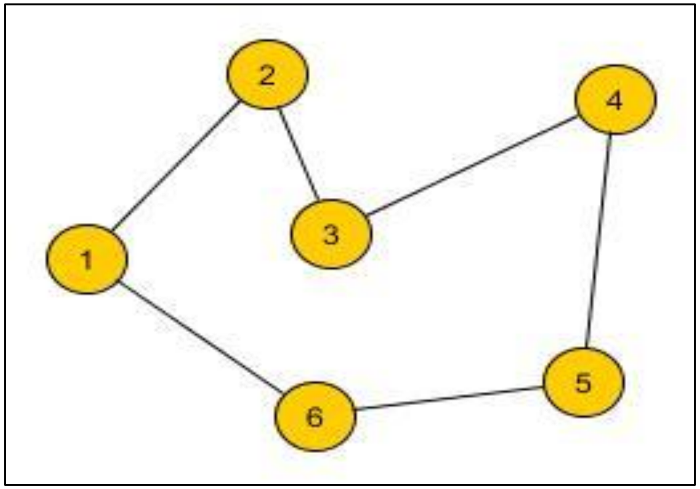
CSE 402
Offline 3

Local Search

- Technique used for difficult optimization problem or constraint satisfaction problem.
- For some problem, we don't need the path to the solution

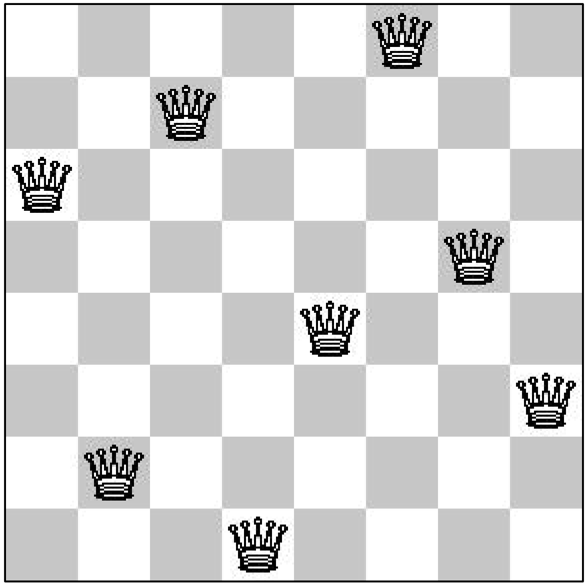
Traveling Salesperson Problem(TSP)

- Given *n* points/cities and distance between any two pair of cities.
- Can you find a tour with minimum distance visiting each city?



N queens Problem

- How can *n* queens be placed on an *NxN* chessboard so that no two of them attack each other?



Key Idea

- A local search algorithm usually looks like following
  1. Pick an initial state (Randomly or using some heuristics)
  2. Make local modification to improve current state
  3. Repeat step 2 until goal state found (or out of time)

Some Issues

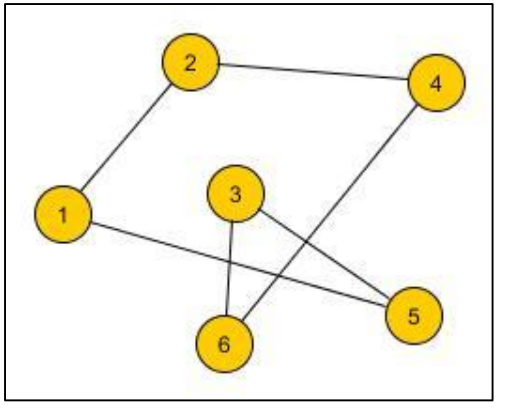
- What is a state?
- How you define local?
- How to measure improvement?

What is a state?

- State can be partial or complete solution
- Usually a complete solution

Solution in TSP

- Solution in TSP may be a tour.



- We can represent it using a vector of integers.

<1,2,4,6,3,5>

Solution in *n*-queens

- Solution in TSP may be an assignment of queens in the chessboard.

		Q	
	Q		
			Q
Q			

- We can represent it using a 2d array or 1d array

0 0 1 0

0 1 0 0

0 0 0 1 <3,2,4,1>

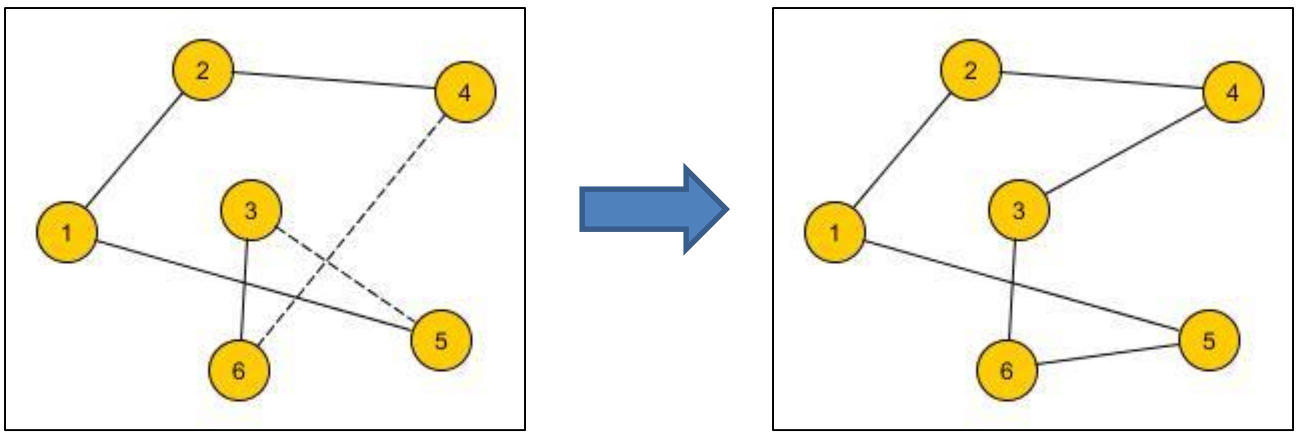
1 0 0 0

How you define local?

- Small change in current state/solution.
- Usually we define one or more neighborhood function(s).

Neighborhood in TSP

- 2-opt operator:



Neighborhood in *n*-queens

- Swap two rows:

		Q	
	Q		
			Q
Q			

Q			
	Q		
			Q
		Q	

How to measure improvement?

- For optimization problem we have a objective function to optimize.
- For CSP, we can use a heuristic function.

How to measure improvement?

- In TSP:
  - Cost of current tour. (Improvement means reduction of cost)
- In *n*-queens:
  - # of conflicts. (Improvement means less number of conflicts)

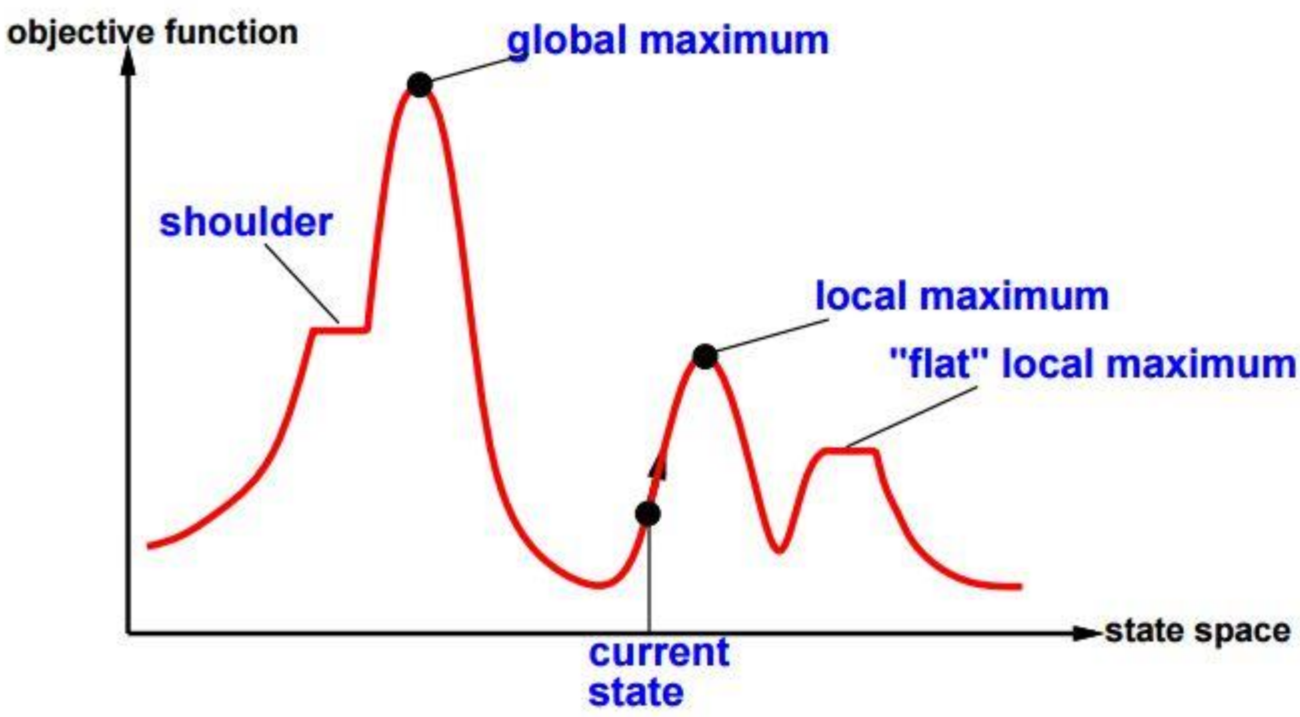
Hill Climbing

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node
    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
    end
```

- At each step, move to a neighbor of higher value in hopes of getting to a solution having the highest possible value
- Can easily modified for minimization problem

Hill Climbing

- Can stuck into local optima



Variations

- Steepest ascent hill climbing:
  - Take best neighbor
- Stochastic hill climbing:
  - Select random better neighbor
- First choice hill climbing:
  - Take first better randomly generated neighbor
- Random restart hill climbing:
  - Restart again from random state if you stuck in local optima

Simulated annealing

- Basic Idea:
  - Like hill climbing identify the quality of local improvements
  - Assume that change in objective function is δ
  - If δ is positive, move to that state
  - Otherwise move to that state with a probability proportional to δ and *T(?)*
  - Over time make it less likely to accept bad moves.

Simulated annealing

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling prob. of downward steps

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  for $i \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow$ *schedule*[i]
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* ← *next*
    else with probability $e^{\Delta E/T}$, set *current* ← *next*

Simulated annealing

- A Physical Analogy:
  - imagine letting a ball roll downhill on the function surface – this is like hill-climbing (for minimization)
  - now imagine shaking the surface, while the ball rolls, gradually reducing the amount of shaking – this is like simulated annealing

Scheduling function

- T should be large at beginning and gradually decrease
- If T is lowered slow enough, then we can reach global optima
- What do you mean by slow enough?
- In literature different function can be found:
  - Linear Cooling: $schedule(T) = T_0 - \mu t$
  - Exponential Cooling: $schedule(T) = T_0\, \alpha^t$, $0<t<1$
  - Logarithmic Cooling: $schedule(T) = c/log(1+t)$

Offline-3

Offline

- You have to implement local search algorithms for *n*-queens problem.
- **Algorithms**:
  - Steepest Ascent hill climbing
  - Simulated annealing
- **Representation**:
  - A vector of size *n* where each *i*-th is an integer representing the column number of the queen in the *i*-th row.

i)Steepest Ascent Hill Climbing

- You have to implement first choice hill climbing in following manner

Algorithm:

```
current ← randomly generated state
max_step← maximum number of iteration
count ← 1
 for( i ← 0 ; i < max_step ; i++, count++) {
        neighbor← best successor of current state according to
                  neighborhood function
        if( conflict(neighbor) >= conflict(current) )
                  return current;
        current← neighbor
 }
return current
```

i)Steepest Ascent Hill Climbing

- Neighborhood function:
  - Select a row randomly.
  - Move the queen in that row in any other column.

- Evaluating function:
  - Total no. of conflicts.

Q			
	Q		
	Q		
			Q

No of conflicts =4

ii)Simulated Annealing

- You have to implement Simulated Annealing algorithm shown previously.
- Also store best result so far.
- You will use linear cooling scheme as scheduling function.
- Neighborhood and Evaluation function is same as shown in Hill climbing approach.

Main function

- You will have to run hill climbing and simulated annealing for same initial state.

```
for(i=1 to 10)
        Solution x = GenerateRandomState()
        HillClimb(x)
        SimulatedAnnealing(x)
```

Output

- You have to run your code for n=4,6,8,…,12
- For each n run your program ten times.
- Generate following table for and submit it along with source code.
- Also discuss the result.

n	Average No of Iteration (HC)	Average Conflicts (HC)	Minimum Conflicts (HC)	Average No of Iteration (SA)	Average Conflicts (SA)	Average Conflicts (SA)

Submission Deadline

- Deadline is 8 November 2016, 2:00 AM
- And Please Do not Copy