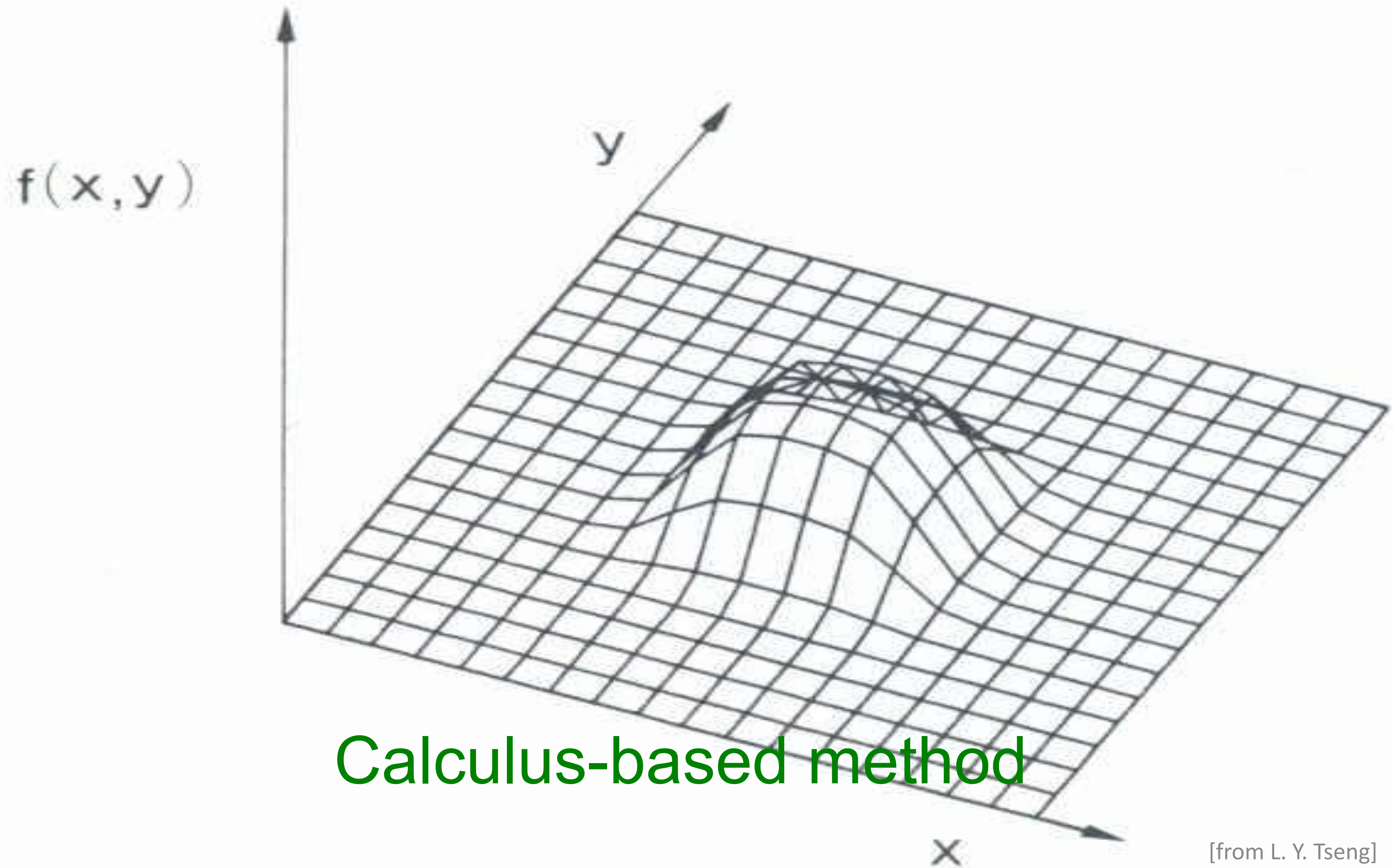
Session 8
Local Search and Metaheuristics

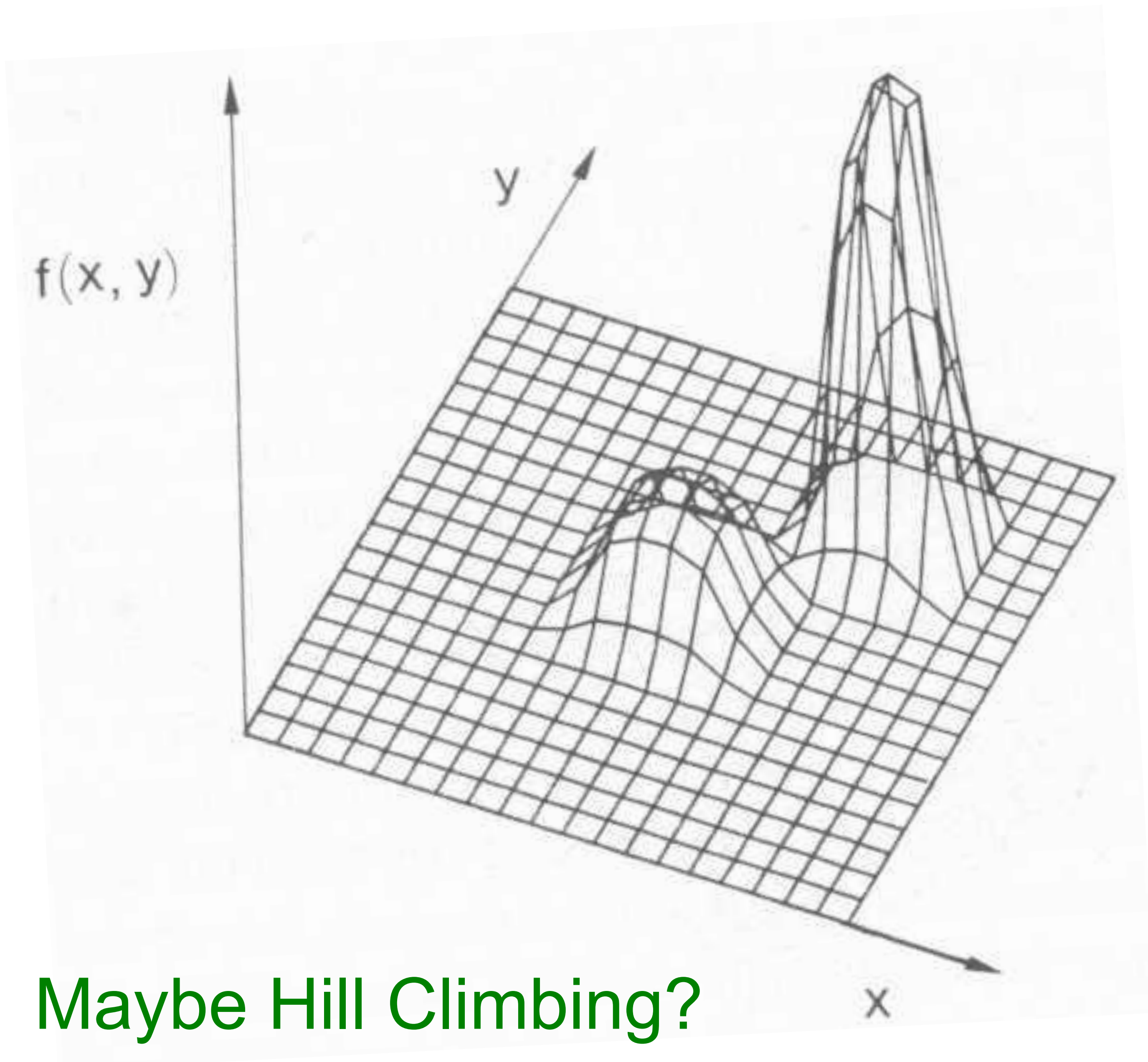
Daniel Diaz
Philippe Codognet
Salvador Abreu

Optimization problems



Calculus-based method

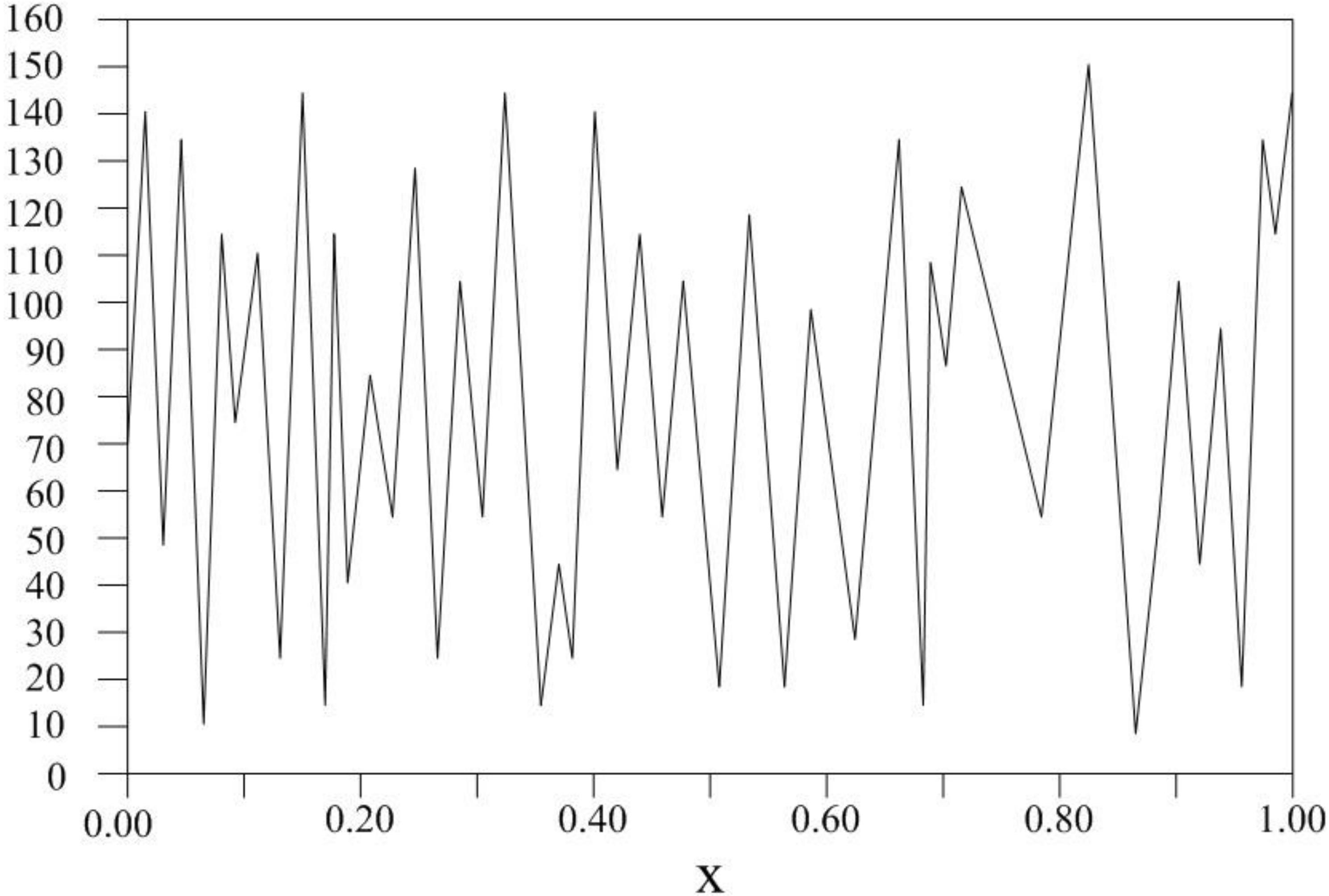
[from L. Y. Tseng]



Maybe Hill Climbing?

[from L. Y. Tseng]

How about this?



[from L. Y. Tseng]

COP: Motivation for Heuristic Methods

- NP-complete problems (most of the time, but not all)
- Optimization at least as hard as decison
- NP-complete decision problem -> NP-hard optimization problem
- For NP-hard COP there is probably no exact method where computing time is limited by a polynomal (in the instance size)
- Different choices
  - Exact methods (enumerative)
  - Approximation method (polynomial time)
  - Heuristic method (no a priori guarantees)
- In the real world:
  - Often requirements on response time
  - Optimization only one aspect
  - Problem size and response time requirements often rules out exact solution methods

[From A. Løkketangen]

Heuristics & Metaheuristics

- From ancient greek *ευρίσκω* : « I find »
- current meaning: ”a technique to guide the search toward the solution”
- Basic idea: should exploit problem knowledge
- The term *heuristic* was introduced in the book “How to solve it” [Polya 1957] (A guide for solving math problems)
- *meta* is from ancient greek μετά : above, beyond
  - e.g. metaphysics …
- Meta-heuristics are higher-level strategies to guide the search
- In particular to avoid getting trapped in local optima

How to Guide the Search ?

1. Find a solution
   - Random configuration
   - Greedy solution
   - …

2. Check "how good" this solution is
   e.g. use the value of the objective function

3. Try to find a better solution
   i.e. try to identify best direction to go to

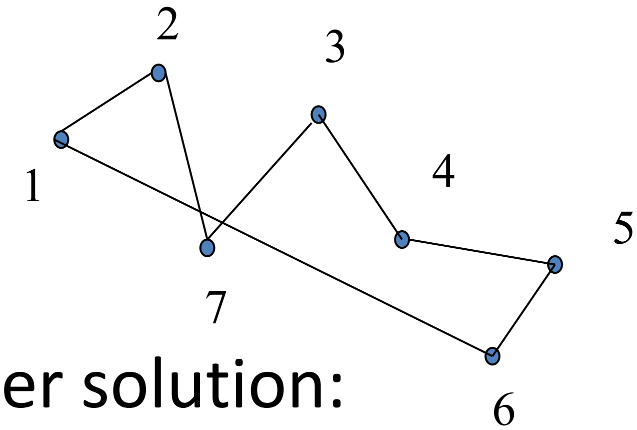
4. continue

Given a Solution, How to Find a Better One?

- Modification of a given solution gives a ”neighbor solution”
- A certain set of **operations** on a solution yields a set of neighbor solutions, called a **neighborhood**
- Evaluations of neighbors
  - Objective function value
  - Feasibility ?

[From A. Løkketangen]

Example of TSP



Earlier solution:

1 2 7 3 4 5 6 1 (184)

Trivial solution:

1 2 3 4 5 6 7 1 (288)

Greedy construction:

1 3 5 7 6 4 2 1 (160)

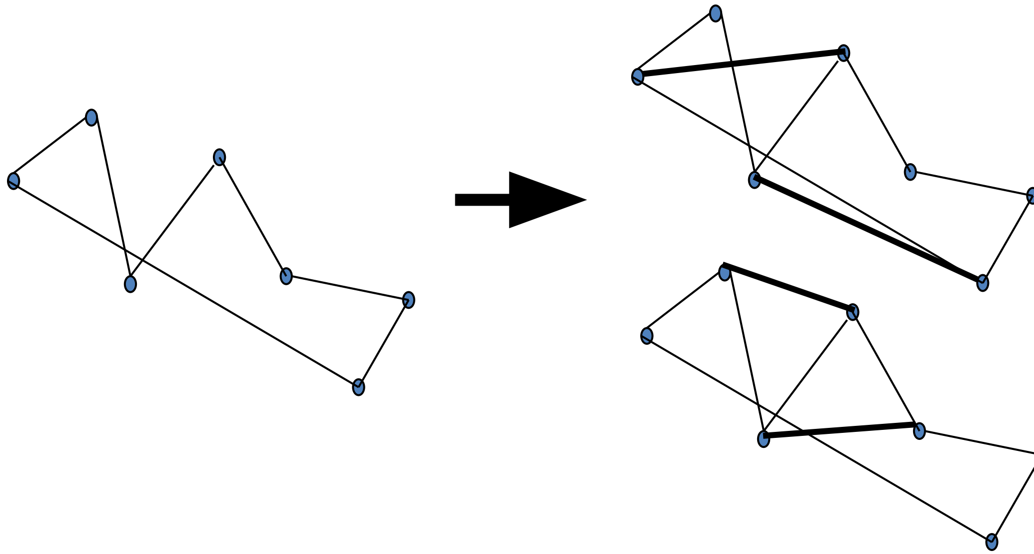
	1	2	3	4	5	6	7
1	0	18	**17**	23	23	23	23
2	**2**	0	88	23	8	17	32
3	17	33	0	**23**	**7**	43	23
4	33	**73**	4	0	9	23	19
5	9	65	6	65	0	54	**23**
6	25	99	2	**15**	23	0	13
7	83	40	23	43	77	**23**	0

... Better solutions ?

[From A. Løkketangen]

Example of TSP (2)

- Operator: 2-opt
  - choose 2 edges
    {a1,b1} & {a2,b2}
  - replace by
    {a1,a2} & {b1,b2}



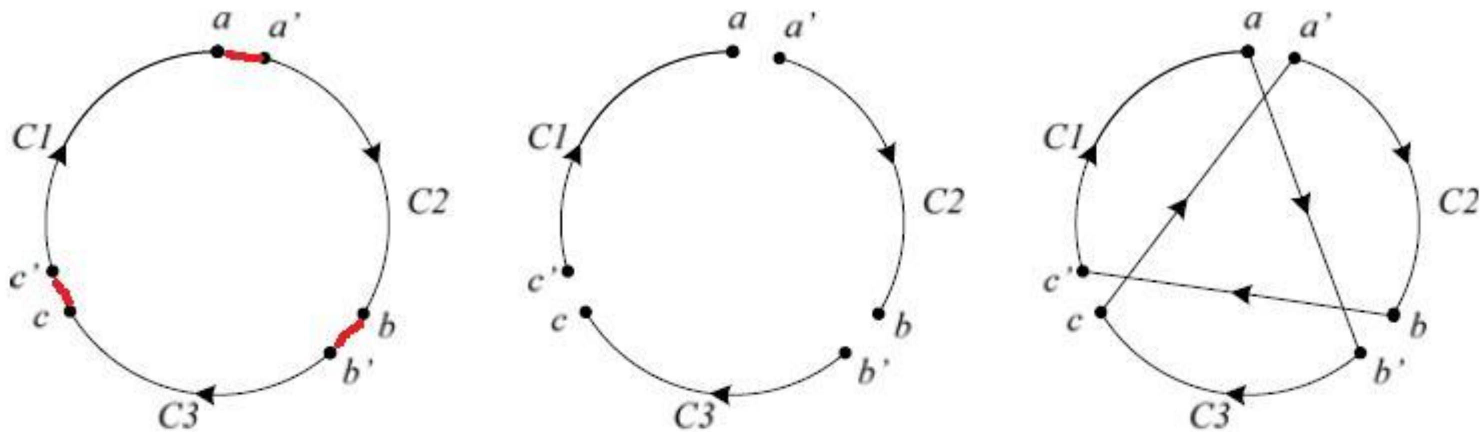
- Operator: 1-opt
  - swap 2 cities in the tour

  $(c_1, c_2, \ldots, c_i, \ldots, c_j, \ldots c_n)$
  becomes $(c_1, c_2, \ldots, c_j, \ldots, c_i, \ldots c_n)$

[From A. Løkketangen]

Example of TSP (3)

- Operator: 3-opt



- size of neighborhood ?
  - $O(n^k)$ for k-opt
  - Can become quite big for large problem instances

Example of Knapsack Problem

- Knapsack with capacity 101
- 10 items: 1,...,10
- Trivial solution: empty backpack, value 0
- Greedy solution, assign the items after value:
  - (0000010000), value 85
  - Better solutions?

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52

[From A. Løkketangen]

Example of Knapsack Problem (2)

- Greedy 0000010000 value 85
- Another solution 0010100000 value 73
- Natural operator: ”Flip” a bit
  - If the item is in the knapsack, take it out
  - If the item is not in the knapsack, include it
- Some Neighbors of 0010100000 :
  - 0110100000 value 105
  - 1010100000 value 152, not feasible
  - 0010000000 value 47

	1	2	3	4	5	6	7	8	9	10
Value	79	32	47	18	26	85	33	40	45	59
Size	85	26	48	21	22	95	43	45	55	52
	0	0	1	0	1	0	0	0	0	0

[From A. Løkketangen]

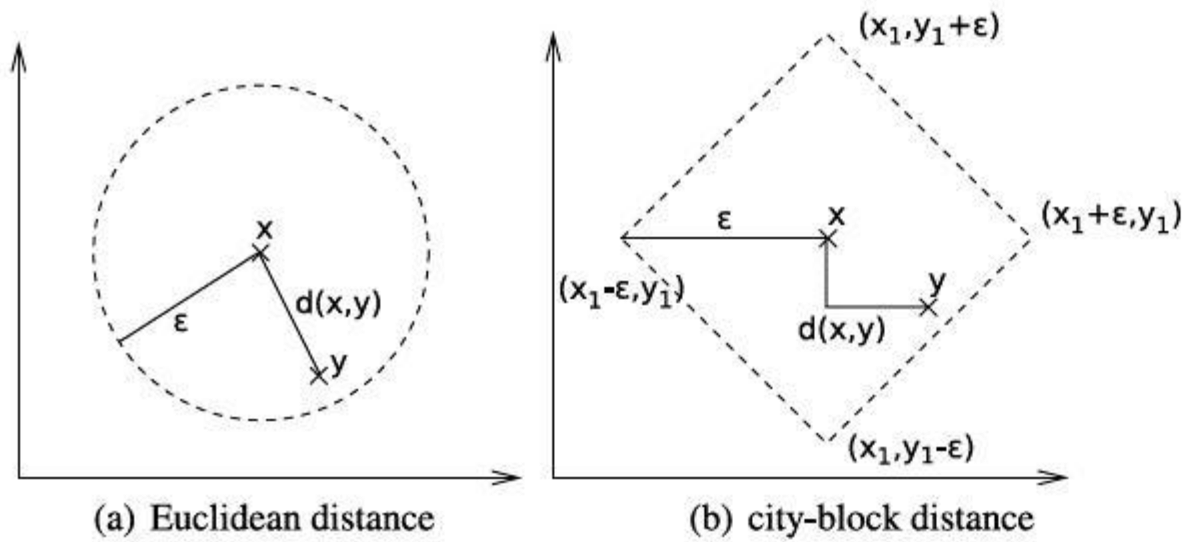
Definition: Neighborhood

- Let (S,f) be a COP-instance
- A *neighborhood function* is a mapping from solutions to the set of possible solutions (i.e.,reached by a move) $N : S \rightarrow 2^S$
- For a given solution $s \in S$, *N* defines a *neighborhood* of solutions: $N(s) \subseteq S$ i.e., solutions in some sense ”near” s
- $t \in N(s)$ is a *neighbor* of s

[From A. Løkketangen]

Neighborhood

- A neighborhood somehow define a notion of proximity, thus *distance* between solutions
- It can be any type of distance, not only Euclidian
- e.g. Manhattan distance or Hamming distance



[From F. Rothlauf]

Neighborhood Operators

- Neighborhoods are most often defined by a given operation on solutions

- e.g. simple operations
  - Remove/Add an element (cf knapsack)
  - Interchange two or more elements of a solution, e.g. swap two elements (cf TSP)
  - changing the value of one or a few elements in solutions defined as vectors of decision variables

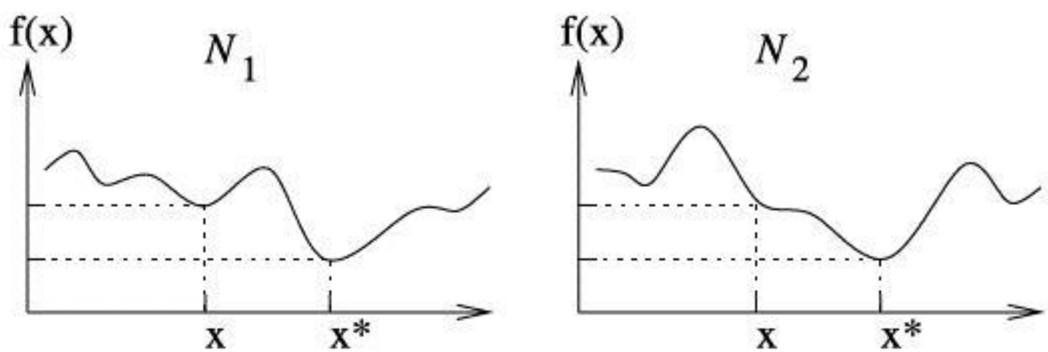
[From A. Løkketangen]

Different neighborhoods

- Different neighborhood operators for same problem:
  - e.g. 1-opt and 2-opt for the TSP

$$N_\sigma(s), \sigma \in \Sigma$$

- Idea: change dynamically between them
  - change in the shape of the "landscape"
  - can help avoiding local minima



Changing the neighborhood from N_1 to N_2

[From F. Rothlauf]

(recall) Terminology: Optima

- Assume we want to solve $$\max_{x \in \mathcal{F} \subseteq \mathcal{S}} f(x)$$
- Let **x** be our current (incumbent) solution in a local search
- If f(**x**) ≥ f(**y**) for all **y** in **F**, then we say that **x** is a **global optimum** (of f)

[From A. Løkketangen]

(new) Terminology: Optima

- assume that **N** is a neighborhood operator, so that **N**(**x**) is the set of neighbors of **x**

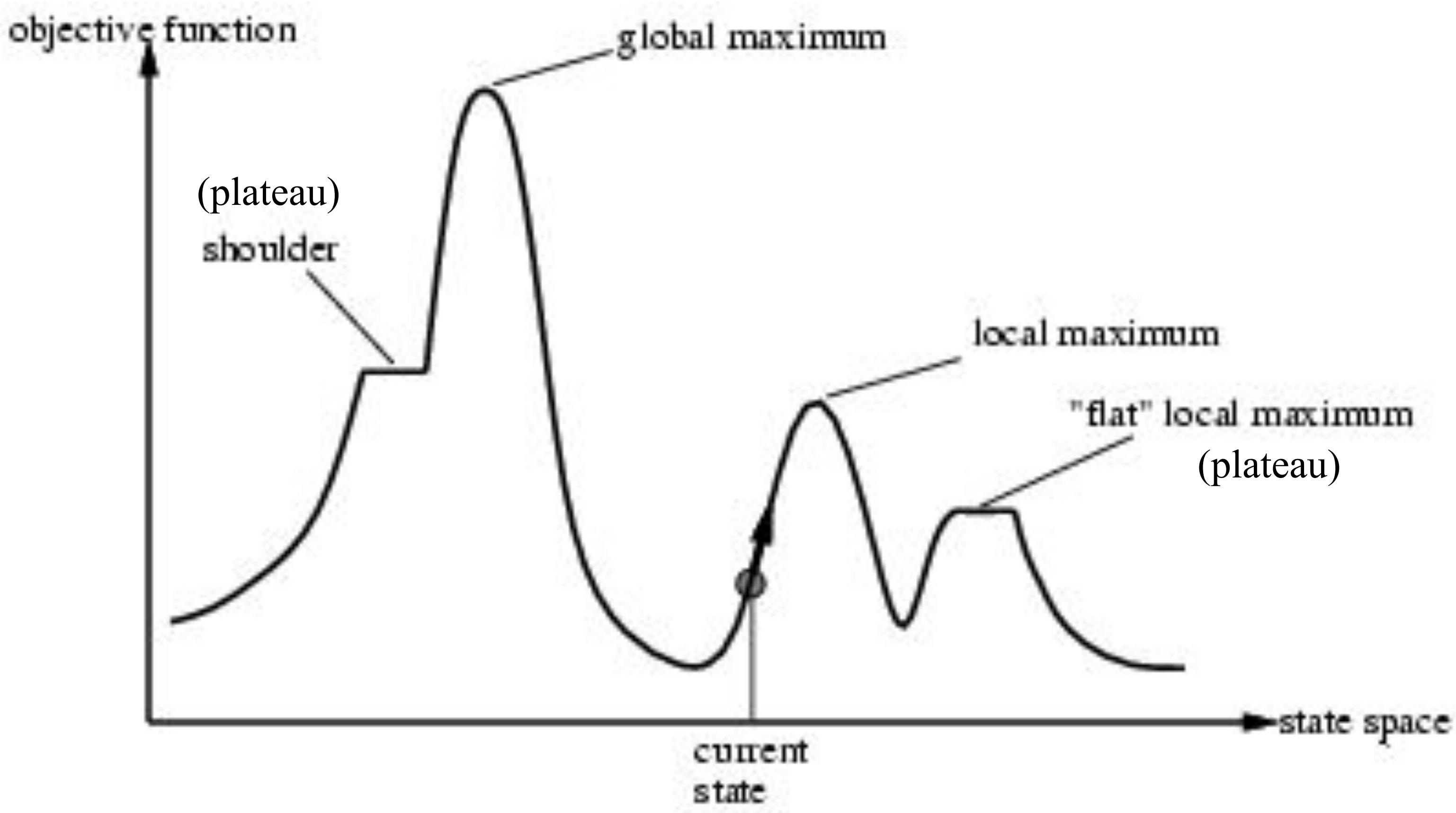
- If f(**x**) ≥ f(**y**) for all **y** in **N**(**x**), then we say that **x** is a **local optimum**

  (of f, with respect to neighborhood operator **N**)

A neighborhood function **N** is **exact** if every local optima w.r.t. **N** is also a global optima

[From A. Løkketangen]

Graphically



[from Russell & Norvig 2010]

Example of Local Search : Sorting

- Problem: sort a sequence of numbers from 1 to n
- Consider sequences of numbers in {1,…n} as solutions
- Objective function: $f(\pi) = \sum_{i=1}^{n} i * \pi_i$ (to maximize)
- This function guarantees that the optimum is the configuration sorted in increasing order
- *K-exchange neighborhood*:
  - Exchanging k elements in a given sequence or partition
- K-exchange neighborhood is exact for sorting
- 1-exchange neighborhood restricted to adjacent pairs

Example of Local Search : Sorting

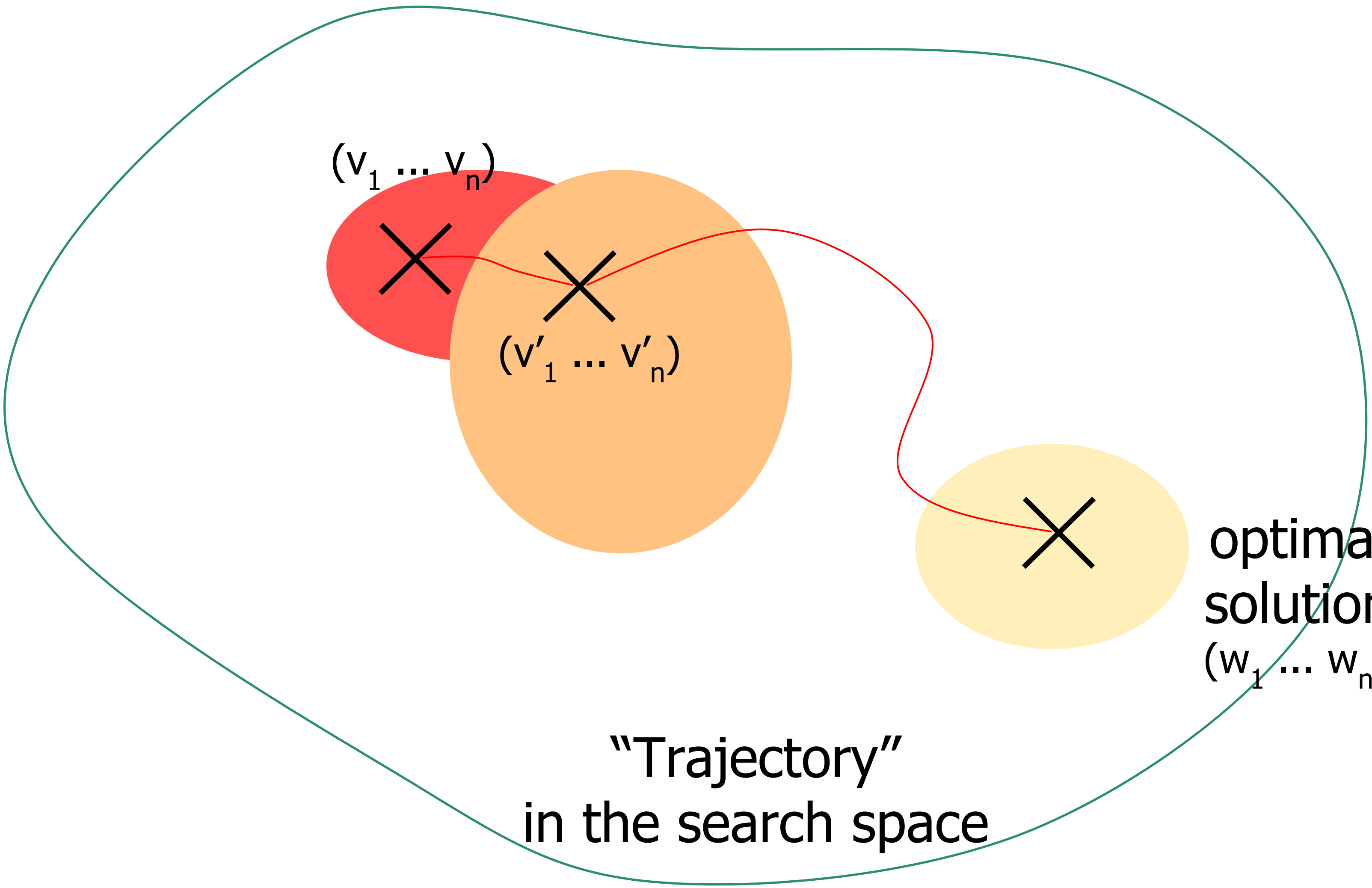
- Problem: sort a sequence of numbers from 1 to n
- Consider sequences of numbers in {1,…n} as solutions
- Objective function: $f(\pi) = \sum_{i=1}^{n} i * \pi_i$ (to maximize)
- This function guarantees that the optimum is the configuration sorted in increasing order
- *K-exchange neighborhood*:
  - Exchanging k elements in a given sequence or partition
- K-exchange neighborhood is exact for sorting
- 1-exchange neighborhood restricted to adjacent pairs
- This is bubble sort !

(Basic) Bubble Sort Algorithm

```
Swapped = false
Repeat
  For i=1 to n-1
   if(A[i] > A[i+1])
     swap(A[i],A[i+1])
     swapped = true
Until not swapped
```

1	2	3	4	5	6
77	42	35	12	101	5

Iterative Improvement



Basic Local Search : Iterative Improvement

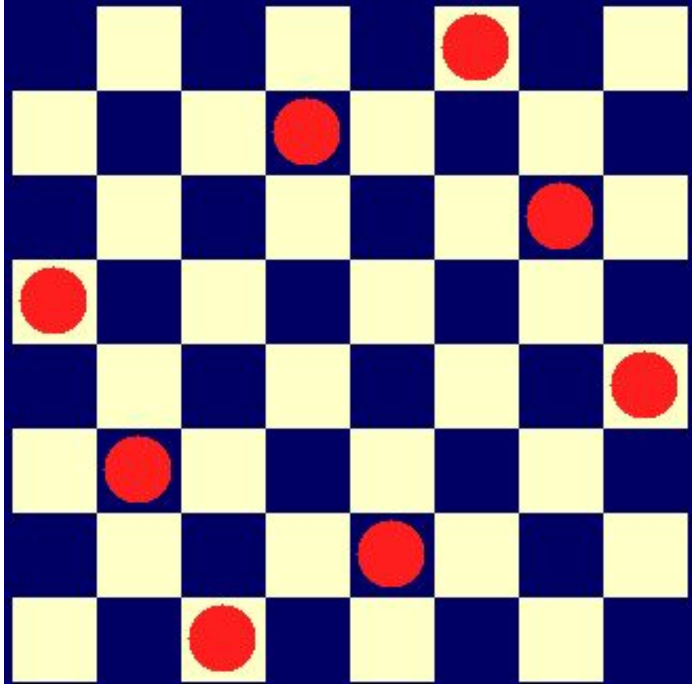
$s \leftarrow$ GenerateInitialSolution()
repeat
  $s \leftarrow$ Improve($\mathcal{N}(s)$)
until no improvement is possible

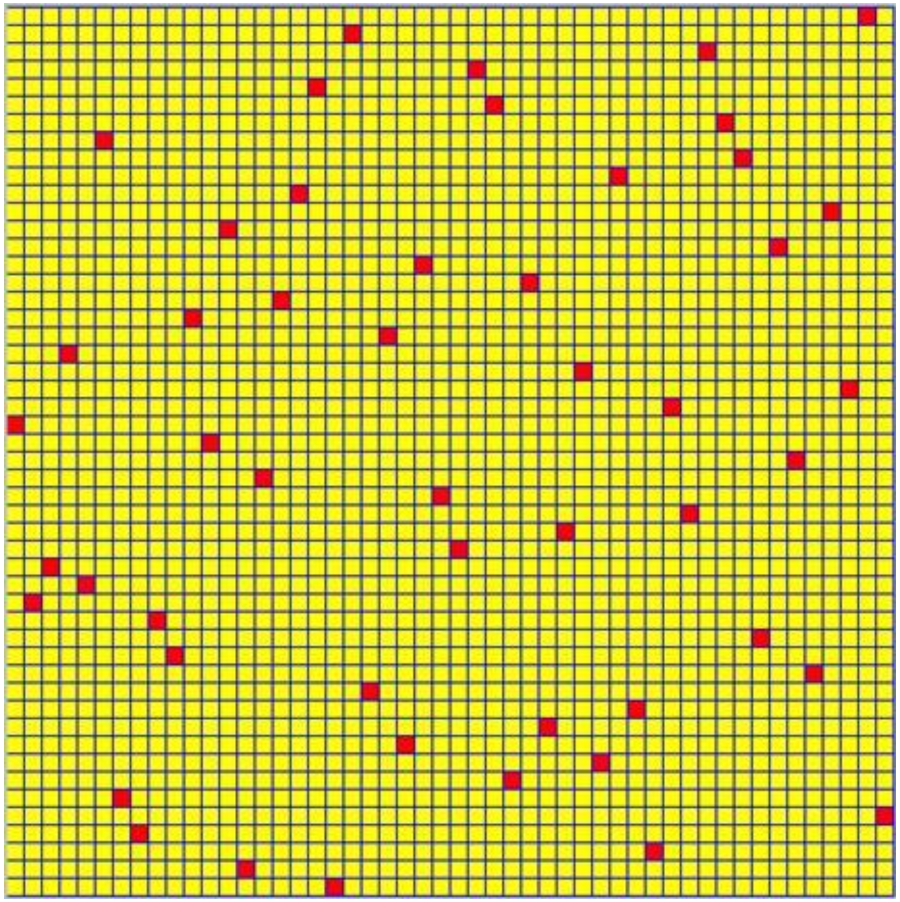
- Improve(**N**(S)) can be :
  1. First improvement
  2. Best improvement
  3. Intermediate option, e.g. "Best among *n*"

- observation: stops in local optimum

God save the Queens

- Place 8 queens on a chessboard so that no two queens attack each other
- Generalized to NxN chessboard

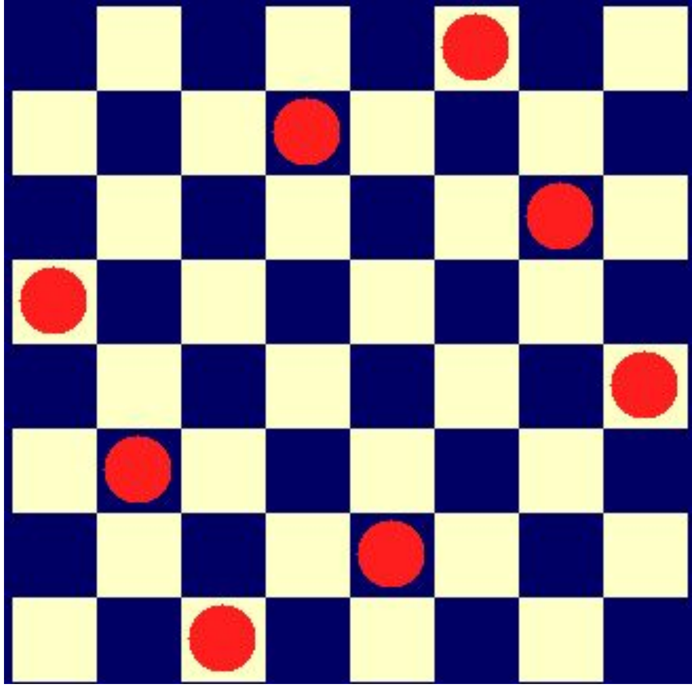


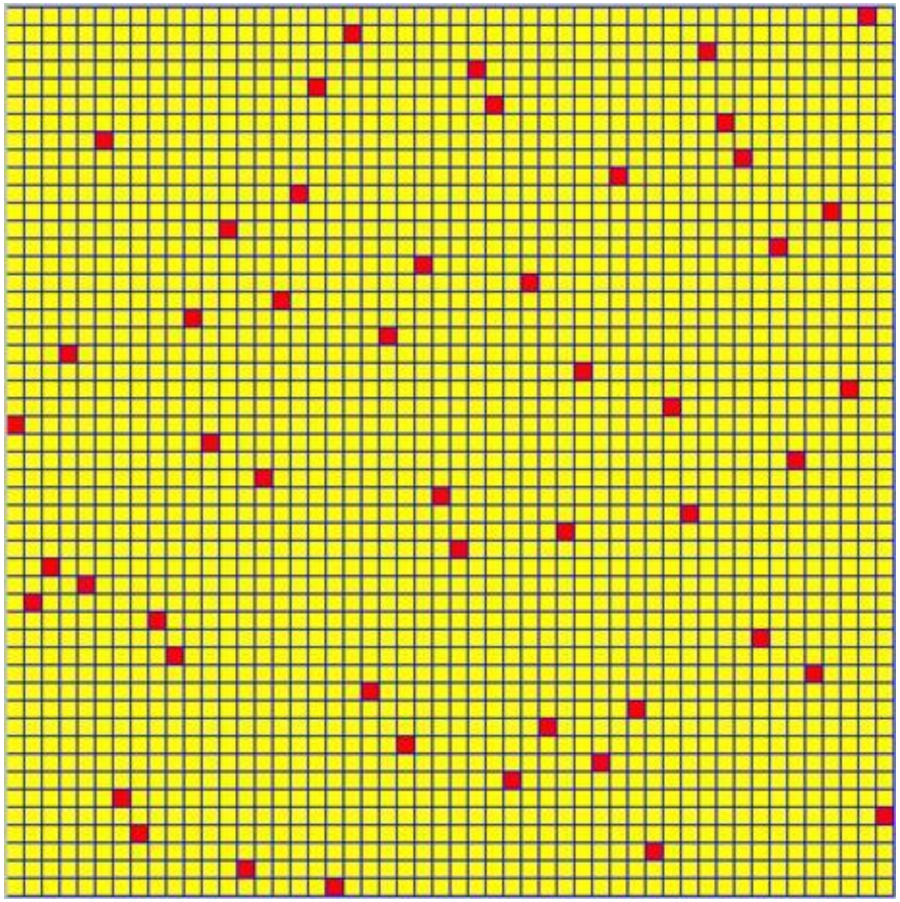


one solution for 50 x 50 chessboard

God save the Queens

- Place 8 queens on a chessboard so that no two queens attack each other
- Generalized to NxN chessboard





one solution for 50 x 50 chessboard

Can we solve this problem by Local Search ?

Local Search for N-Queens

- Configuration: (Q1,…,Qn)

  Qi = j means queen on row i and column j

- Objective function :

  minimize the number of attacks

  (= 2 x number of violated disequation constraints)

  - Neighborhood operator 1: change the position of one queen (i.e. change value of one Qi)
  - Size of neighborhood: *n*(n-1)*
  - At each step there is a quadratic number of neighbors to evaluate…

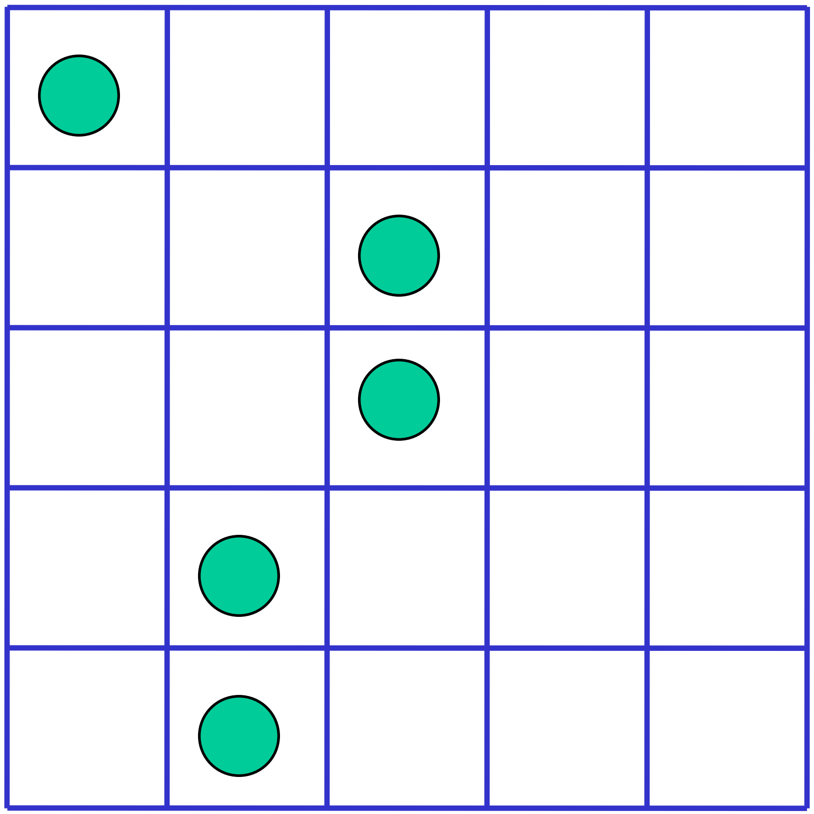
Local Search for N-Queens (2)

- Neighborhood operator 2:
  - compute for each queen the number of other queens attacking it
  - select the most conflicting queen
  - Consider only its alternative positions as neighbors

- Neighborhood of size *n-1*

- Each step of local search is thus faster
- But... is this a good heuristic?

Example



1

1

3

2

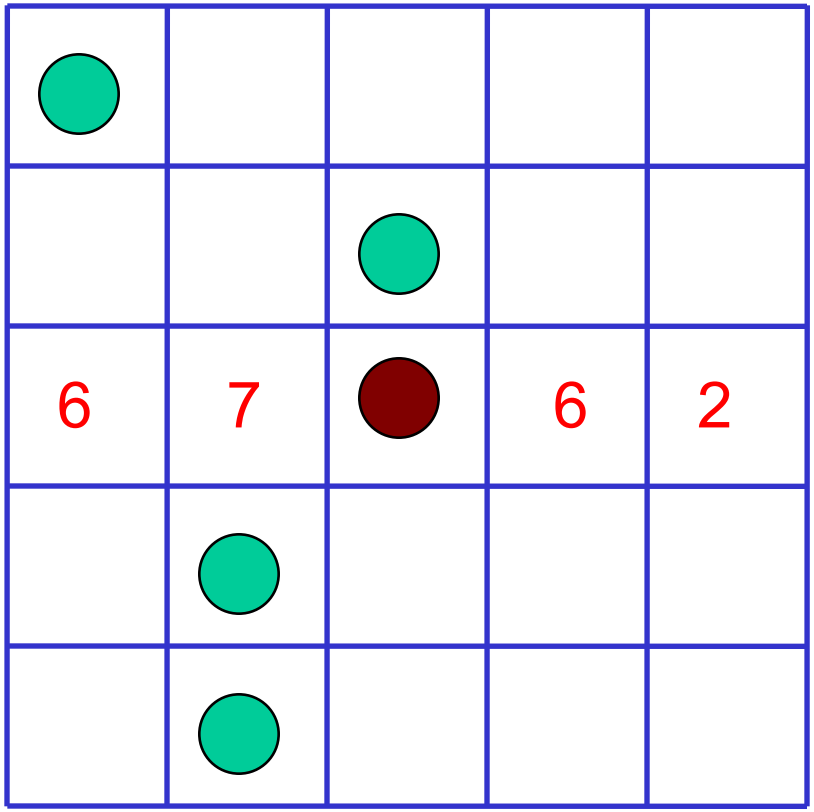
1

8

← cost for each queen

← Global cost

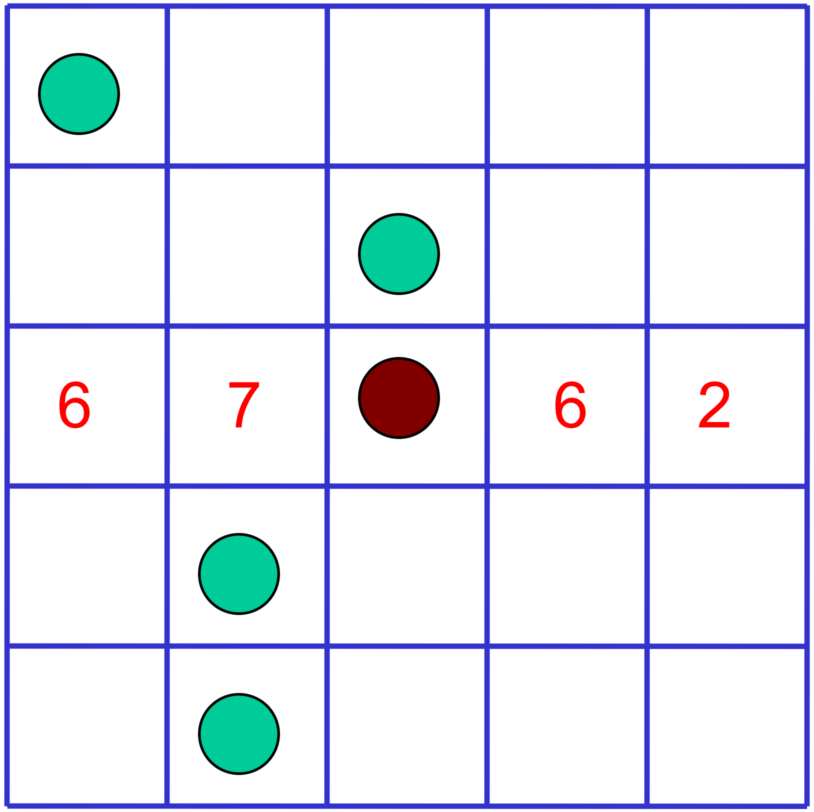
Example



Queen 3 will be selected

Alternative values gives
Other global costs

Example

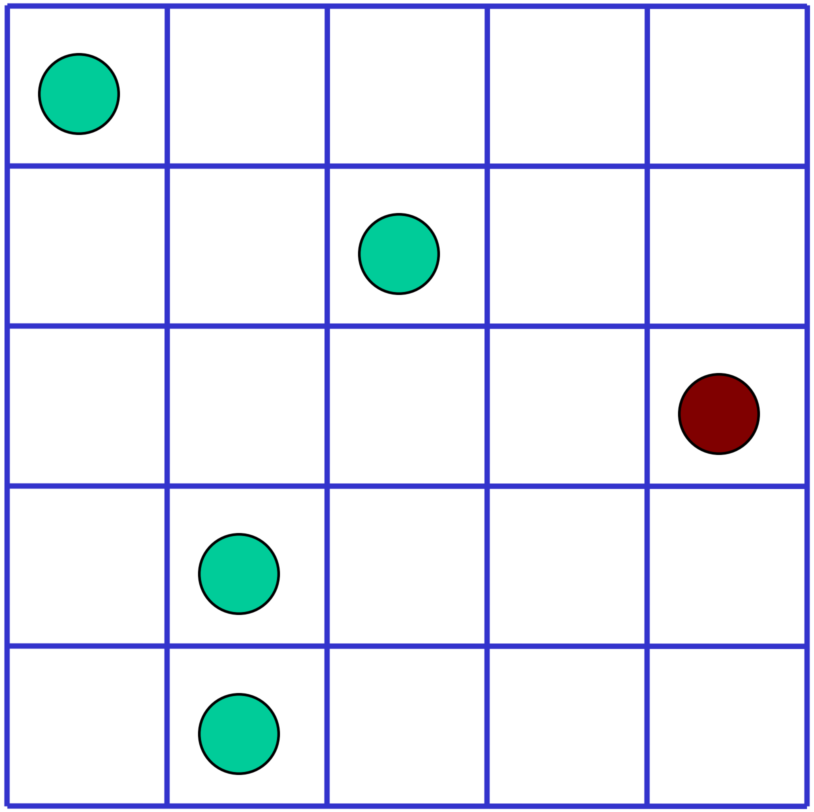


Queen 3 will be selected

Alternative values gives
Other global costs

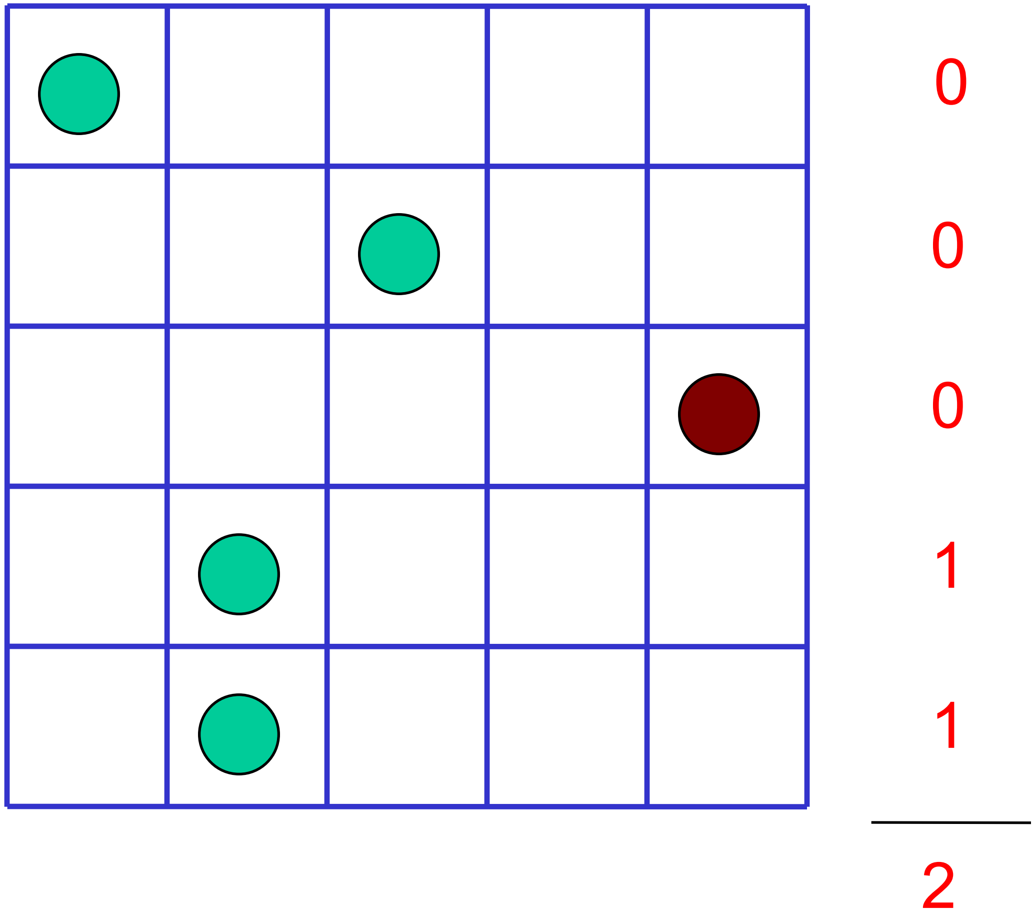
Move to row 5 is the best

Example

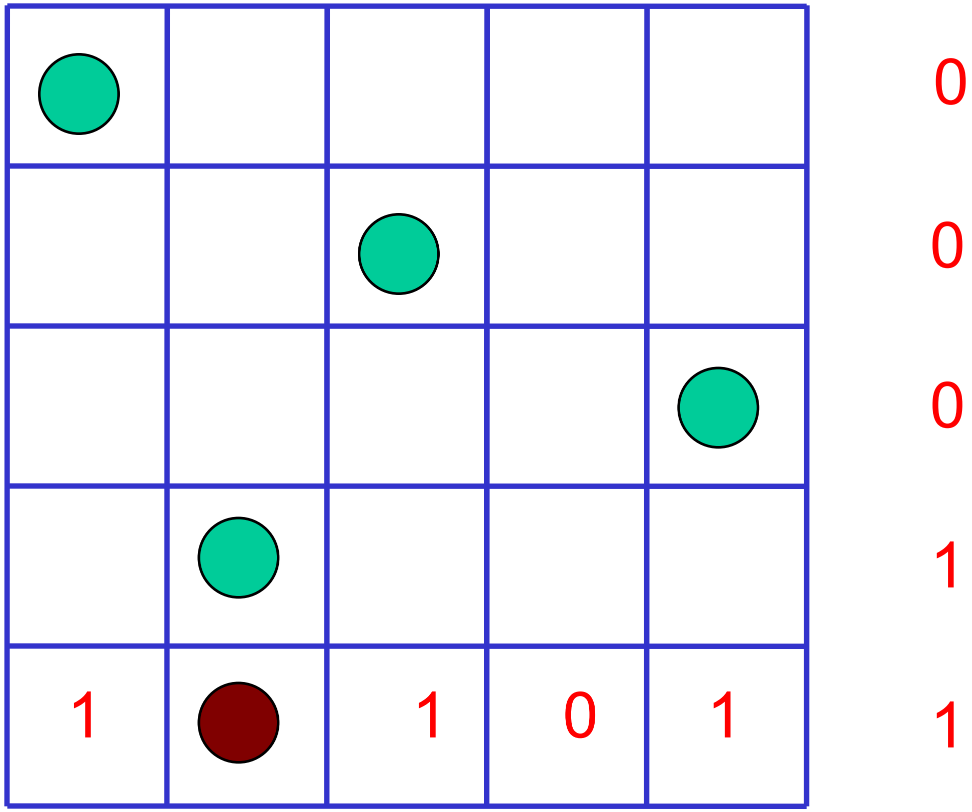


… and continue !

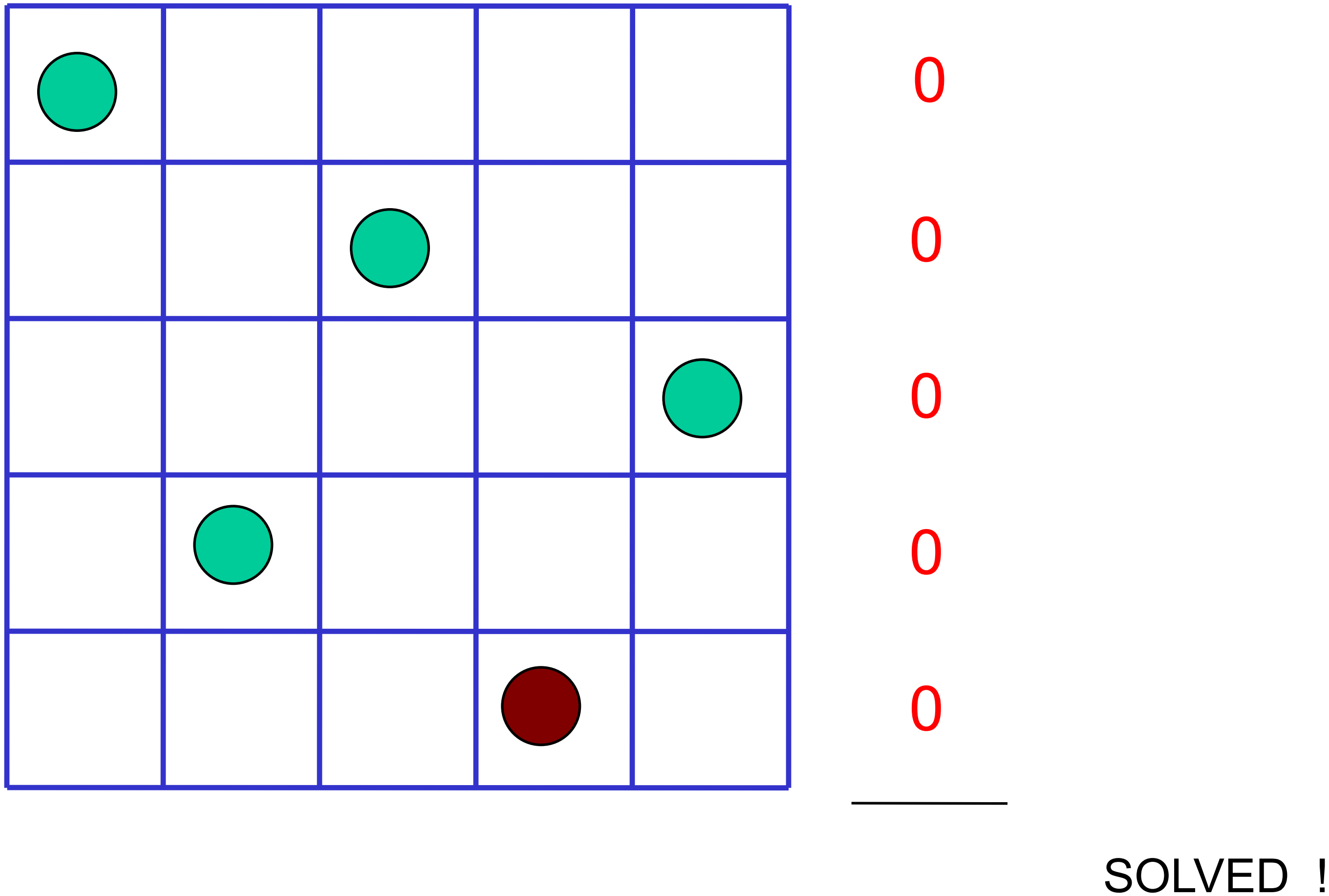
Example



Example



Example



Another Model for N-Queens LS

- Another model by considering:
  - permutation of {1,…,n} as configurations
  - objective function: minimize conflicting queens (diagonals only)
  - Neighborhood: all possible swaps of 2 values

Local Search / Neighborhood Search

- a Combinatorial Optimization Problem (COP)
- an initial solution (e.g. random)
- a defined search neighborhood (neighboring solutions)
- a move operator (e.g. flip a variable), going from one solution to a neighboring solution
- an evaluation function for moves (rating possibilities)
  - often *myopic*
- a neighborhood evaluation strategy (first, best, etc)
  - i.e. a move selection strategy
- a stopping criterion
  - e.g. local optimum

[From A. Løkketangen]

Advantages of Local Search

- For many problems, it is quite easy to design a local search
  - i.e., LS can be applied to almost any problem
- The idea of improving a solution by making small changes is easy to understand
- The use of neigborhoods sometimes makes the optimal solution seem ”close”, e.g.:
  - A knapsack has n items
  - The search space has 2^n members
  - From *any* solution, no more than n flips are required to reach an optimal solution!

[From A. Løkketangen]

Disadvantages of Local Search

- Some neighborhoods can become very large (time consuming to examine all the neighbors)

- The search stops when no improvement can be found: *local* optimum
- Restarting the search might help, but is often not very effective in itself – needs a strategy!

- How can we **avoid** getting stuck in a **local optimum**?

[From A. Løkketangen]

Metaheuristics (1)

- Concept introduced by Glover (1986)
- Generic heuristic solution approach designed to control and guide specific problem-oriented heuristics
- Often inspired by analogy with natural processes
- Rapid development over the last 15 years

[From A. Løkketangen]

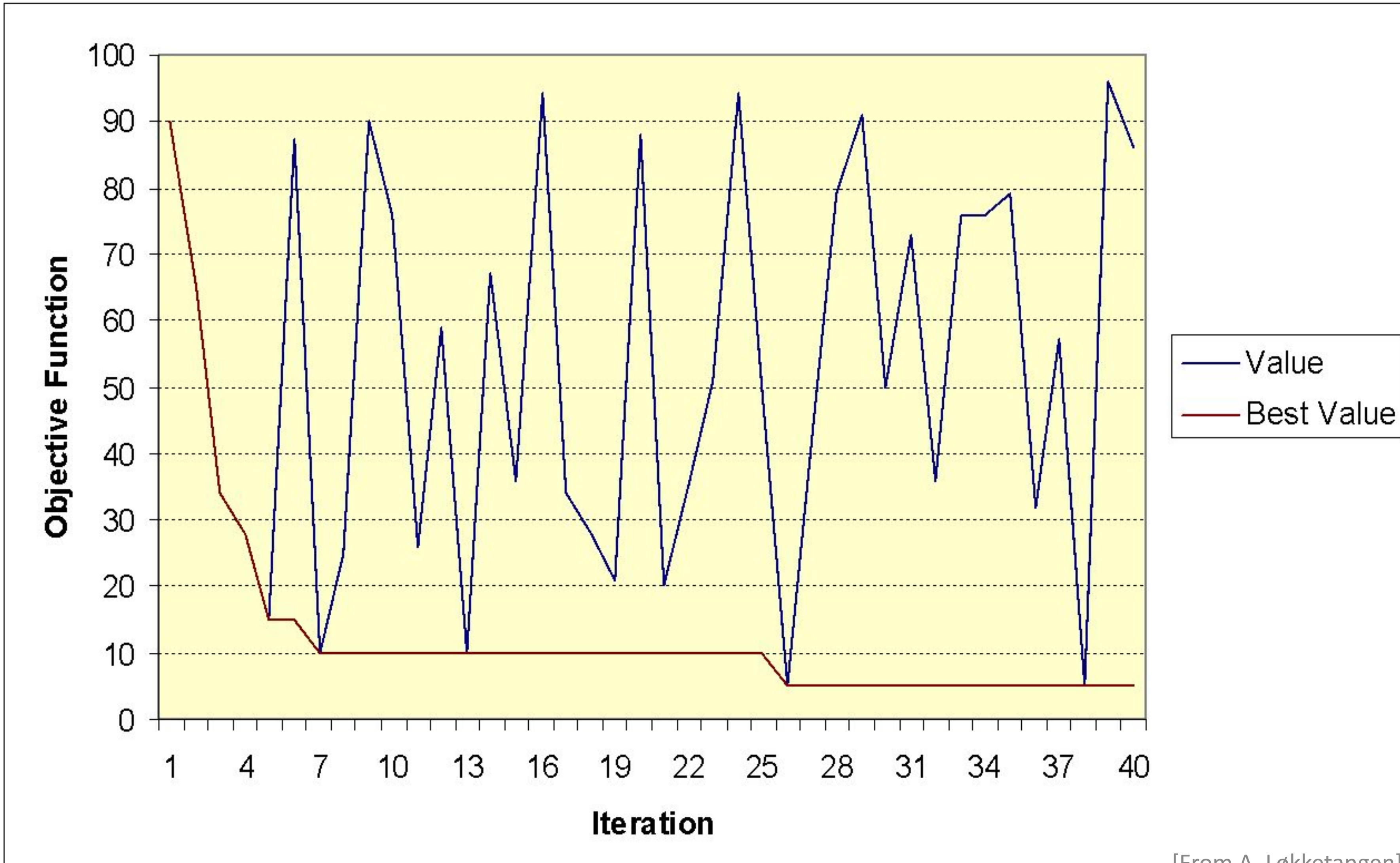
Metaheuristics (2)

- Main point is to escape local minima

- Many different ideas:
  - Random restart
  - Accept “bad moves” (i.e. worse w.r.t. objective function)
  - Use memory to record “bad solutions” to be avoided
  - Use a population of solutions
  - Mix any of the above

- This gives a lot of different methods!

Typical Search Trajectory



[From A. Løkketangen]

Some well-known Metaheuristics

- Simulated Annealing (SA)
- Tabu Search (TS)
- Genetic Algorithms (GA)
- Scatter Search (SS)

[From A. Løkketangen]

Other Metaheuristics

- Iterative Local Search (ILS)
- Guided Local Search (GLS)
- Adaptive Memory Procedures (AMP)
- Variable Neighborhood Search (VNS)
- Threshold Acceptance methods (TA)
- Ant Colony Optimization (ACO)
- Greedy Randomized Adaptive Search Procedure (GRASP)
- Evolutionary Algorithms (EA)
- Memetic Algorithms (MA)
- And many others:
  – Particle Swarm, The Harmony Method, The Great Deluge Method, Shuffled Leaping-Frog Algorithm, Squeaky Wheel Optimization, Artificial Bee Colony, Cukoo Search, Firefly Optimization …

[From A. Løkketangen]

Metaheuristic Classification

- x/y/z Classification
  - x = A (adaptive memory) or M (memoryless)
  - y = N (systematic neighborhood search)
    or S (random sampling)
  - z = 1 (one current solution)
    or P (population of solutions)
- Examples
  - Tabu search (A/N/1)
  - Simulated Annealing (M/S/1)
  - Genetic Algorithms (M/S/P)
  - Scatter Search (M/N/P)

[From A. Løkketangen]

Single Solution vs Population-based

- Single-solution based algorithms
  - Hill Climbing
  - Simulated Annealing
  - Tabu Search

- Population based algorithms
  - Genetic Algorithm
  - Ant Colony Optimization
  - Particle Swarm Optimization

Nature inspired vs Non-nature inspired

- Genetic Algorithms
- Swarm Intelligence
  - Ant Colony Optimization
  - Particle Swarm Optimization
- Also
  - Bee Colony Optimization
  - Firefly Optimization
  - Cuckoo Search