vector solution representation:

$$Sol = [d, R_1, d, R_2, ..., d, R_m]$$

where $\boldsymbol{R_i}$ denotes the list of customers serviced by vehicle $\boldsymbol{i}$ and these lists are separated by d denoting the depot.

Vehicle Routing Problem

- **Neighborhood**

The neighborhood can be defined by:

- ◇ **Swapping** two randomly selected cities in any two randomly selected routes,
- ◇ **Removing** a randomly selected city from one randomly selected route and **inserting** it into a randomly selected position in a different randomly selected route,
- ◇ There are **no swaps** between two cities in the same route.

Vehicle Routing Problem

- **Tabu Memory Structure**
  - ◇ The tabu memory structure should record moves to prevent reversing them.
  - ◇ The moves in this problem are 2 types: swapping cities between routes and inserting/removing cities from routes.
  - ◇ In order to prevent reversing the move, we need to record the city and the routes involved.
  - ◇ A matrix of cities & routes will be a good structure.

Cities	Routes			
	R_1	R_2	...	R_m
1				
2				
3				
4				
...				
n				

Vehicle Routing Problem

- **Tabu Memory Structure**
  - ◇ If a **swap** happens between Cities i & j from Routes R_p & R_q respectively, then we enter a tabu entry at (i,p) and at (j,q) to avoid these moves. The entry will be tabu length *h* (iterations).

Cities	Routes						
	R_1	R_2	…	R_p	R_q	…	R_m
1							
2							
….							
i				h			
j					h		
…							
n							

Vehicle Routing Problem

- **Tabu Memory Structure**
  - ◇ If a city ***i*** is **removed from route** R_p and inserted in R_q, we can **prevent inserting it back** by entering *h* in (i,p) (same as in swap).

Cities	Routes						
	R_1	R_2	...	R_p	R_q	...	R_m
1							
2							
....							
i				h			
j							
...							
n							

Vehicle Routing Problem

- **Tabu Memory Structure**
  - ◇ To **prevent removing it** from R_q we can enter –h in (i,q) –ve sign indicate tabu on **deletion operation**.

Cities	Routes						
	R_1	R_2	…	R_p	R_q	…	R_m
1							
2							
….							
i				h			
j					-h		
…							
n							

Vehicle Routing Problem

- **Symmetric Problem**

Given:

The following symmetric problem: $D = \begin{bmatrix} 0 & 12 & 20 & 25 & 30 & 20 \\ & 0 & 10 & 11 & 22 & 30 \\ & & 0 & 2 & 11 & 25 \\ & & & 0 & 10 & 20 \\ & & & & 0 & 12 \\ & & & & & 0 \end{bmatrix}$, $Depot = \begin{bmatrix} 10 \\ 20 \\ 25 \\ 25 \\ 20 \\ 10 \end{bmatrix}$

Assume that when a city gets removed from one route inserted into the other, it only gets inserted at the beginning of the route.

The cost of a route R_i is given by (n is the number of cities in a route):

$$Cost(R_i) = Depot(R_{i1}) + \sum_{j=1}^{n-1} D(R_{ij}, R_{ij+1}) + Depot(R_{in})$$

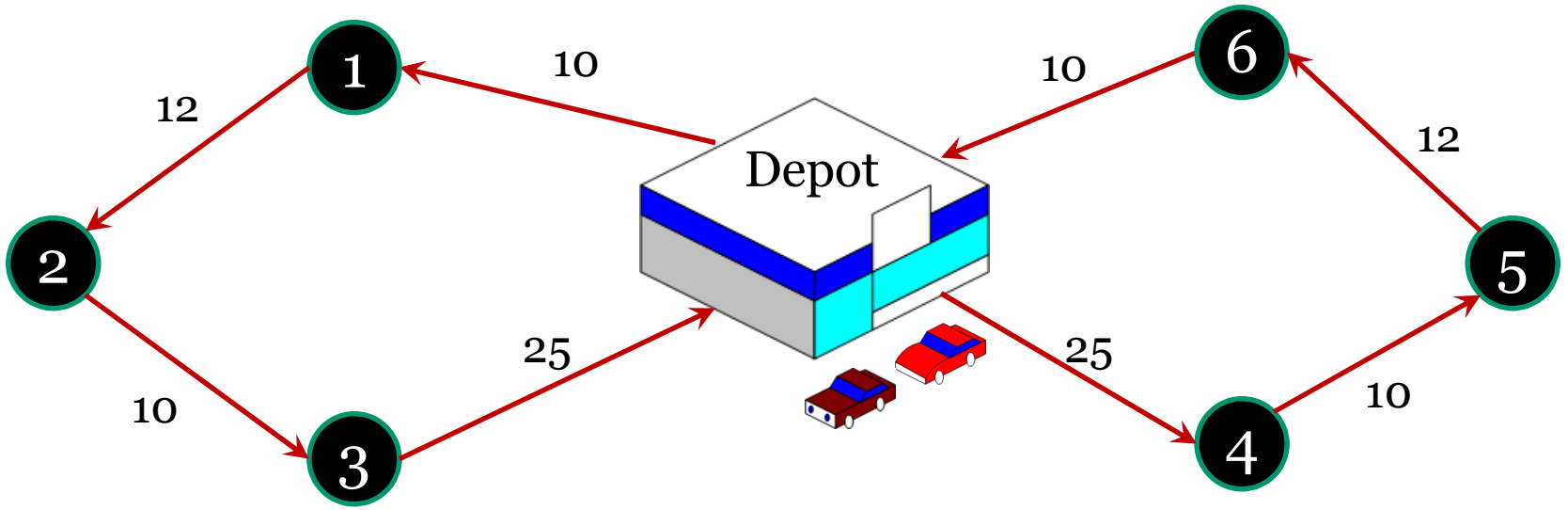
and the total cost of a solution (for **two vehicles**) is given by:

$$Cost(Sol) = Cost(R_1) + Cost(R_2)$$

Vehicle Routing Problem

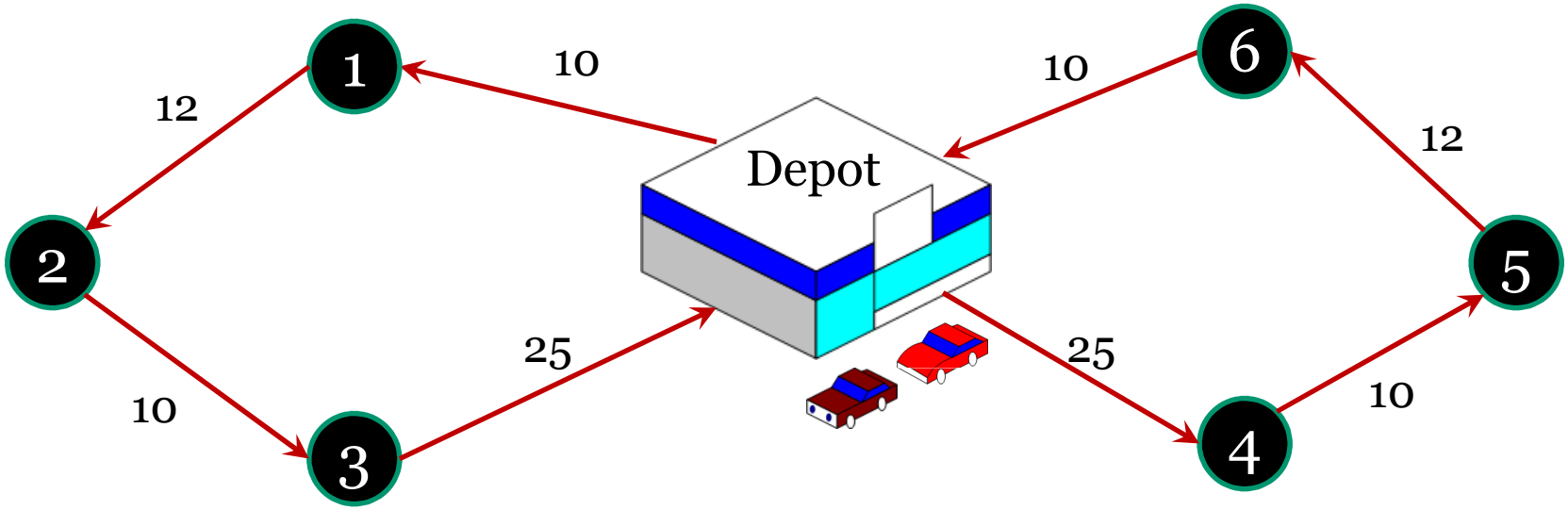
- **2 Vehicles Problem**

$$D = \begin{bmatrix} 0 & 12 & 20 & 25 & 30 & 20 \\ & 0 & 10 & 11 & 22 & 30 \\ & & 0 & 2 & 11 & 25 \\ & & & 0 & 10 & 20 \\ & & & & 0 & 12 \\ & & & & & 0 \end{bmatrix}, \quad Depot = \begin{bmatrix} 10 \\ 20 \\ 25 \\ 25 \\ 20 \\ 10 \end{bmatrix}$$



Vehicle Routing Problem

- **2 Vehicles Problem**



Problem has 2 vehicles so we have 2 routes.

Initial Solution: $Sol = [d, 1, 2, 3, d, 4, 5, 6]$

Cost=(10+12+10+25)+(25+10+12+10)=114

Vehicle Routing Problem

- **2 Vehicles Problem**

$$\text{Initial Sol} = [d,1,2,3,d,4,5,6]$$

Cost=114

$$D = \begin{bmatrix} 0 & 12 & 20 & 25 & 30 & 20 \\ & 0 & 10 & 11 & 22 & 30 \\ & & 0 & 2 & 11 & 25 \\ & & & 0 & 10 & 20 \\ & & & & 0 & 12 \\ & & & & & 0 \end{bmatrix}, \quad Depot = \begin{bmatrix} 10 \\ 20 \\ 25 \\ 25 \\ 20 \\ 10 \end{bmatrix}$$

Neighbors (using swaps):	**Cost**
[d , 4 , 2 , 3 , d , 1 , 5 , 6]	(25+11+10+25)+(10+30+12+10)=133
[d , 5 , 2 , 3 , d , 4 , 1 , 6]	(20+22+10+25)+(25+25+20+10)=158
[d , 6 , 2 , 3 , d , 4 , 5 , 1]	=150
[d , 1 , 4 , 3 , d , 2 , 5 , 6]	=126
[d , 1 , 5 , 3 , d , 4 , 2 , 6]	=154
[d , 1 , 6 , 3 , d , 4 , 5 , 2]	=157

Vehicle Routing Problem

- **2 Vehicles Problem** Initial Sol $= [d, 1, 2, 3, d, 4, 5, 6]$ Cost=114

Neighbors (using swaps):	**Cost**
[d , 4 , 2 , 3 , d , 1 , 5 , 6]	(25+11+10+25)+(10+30+12+10)=133
[d , 5 , 2 , 3 , d , 4 , 1 , 6]	(20+22+10+25)+(25+25+20+10)=158
[d , 6 , 2 , 3 , d , 4 , 5 , 1]	=150
[d , 1 , 4 , 3 , d , 2 , 5 , 6]	=126
[d , 1 , 5 , 3 , d , 4 , 2 , 6]	=154
[d , 1 , 6 , 3 , d , 4 , 5 , 2]	=157
[d , 1 , 2 , 4 , d , 3 , 5 , 6]	=116
[d , 1 , 2 , 5 , d , 4 , 3 , 6]	=124
[d , 1 , 2 , 6 , d , 4 , 5 , 3]	=123

Vehicle Routing Problem

- **2 Vehicles Problem** Initial Sol $= [d,1,2,3,d,4,5,6]$ Cost=114

  Using insertion/deletion

[d , 2 , 3 , d , 1 , 4 , 5 , 6] = 122

[d , 1, 3 , d , 2 , 4 , 5 , 6] = 118

[d , 1 , 2 , d , 3 , 4 , 5 , 6] = 101

[d , 4 , 1 , 2 , 3 , d , 5 , 6] = 139

[d , 5 , 1 , 2 , 3 , d , 4 , 6] = 152

[d , 6 , 1 , 2 , 3 , d , 4 , 5] = 132

Best cost among all these is **101**

[d , 1 , 2 , d , 3 , 4 , 5 , 6]

Vehicle Routing Problem

- **2 Vehicles Problem**

Initial Sol $= [d, 1, 2, 3, d, 4, 5, 6]$

Best solution (deletion/insertion) so far is:

[d , 1 , 2 , d , 3 , 4 , 5 , 6]

The move is delete 3 from R_1 and add it to R_2

This means that next time there is a move to insert 3 in R_1 it is tabued also deletion of 3 from R_2 is tabued.

	R_1	R_2
1		
2		
3	h	-h
4		
5		
6		

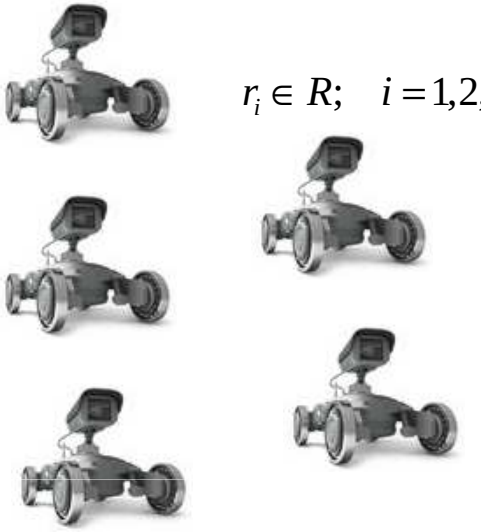
Outline

- Tabu Search
- Solving Permutation Optimization
- Solving Graph Problems
- Solving Constraint Satisfaction Problems
- Vehicle Routing Problem (VRP)
- **Assignment Problem**
- Knapsack Problem
- Adaptation
- Cooperation
- Summary

Assignment Problem

- Assignment problems deal with the question **how to assign n items (e.g. jobs) to n machines (or workers) in the best possible way**.
- They consist of two components:
  - ◇ The **assignment** as underlying combinatorial structure and
  - ◇ An objective function modeling the ”**best way**”.

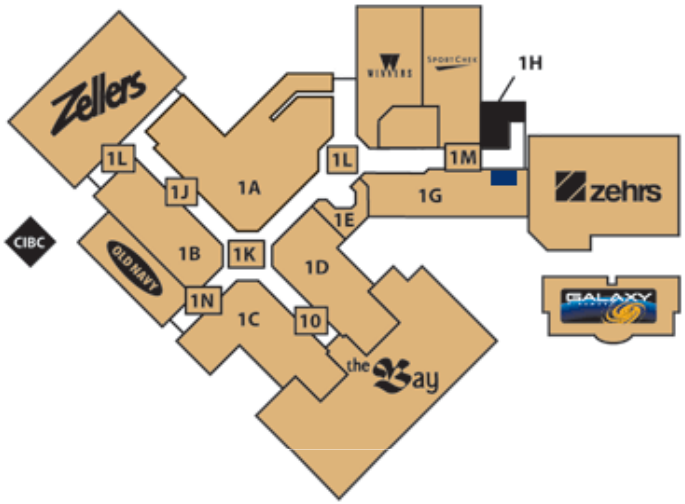
Assignment Problem



$r_i \in R; \quad i = 1,2,...,n$

$\pi : T \rightarrow R$

$t_j \in T; \quad j = 1,2,...,m$



A set of n mobile Sensors: R

A set of m surveillance tasks: T

A set of n mobile sensors to carry out m surveillance tasks. If the sensor i does task j, it costs c_{ij} units. Find as assignment π which minimizes $\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}\pi_{ij}$

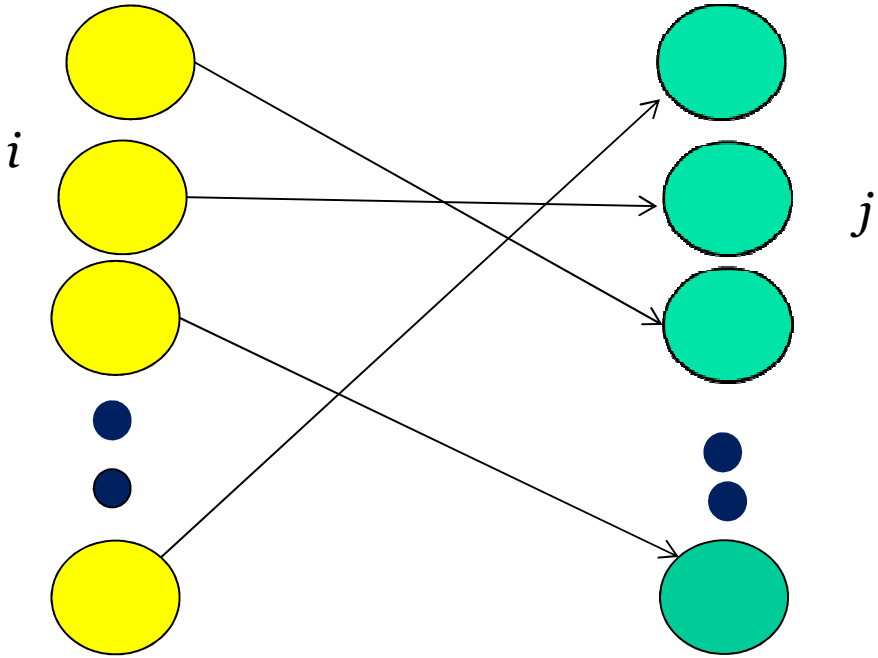
Assignment Problem

- **Linear Assignment Problem: Mathematical Modelling**

bijective mapping

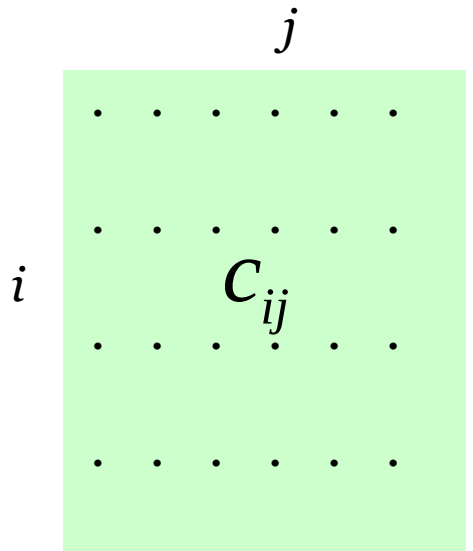
Set of Objects

(persons, facility, etc)



Set of sites

Jobs, locations, etc.



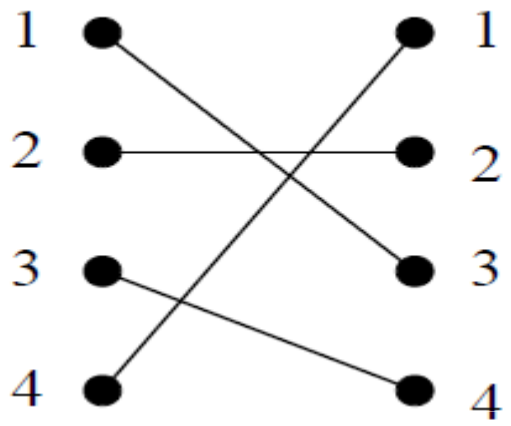
Bipartite graph

bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint sets

Assignment Problem

- **Linear Assignment Problem: Mathematical Modelling**

Assume that:



$G_\Phi = (V,W;E)$: **bipartite graph**, where the vertex sets V and W have n vertices, i.e., $|V| = |W| = n$, and there is an edge $(i, j) \in E$ iff $= \Phi(i)$,

Φ: every **permutation** of the set $N = \{1, \ldots, n\}$ $\phi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 1 \end{pmatrix}$

X_Φ : **adjacency matrix** of a bipartite graph G_Φ

$X_\Phi = (x_{ij})$ with $x_{ij} = 1$ for $j = \Phi(i)$ and $x_{ij} = 0$ for $j \neq \Phi(i)$.

$$X_\phi = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Assignment Problem

- **Linear Assignment Problem: Mathematical Modelling**

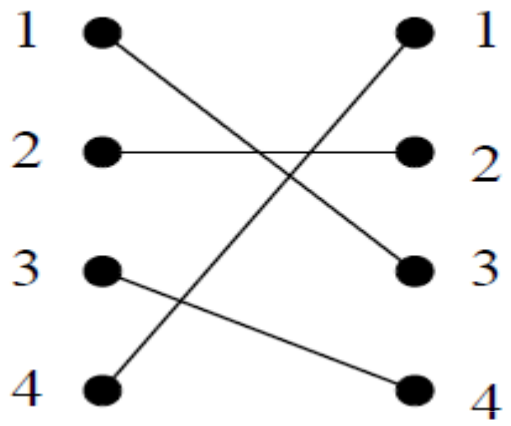
The set of all assignments of n items will be denoted by S_n and has $n!$ elements.

We can describe this set by the following equations called **assignment constraints**:

$$\sum_{i=1}^{n} x_{ij} = 1 \text{ for all } j = 1,..n,$$

$$\sum_{j=1}^{n} x_{ij} = 1 \text{ for all } i = 1,..n$$

$$x_{ij} \in \{0,1\} \text{ for all } i, j = 1,..n$$



$$\phi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 1 \end{pmatrix}$$

$$X_{\phi} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Assignment Problem

- **Linear Assignment Problem: Mathematical Modelling**

Mathematically, the problem can be formulated as follows:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

over all permutations of {1, 2, . . . , n},

Subject to:

$$\sum_{i=1}^{n} x_{ij} = 1 \text{ for all } j = 1,..n,$$

$$\sum_{j=1}^{n} x_{ij} = 1 \text{ for all } i = 1,..n$$

$$x_{ij} \in \{0,1\} \text{ for all } i, j = 1,..n$$

Assignment Problem

- **Linear Assignment Problem: Example**

  Workers: A, B, C, D **Jobs:** P, Q, R, S.

  Cost matrix:

	Job P	Job Q	Job R	Job S
Worker A	5	6	7	6
Worker B	4	3	2	3
Worker C	2	3	5	2
Worker D	5	5	2	8

  - ◇ **Given:** each worker need perform only one job and each job need be assigned to only one worker.
  - ◇ **Question:** how to assign the jobs to the workers to minimize the cost?

Assignment Problem

- **Linear Assignment Problem: Example**

Consider m workers to whom n jobs are assigned.

The cost of assigning worker i to job j is c_{ij}.

Let $x_{ij} = \begin{cases} 1 & \text{if job j is assigned to worker i} \\ 0 & \text{if job j is not assigned to worker i} \end{cases}$

Objective function: $\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$

Assignment Constraints: $\sum_{i=1}^{n} x_{ij} = 1 \text{ for all } j = 1,..n,$

$$\sum_{j=1}^{n} x_{ij} = 1 \text{ for all } i = 1,..n$$

$$x_{ij} \in \{0,1\} \text{ for all } i, j = 1,..n$$

Assignment Problem

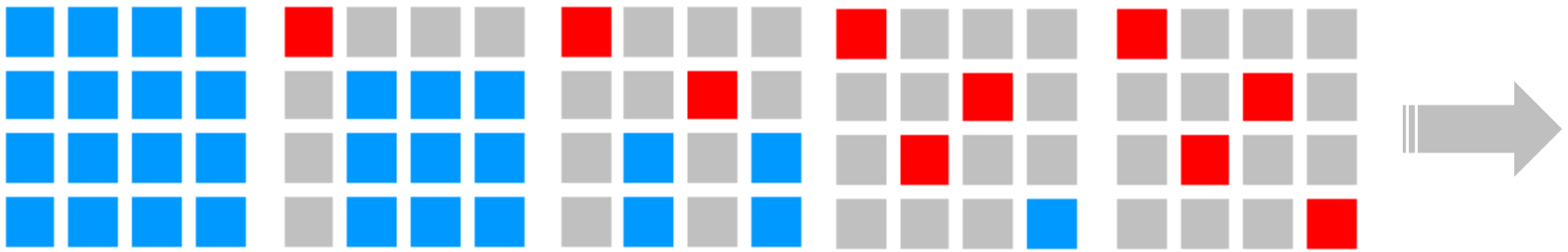
- **Linear Assignment Problem: Example**

  Brute-force method

  ◇ Enumerate all candidates sets

  ◇ n! possible assignment sets

  ◇ Exponential runtime complexity



4! = 24 10! = 3,628,800 100! ≈9.3 ×10^{157}

Assignment Problem

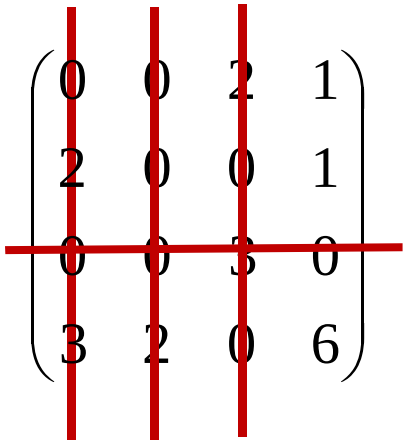
- **Linear Assignment Problem: Example**

  Hungarian method

  For each row, subtract the minimum number in that row from all numbers in that row; do the same for each column.

$$\begin{pmatrix} 5 & 6 & 7 & 6 \\ 4 & 3 & 2 & 3 \\ 2 & 3 & 5 & 2 \\ 5 & 5 & 2 & 8 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1 & 2 & 1 \\ 2 & 1 & 0 & 1 \\ 0 & 1 & 3 & 0 \\ 3 & 3 & 0 & 6 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 0 & 2 & 1 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 3 & 0 \\ 3 & 2 & 0 & 6 \end{pmatrix}$$

  There exists an optimal solution, when # of assignments = minimum # of lines required to cover all 0s.

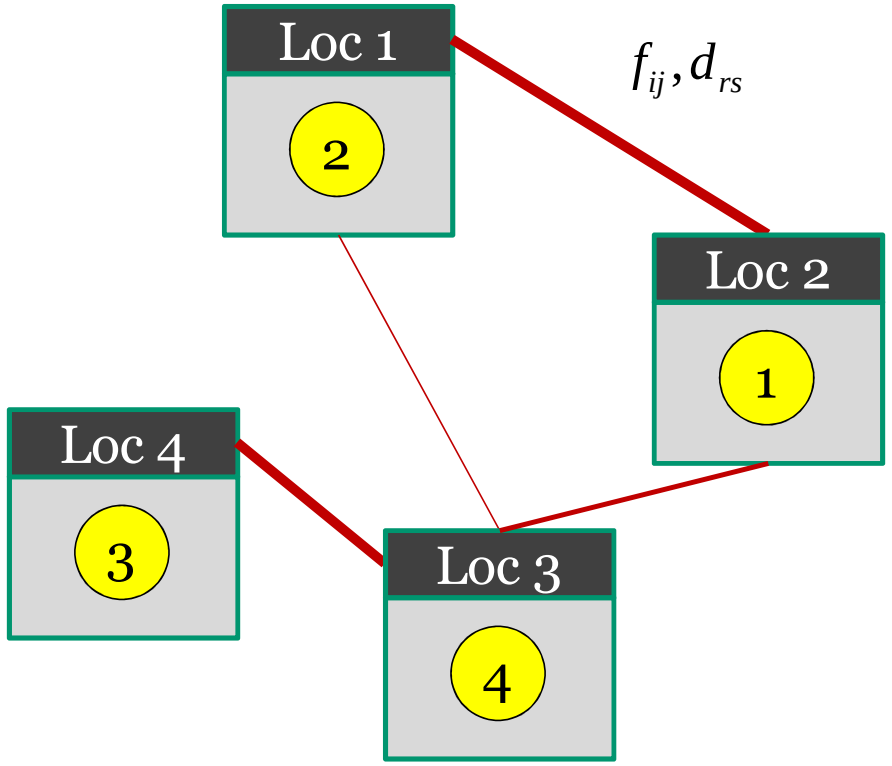


[For reading]

For more info: http://www.math.harvard.edu/archive/20_spring_05/handouts/assignment_overheads.pdf

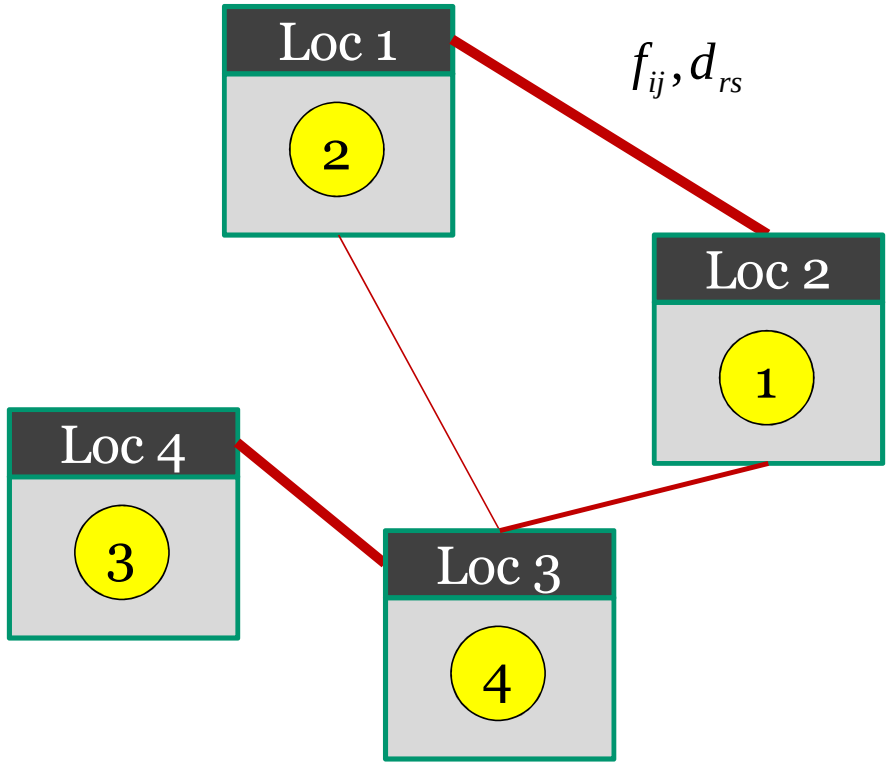
Assignment Problem

- **Quadratic Assignment Problem (QAP)**
  - ◇ Given n objects and the flows f_{ij} between the object i and the object j $(i, j = 1 \ldots n)$, and
  - ◇ Given n sites with distance d_{rs} between the sites r and s $(r, s = 1 \ldots n)$,



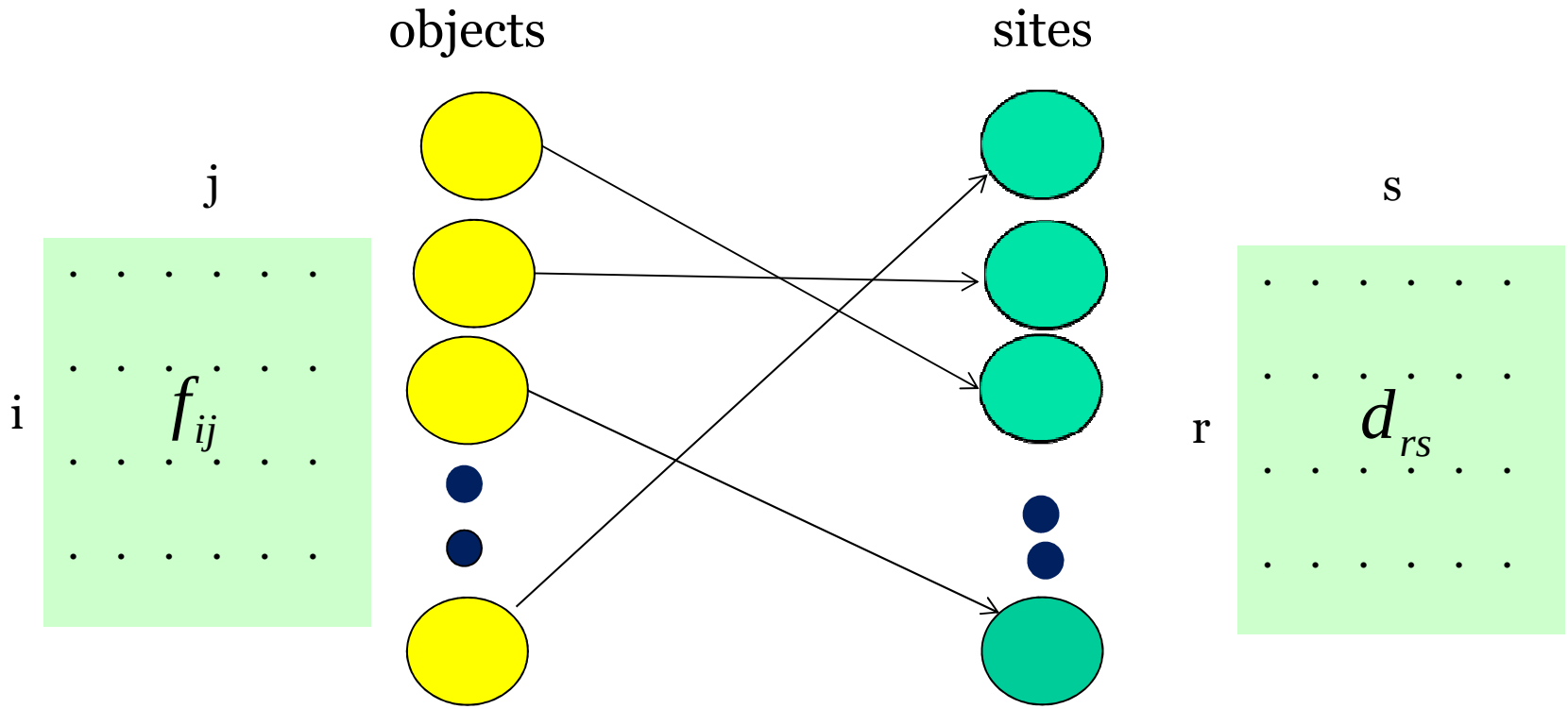
Assignment Problem

- **Quadratic Assignment Problem (QAP)**
  - The problem deals with placing the n objects on the n sites so as to minimize the sum of the products, **flows × distances**.
  - Mathematically, this is equivalent to find a permutation p, whose i^{th} component denotes p_i the object assigned to site i, which minimizes $\sum_{i=1}^{n}\sum_{j=1}^{n} f_{p_i p_j} d_{ij}$



Assignment Problem

- **Quadratic Assignment Problem (QAP)**



Bipartite graph

[1]

Assignment Problem

- **Quadratic Assignment Problem (QAP)**

  Let us consider the small **5×5 instance of quadratic assignment problem**, known in literature as **NUG5**, with flows matrix *F* and distances matrix *D*:

$$F = \begin{pmatrix} 0 & 5 & 2 & 4 & 1 \\ 5 & 0 & 3 & 0 & 2 \\ 2 & 3 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 5 \\ 1 & 2 & 0 & 5 & 0 \end{pmatrix} \qquad D = \begin{pmatrix} 0 & 1 & 1 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 \\ 1 & 2 & 0 & 1 & 2 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 2 & 3 & 1 & 0 \end{pmatrix}$$

[1]

Assignment Problem

- **Quadratic Assignment Problem (QAP): Iteration-0**

Initial solution: $P = (2 \quad 4 \quad 1 \quad 5 \quad 3)$

which means object 2 is assigned to site 1, object 4 to site 2,... etc.

$$\text{cost} = \sum_{i=1}^{n}\sum_{j=1}^{n} f_{p_i p_j} d_{ij}$$

To calculate the cost, take the permutation and apply it to the matrix F (rows and columns) then multiply corresponding rows of permuted F and D

$$PF = (2 \quad 4 \quad 1 \quad 5 \quad 3)\begin{pmatrix} 0 & 5 & 2 & 4 & 1 \\ 5 & 0 & 3 & 0 & 2 \\ 2 & 3 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 5 \\ 1 & 2 & 0 & 5 & 0 \end{pmatrix} = \begin{pmatrix} 5 & 0 & 3 & 0 & 2 \\ 4 & 0 & 0 & 0 & 5 \\ 0 & 5 & 2 & 4 & 1 \\ 1 & 2 & 0 & 5 & 0 \\ 2 & 3 & 0 & 0 & 0 \end{pmatrix}$$

[1]

Assignment Problem

- **Quadratic Assignment Problem (QAP): Iteration-0**

Initial solution: $P = (2 \quad 4 \quad 1 \quad 5 \quad 3)$

$$PFP = \begin{pmatrix} 5 & 0 & 3 & 0 & 2 \\ 4 & 0 & 0 & 0 & 5 \\ 0 & 5 & 2 & 4 & 1 \\ 1 & 2 & 0 & 5 & 0 \\ 2 & 3 & 0 & 0 & 0 \end{pmatrix} (2 \quad 4 \quad 1 \quad 5 \quad 3) = \begin{pmatrix} 0 & 0 & 5 & 2 & 3 \\ 0 & 0 & 4 & 5 & 0 \\ 5 & 4 & 0 & 1 & 2 \\ 2 & 5 & 1 & 0 & 0 \\ 3 & 0 & 2 & 0 & 0 \end{pmatrix}$$

$$PFP * D = \begin{pmatrix} 0 & 0 & 5 & 2 & 3 \\ 0 & 0 & 4 & 5 & 0 \\ 5 & 4 & 0 & 1 & 2 \\ 2 & 5 & 1 & 0 & 0 \\ 3 & 0 & 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 \\ 1 & 2 & 0 & 1 & 2 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 2 & 3 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 5+4+9=18 \\ 8+5=13 \\ 5+8+1+4=18 \\ 4+5+1=10 \\ 9+4=13 \end{pmatrix}$$

Cost=18+13+18+10+13=72

Assignment Problem

- **Quadratic Assignment Problem (QAP): Iteration-0**

  Initial solution: $P = (2 \quad 4 \quad 1 \quad 5 \quad 3)$

  The tabu search can be started by initializing a tabu matrix

  sites

$$T = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$ elements

Assignment Problem

- **Quadratic Assignment Problem (QAP): Iteration-0**

  Initial solution: $P = (2 \quad 4 \quad 1 \quad 5 \quad 3)$

  Neighboring solution: calculate value $\Delta(\mathbf{p}, (i, j))$ is calculated for each transposition (i, j):

Move	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(2, 3)	(2, 4)	(2, 5)	(3, 4)	(3, 5)	(4, 5)
Cost	-12	2	-12	2	0	-10	-12	4	8	6

It can be realized that three moves produce a maximum profit of 12: to exchange the objects on the sites (1, 2), (1, 4) and (2, 5).

Assignment Problem

- **Quadratic Assignment Problem (QAP): Iteration-1**

Current solution: $P = (2 \quad 4 \quad 1 \quad 5 \quad 3)$ Cost=72

One can assume that it is the first of these moves, (1, 2), which is retained.

New solution: $P = (1 \quad 4 \quad 2 \quad 5 \quad 3)$ Cost=60

If it is prohibited during $t = 6$ iterations to put element 2 on site 1 and element 1 on site 3 simultaneously, the following matrix is obtained:

$$T = \begin{pmatrix} 0 & 0 & 6 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Assignment Problem

- **Quadratic Assignment Problem (QAP): Iteration-2**

Current solution: $P = (1 \quad 4 \quad 2 \quad 5 \quad 3)$

Neighboring solution:

Move	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(2, 3)	(2, 4)	(2, 5)	(3, 4)	(3, 5)	(4, 5)
Cost	14	12	-8	10	0	10	8	12	12	6
	T									

For this iteration, it should be noticed that the reverse of the preceding move is now prohibited.

The authorized move **(1, 4)**, giving minimum cost, is retained, for a profit of 8.

tabu duration of the reverse move

is t = 6, the matrix T becomes:

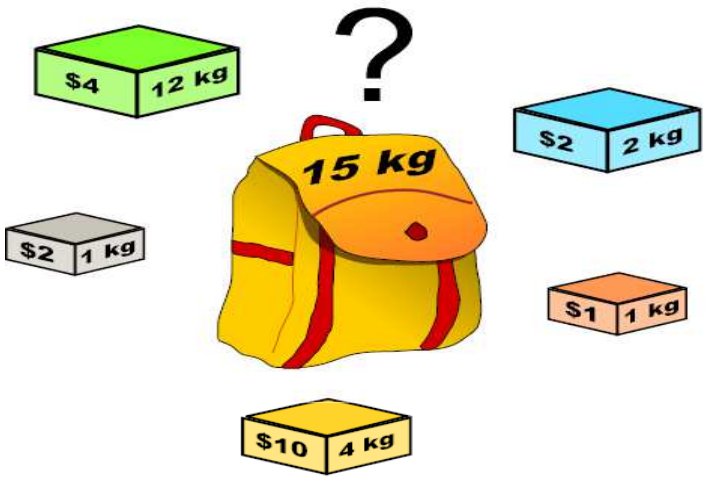
$$T = \begin{pmatrix} 6 & 0 & 5 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Outline

- Tabu Search
- Solving Permutation Optimization
- Solving Graph Problems
- Solving Constraint Satisfaction Problems
- Vehicle Routing Problem (VRP)
- Assignment Problem
- **Knapsack Problem**
- Adaptation
- Cooperation
- Summary

Knapsack Problem

The knapsack problem is a widely studied **NP-Complete** problem and can be stated as follows:



- A set of ***n* items** is given and for each item *i* its **utility u_i** and its **weight w_i** is known.
- A knapsack of **capacity C** is also given.
- The problem is to find a **subset of the items** set which can be taken into the knapsack such as the following constraints hold:
  - ◇ the knapsack **capacity is not overloaded**;
  - ◇ the total **utility** of the items taken is **maximum**.

[2]

Knapsack Problem

Assume the following:

- Knapsack capacity: C= 50;
- Number of items n= 7 (each item will be identified by its number)

Item	Weight	Utility
1	10	2
2	12	1
3	15	3
4	27	4
5	30	1
6	20	3
7	7	1

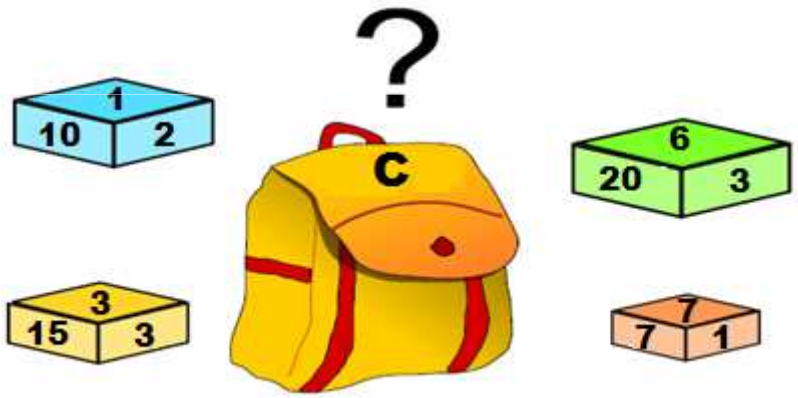
[2]

Knapsack Problem

- **Initial Solution**

Initial solution: 1 0 1 0 0 1 1 corresponds to the items 1, 3, 6 and 7 to be considered.

Item	Weight	Utility
1	10	2
2	12	1
3	15	3
4	27	4
5	30	1
6	20	3
7	7	1



Knapsack Problem

- **Evaluation Function**

We will penalize any overload of the knapsack capacity with an amount of 50.

f_1 = the difference (in absolute value) between the **total weight** of the selected items and the **knapsack capacity** + the **penalty** (50) in case the capacity is overloaded (and this criterion is to be minimized).

$$f_1 = \left| \sum_{i=1}^{m} w_i - C \right| + 50 \Big|_{if \sum_{i=1}^{m} w_i \succ C} \quad \text{m is number of selected items}$$

Knapsack Problem

- **Evaluation Function**

But we have to take into account the utility of the selected objects

f_2 = utility of the selected objects (should be as high as possible).

$$f_2 = \sum_{i=1}^{m} u_i \qquad \text{m is number of selected items}$$

- **Overall Evaluation Function**

$$f = f_1 + f_2$$

f_1 : should be minimized

f_2 : should be maximized

f :??!! How to resolve this conflict?

Knapsack Problem

- **Overall Evaluation Function**

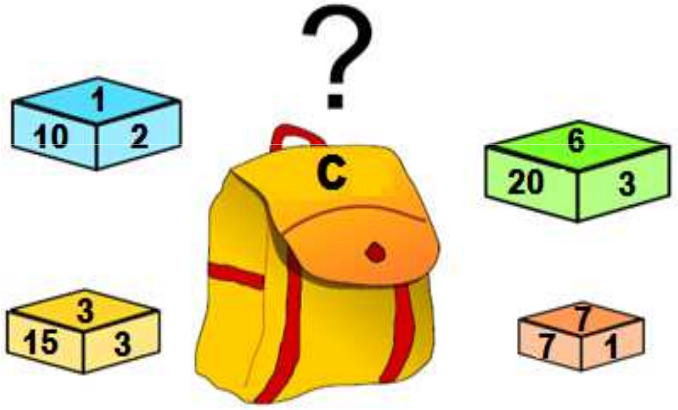
$$f = f_1 + (-f_2)$$

f: should be minimized

For the initial state: 1 0 1 0 0 1 1

$$f_1 = \left|(10+15+20+7)-50\right|+50 = 52$$

$$f_2 = 2+3+3+1 = 9$$



The value of f for the initial state is f= 52-9=43.

Knapsack Problem

- **Iteration-1**

Initial solution: 1 0 1 0 0 1 1

The value of f for the initial solution is 52-9=43.

The tabu list is the empty set.

◇ Neighboring Solutions

	solution	Move	*f*
	1110011	Add-2	54
	1011011	Add-4	66
	1010111	Add-5	72
*	0010011	Delete-1	1
	1000011	Delete-3	7
	1010001	Delete-6	12

Knapsack Problem

- **Iteration-2**

Initial solution: 0 0 1 0 0 1

f=1.

◇ Neighboring Solutions

	solution	Move	f
	0110011	Add-2	46
	0011011	Add-4	58
	0010111	Add-5	64
*	0010010	Delete-7	9
	0000011	Delete-3	23
	0010001	Delete-6	24

◇ Tabu List

1	3
2	
3	
4	
5	
6	
7	

Knapsack Problem

- **Iteration-2**

Initial solution: 0 0 1 0 0 1 f=1.

Current solution: 0 0 1 0 0 1 0 f=9

There is no solution obtaining a better value for f at this step. Thus, a new set of neighboring solutions will be generated.

The process will iterate this way (and we leave the remaining as an exercise). The solution with the optimal value is 0 0 1 1 0 1 0, with items 3, 4 and 6 considered and with a value f = -7.

It might take quite a lot time to reach this solution by allowing only one move at a time.

If we will allow at least **two moves to be performed at a step**, the solution will be reached much faster.

Knapsack Problem

- **Iteration-1**

Initial solution: 1 0 1 0 0 1 1 f=43.

Allowing **two moves** to be performed at a step

◇ Neighboring Solutions

	solution	Move	f
	0110011	Add-2 & Delete-1	46
	1011001	Add-4 & Delete-6	49
	1011010	Add-5 & Delete-7	60
*	1010001	Delete-6	12
	1110010	Add-2 & Delete-7	48
	1010111	Add-5	72

Knapsack Problem

- **Iteration-2**

Current solution: 1 0 1 0 0 0 1 f=12.

Tabu{6}

◇ Neighboring Solutions

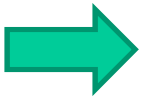
	solution	Move	f
	0010101	Add-5 & Delete-1	44
*	0011001	Add-4 & Delete-1	-7
	1001001	Add-4 & Delete-3	-1
	1110001	Add-2	-1
	0110001	Add-2 & Delete-1	11
	1000001	Delete-3	3-

◇ Tabu List

1	
2	
3	
4	
5	
6	3
7	

Knapsack Problem

- **Iteration-3**

Current solution: 0 0 1 1 0 0 1 f=-7.  This is the final solution

Tabu{6,4,1}

◇ Neighboring Solutions

	solution	Move	f
	0010101	Add-5 & Delete-1	44
*	0011001	Add-4 & Delete-1	-7
	1001001	Add-4 & Delete-3	-1
	1110001	Add-2	-1
	0110001	Add-2 & Delete-1	11
	1000001	Delete-3	3-

◇ Tabu List

1	3
2	
3	
4	3
5	
6	2
7	

Outline

- Tabu Search
- Solving Permutation Optimization
- Solving Graph Problems
- Solving Constraint Satisfaction Problems
- Vehicle Routing Problem (VRP)
- Assignment Problem
- Knapsack Problem
- **Adaptation**
- Cooperation
- Summary

Adaptation

One of the most important parameters of the TS is the length of the tabu list.

Too Small

search can be trapped into cycles

Too Long

too many moves could be prevented at each iteration

Adaptation

One of the adaptive approaches incorporated into TS is to allow the length of the short term memory (tabu list) to **vary dynamically**,

These approaches were introduced by Taillard [3] and Dell' Amico and Trubian [4].

[1]

Adaptation

In [3], the adaptive approach was as follows:

- A range for the tabu list length in computed in advance according to the **problem size**,
- A new list length is **randomly selected** from this range **every predetermined number of iterations**.

[1]

Adaptation

In [4], the adaptive approach was as follows:

- The tabu list length is restricted between $\mathbf{L_{min}}$ and $\mathbf{L_{max}}$,
- If the **current solution is better than the best-so-far**, the tabu **list length is set to 1**,
- If in an improving phase (the solution has **improved** over the last iteration) the tabu list length is **decreased by** 1,
- If not in an improving phase (the solution has **deteriorated** over the last iteration) the tabu list length is **increased by 1**,
- The values of $\mathbf{L_{min}}$ and $\mathbf{L_{max}}$ are randomly changed every A iterations.

[1]

Adaptation

According to the approach described in [4]

Exploration/Diversification **Exploitation/Intensification**

Tabu Tenure

If the current solution has **deteriorated** the tabu list length is **increased** in order to guide the **search away** from an apparently bad region

If the current solution has **improved** the tabu list length is **decreased** in order **to focus** the search in a region of potential improvement

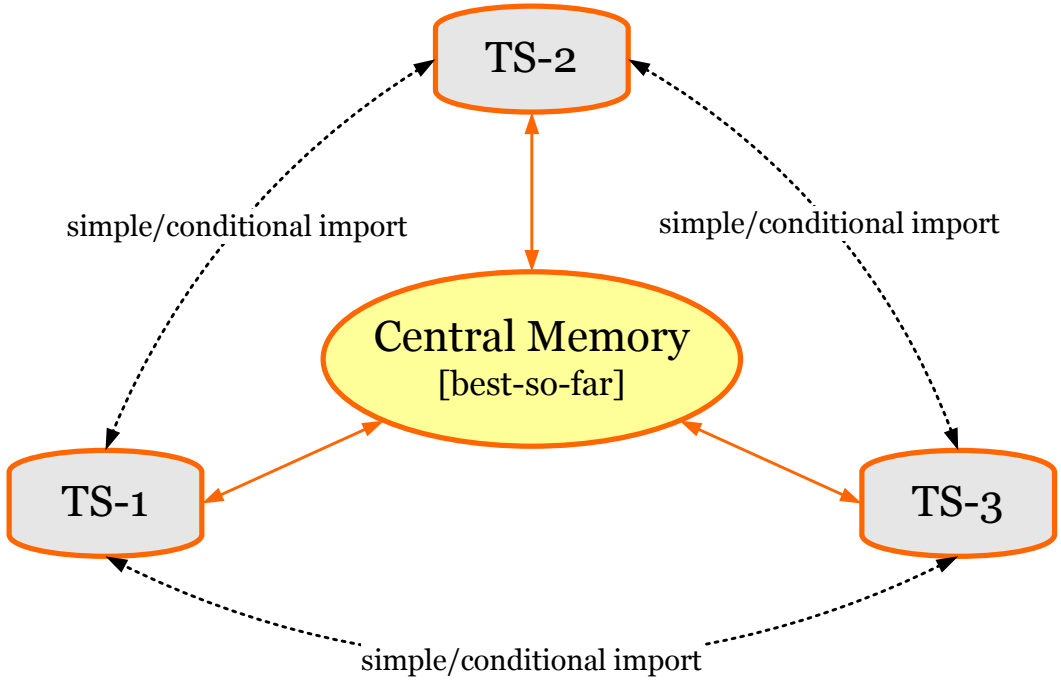
Outline

- Tabu Search
- Solving Permutation Optimization
- Solving Graph Problems
- Solving Constraint Satisfaction Problems
- Vehicle Routing Problem (VRP)
- Assignment Problem
- Knapsack Problem
- Adaptation
- **Cooperation**
- Summary

Cooperation

In Crainic et al. [5], the authors studied the idea of having **p independent tabu searches** working **concurrently** and **exchanging information** every predetermined number of iterations (**synchronous communication**),

This approach was referred to as **coordinated searches**.



[1]

Cooperation

In [5], the different search processes may use different:

- Initial solutions,
- Search strategies (parameter settings),
- Ways of handling the incoming solution:
  - ◇ Replacing its **own best-so-far solution** (**forced diversification**), referred to as **simple import**,
  - ◇ Replacing its own best-so-far solution **only if the incoming solution is better**, referred to as **conditional import**.

Another implementation is found in [6], where the **tabu lists are reset after any synchronization step**. [1]

Cooperation

General conclusions found were:

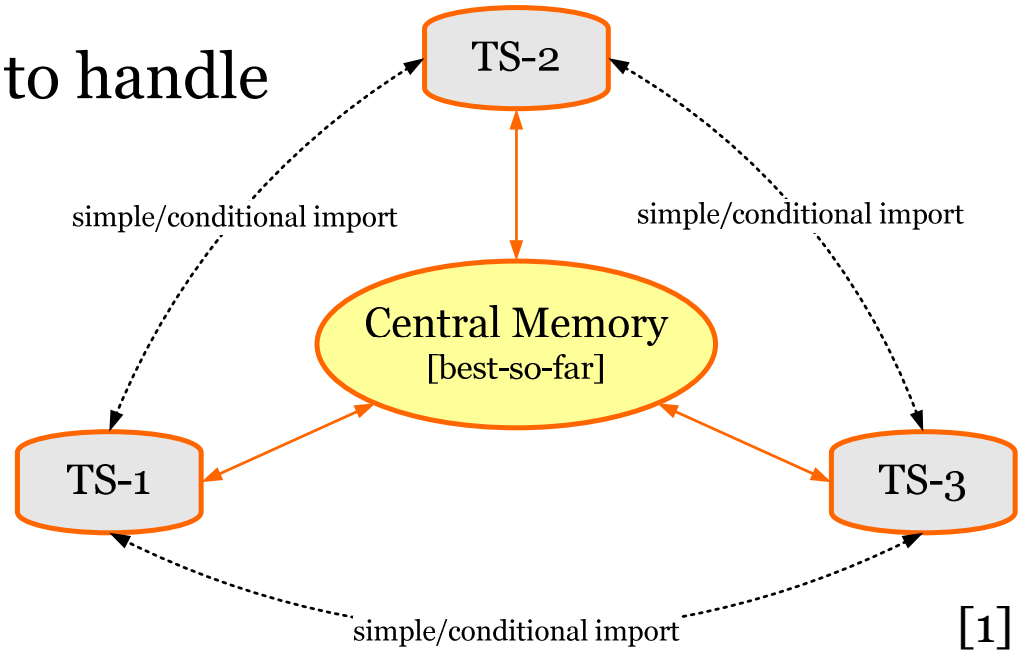
- For a **fixed number of iterations**, increasing the number of processors **improved** the obtained solution up to a certain point,
- **Reducing the number of iterations** between synchronization steps **increases the computational times** because of the increased **message passing overhead**,
- **Conditional import** always produces **better** or similar results to **simple import**.

[1]

Cooperation

An **asynchronous communication** approach was proposed in [7],

- Instead of exchanging information **every predetermined number of iterations**, each search process **only broadcasts** information when its **best-so-far solution** is updated,
- A **central memory** is used to handle the information exchange.



[1]

Cooperation

The mechanism used is as follows:

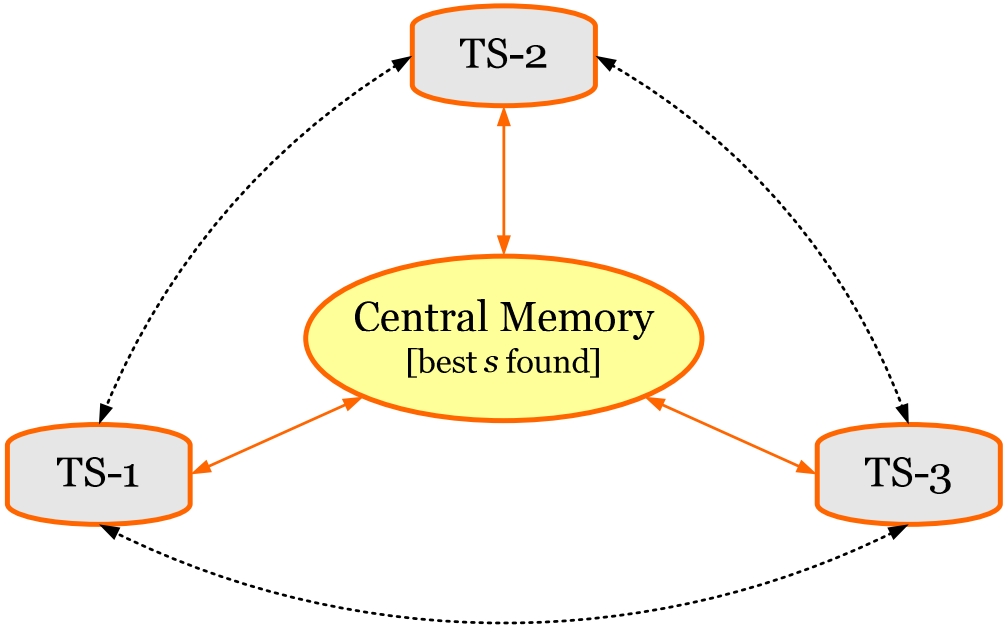
- Each search process sends its **best-so-far solution** to the central memory when it gets updated,
- If the sent solution is **worse** than the solution available in the central memory, the search process uses the **stored solution**,
- If its best-so-far solution is **not improved** for a specified number of iteration, the **search process requests the solution stored in the central memory**.

[1]

Cooperation

Another approach is to implement the **central memory as a pool of the best *s* found solutions**,

When a search process requests a solution from the central memory, it gets a **randomly selected solution** from the **pool** rather than always getting the best one.



[1]

Outline

- Tabu Search
- Solving Permutation Optimization
- Solving Graph Problems
- Solving Constraint Satisfaction Problems
- Vehicle Routing Problem (VRP)
- Assignment Problem
- Knapsack Problem
- Adaptation
- Cooperation
- **Summary**

Summary

- Tabu search is a metaheuristic that guides a local search procedure to explore the solution space beyond local optimality
- Memory-based strategies are the hallmark of tabu search approaches.
- TS accepts non-improving solutions in order to escape from local optima (where all the neighbouring solutions are non-improving).
- TS uses past experiences to improve current decision making by using memory (a “tabu list”) to prohibit certain moves – “makes tabu search attempt to be a **global** optimizer rather than a local optimizer.”

References

1. M. Kamel. ECE457A: Cooperative and Adaptive Algorithms. University of Waterloo, 2010-2011.
2. Crina Grosan and Ajith Abraham. *Intelligent Systems: A Modern Approach*. Springer, 2011.
3. E. Taillard. “Parallel Taboo Search Techniques for the Job Shop Scheduling Problem”. ORSA Journal on Computing, vol. 6, no. 2, 108-117, 1994.
4. M. Dell’Amico and M. Trubian. “Applying Tabu Search to the Job Shop Scheduling Problem”. Annals of Operation Research, vol. 41, no. 3, pp. 231-252, 1993.
5. T. G. Crainic, M. Toulouse and M Gendreau. “Synchronous Tabu Search Parallelization Strategies for Multicommodity Location-Allocation with Balancing Requirements”. Centre de recherche sur les transports, Universite de Montreal, Tech. Rep. 934, 1993.
6. M. Malek, M. Guruswamy, M. Pandya and H. Owens. “Serial and Parallel Simulated Annealing and Tabu Search Algorithms for the Travelling Salesman Problem”. Annals of Operation Research, vol. 21, pp. 59-84, 1989.
7. Revees, C., (ed) Modern Heuristic Techniques for Combinatorial Problems. Ch 3 by Glover and Languna.