Heuristic search

Problem solving is hunting. It is savage pleasure and we are born to it.

- Thomas Harris

How to find solutions?

- **Exact methods**
  - Analytical approach
  - Explicit enumeration
  - Implicit enumeration

- **Approximation Algorithms**
  - provide a theoretical bound on quality of solution
  - an algorithm is an ϵ-approximation algorithm for a minimization problem with optimal cost z^*, if the algorithm runs in polynomial time and returns a feasible solutions with cost z_h:

$$z_h \leq (1 + \epsilon)\, z^*$$

- **Heuristic Algorithms**

Why don’t we always use exact methods?

If a heuristic does not guarantee a good solution, why not use an (exact) algorithm that does?

- The running time of the algorithm
- The link between the real-world problem and the formal problem is already tenuous at best

P vs NP

- **However, for some optimization problems we know of no polynomial time algorithm**
  - Unless P=NP there are none!
- **Sometimes, finding the optimal solution reduces to examining all the possible solutions (i.e., the entire solution space)**
  - Some algorithms do implicit enumeration of the solution space (but this sometimes reduces to examining all solutions)

Problem → Model (approximate) → Solution (precise)

or

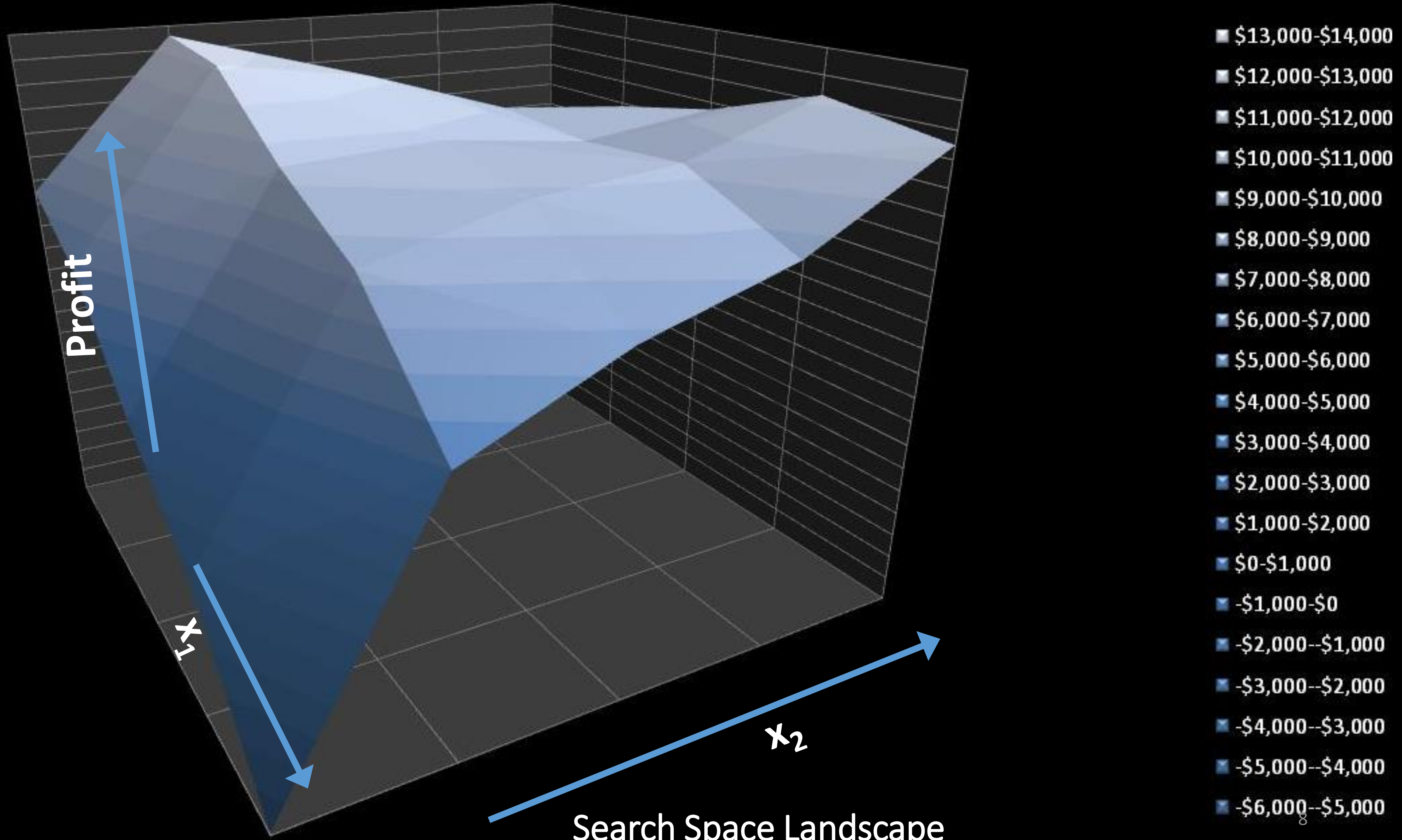
Problem → Model (precise) → Solution (approximate)

Heuristics

- *heuriskein* (meaning ‘to find’)
- Provide a shortcut to solve difficult problems.
- Used under limited time and/or information to make a decision.
- Problem-dependent techniques, i.e., adapted to the problem at hand to take advantage of the particularities of the problem.

Terminology

- The set of all candidate solutions is called a solution space or search space
- Each element in the search space represents one candidate solution.
- Every point x in the search space has a “goodness” value based on the problem specific evaluation function g(x)
- The set of solutions and their objective values form locations and elevation in the search space landscape



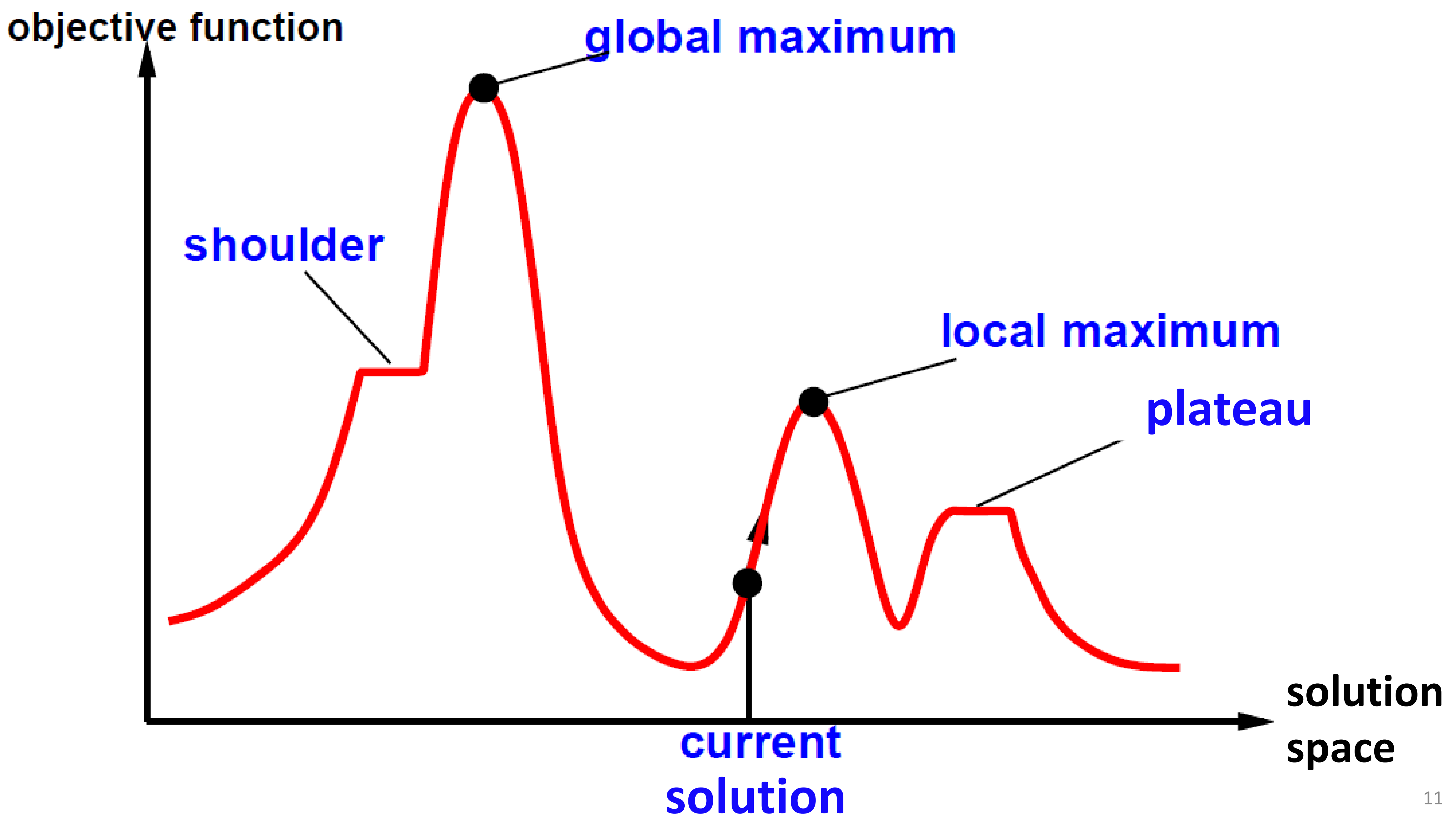
Search Space Landscape

The problem is that the landscape can be very complicated.

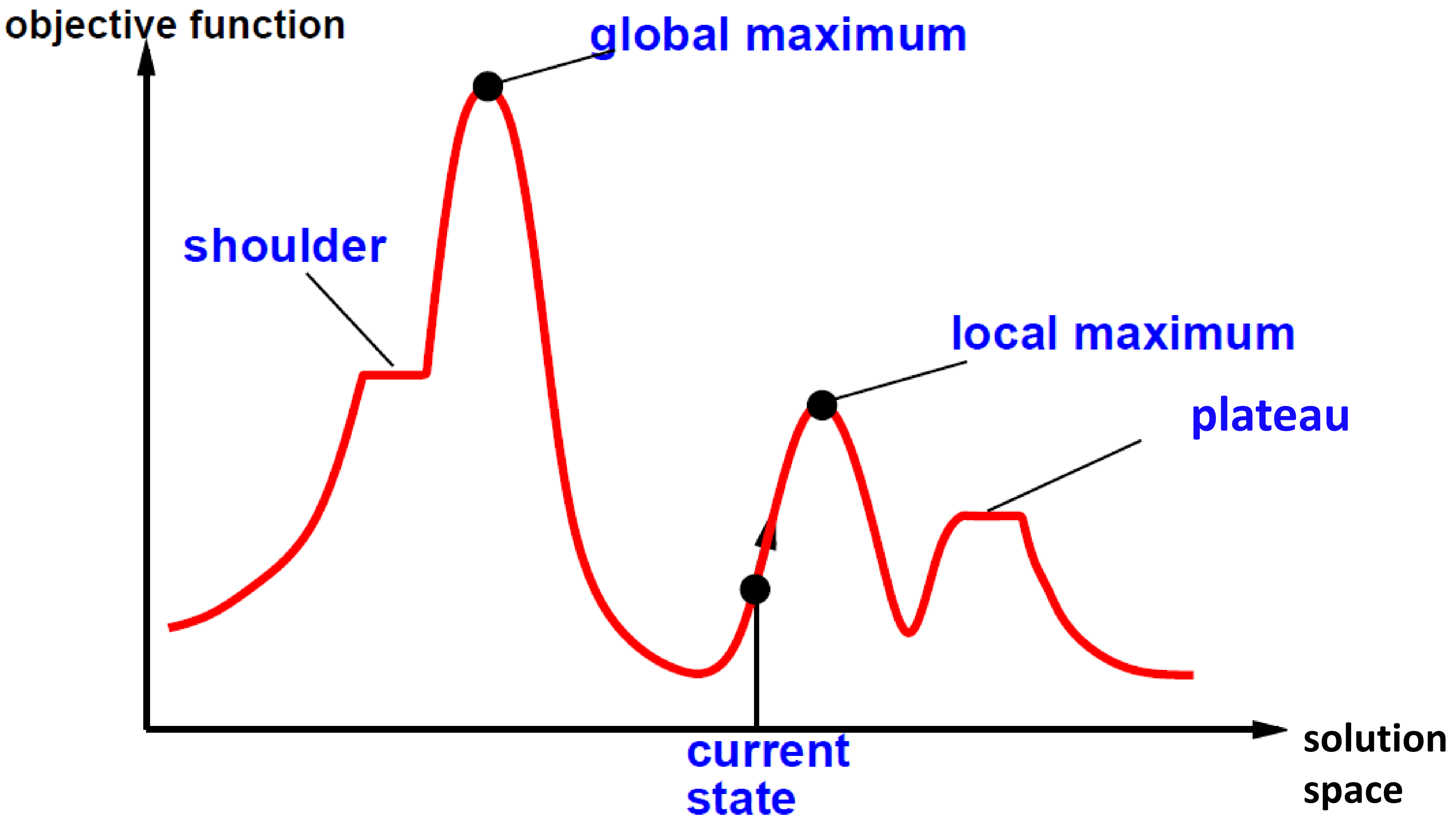
For some problems, it is possible to not even have any idea where to start looking!!

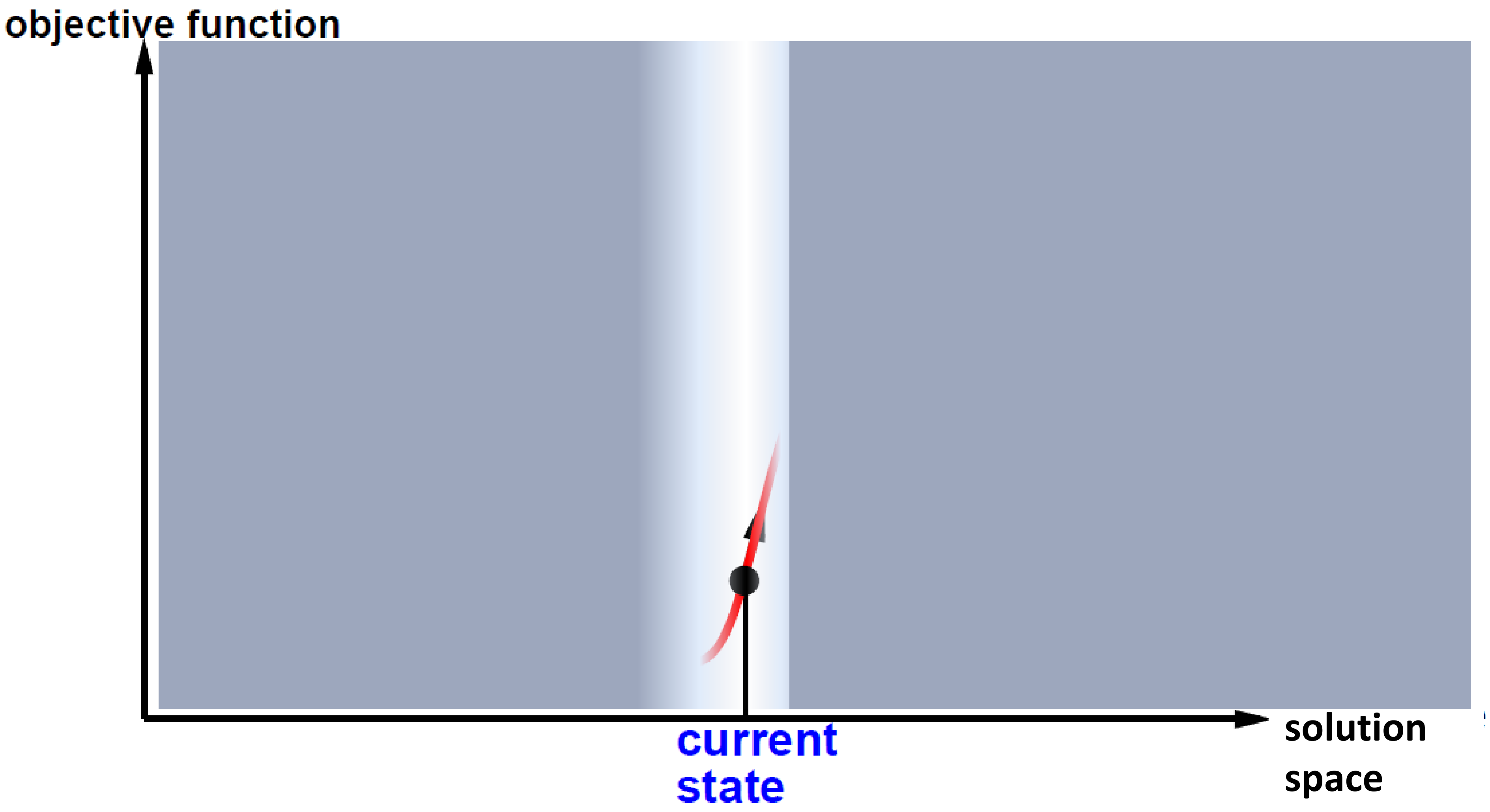
Search Space Landscape

- A complex Search Space may have many hills, valleys, ridgelines, shoulders, and plateaus – just like a mountain range.
- One common problem is to find local maxima or local minima and mistake the solution for the global maximum or global minimum.


objective function
global maximum
shoulder
local maximum
plateau
current
solution
solution
space


Menu (F10)
Message Log (F11)
186
0
1
190 / 190
90 / 90
Our base is under attack!
2:06:54
[Allies] GamingJake: Its so foggy
Marine Hero
Level 10
Damage: 6 (+1.8)
Armor: 0
Kills: 589
Light - Biological - Heroic
Agility: 10
Intellect: 10
Might: 10
Reflexes: 10
Wisdom: 10


objective function
global maximum
shoulder
local maximum
plateau
current
state
solution
space


objective function
current
state
solution
space

Some classes of heuristic search

$$\nabla f(x) \equiv \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

- **Generate and Test** – guessing
- **Gradient Based** – requires differentiable functic
  - gradient vector: vector of partial derivatives w.r.t. each independent variable
- **Neighborhood-based**
- **Population-based**

Note: there are two classes of heuristics
search and *construction* heuristics

neighborhood-based terminology

Search: constructing or improving solutions to obtain the optimum or near-optimum

Encoding: method to represent solutions

Evaluation: To compute the solutions’ feasibility and objective function value

Neighborhood: “nearby” solutions

Move: Transforming current solution to another (usually a neighbor solution)

Local search: based on greedy heuristic (local optimizers) formulation

Optimization

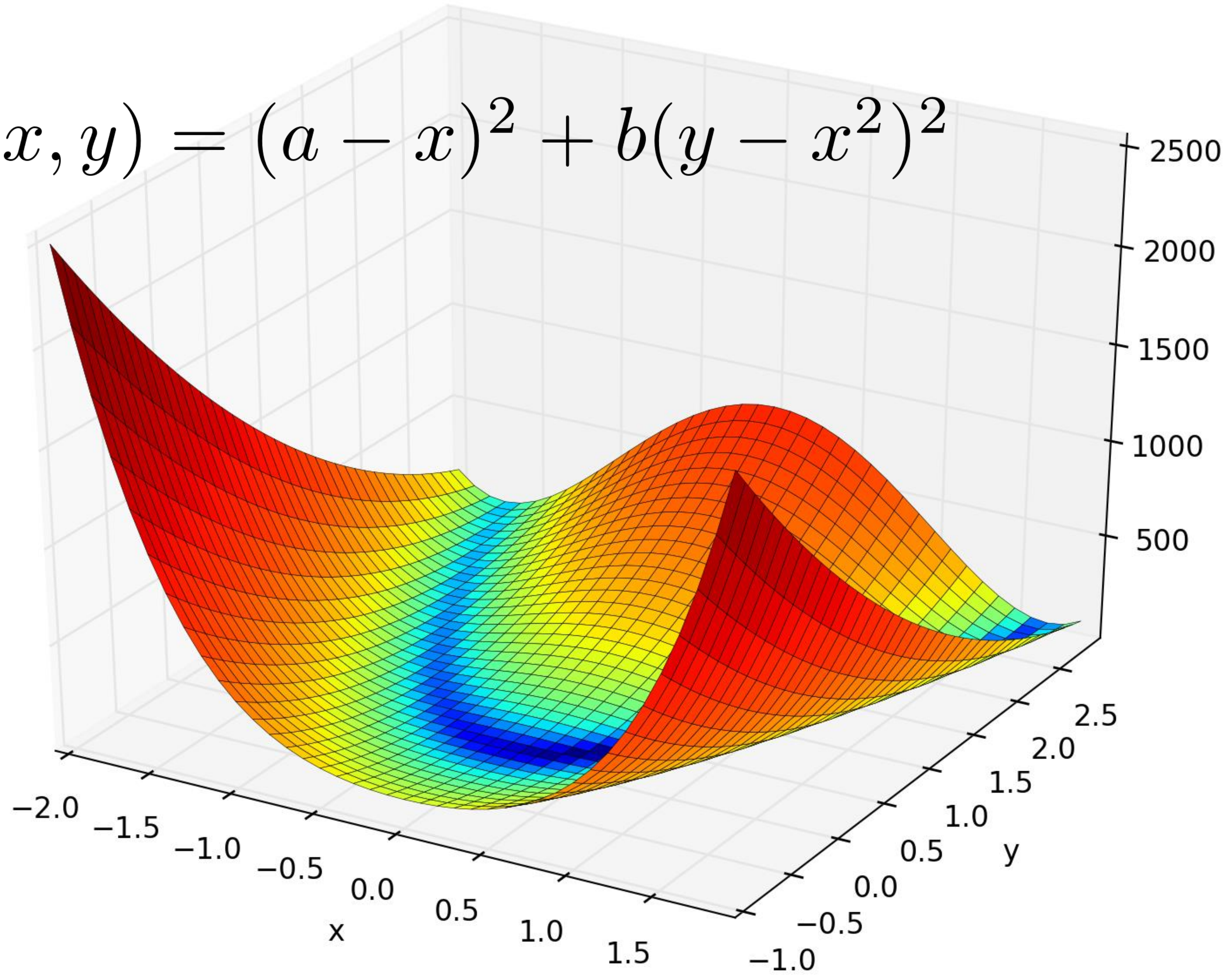
- Decisions
- Objective
- Constraints

Heuristic

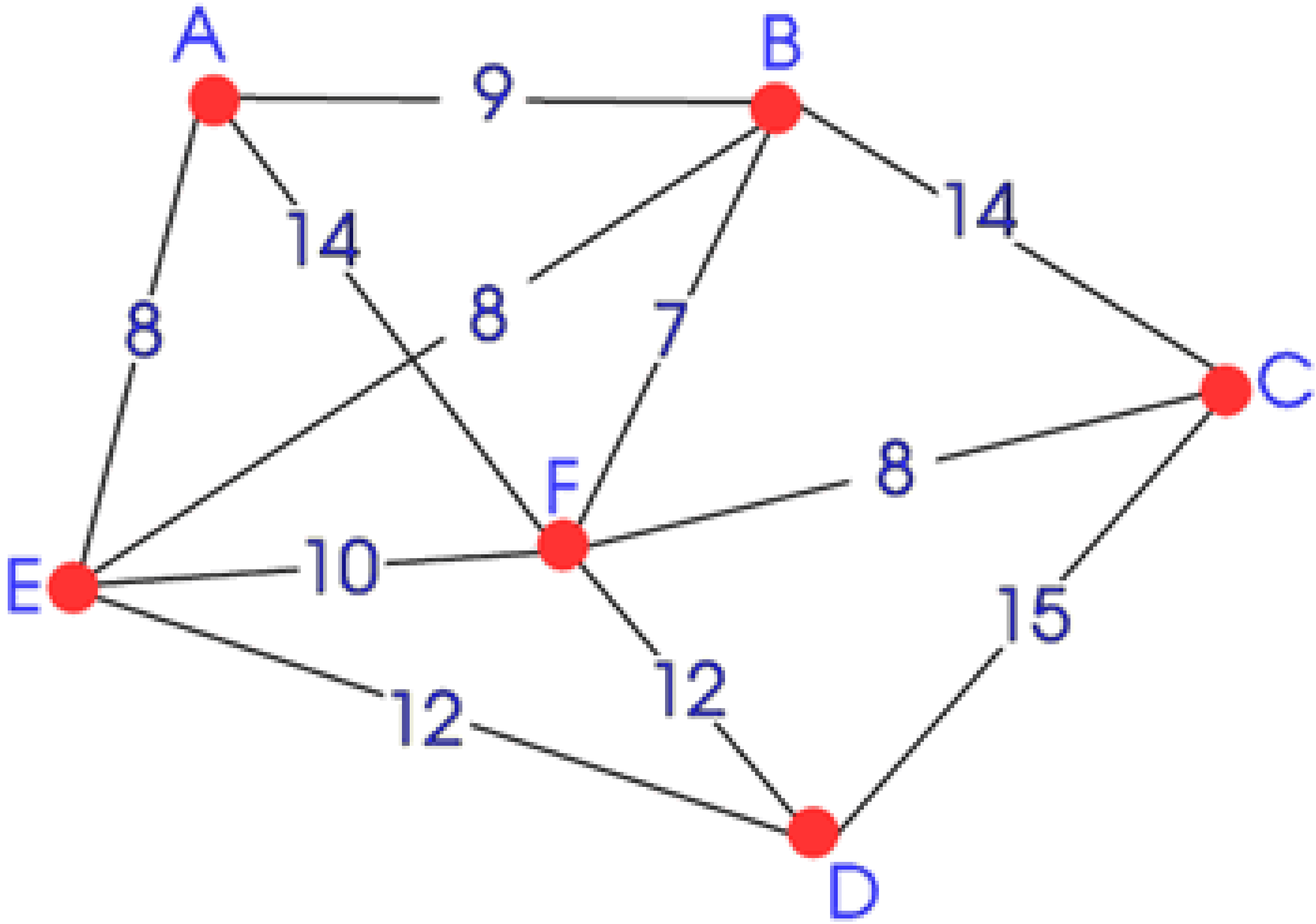
- Decisions
  - Encoding
- ~~Objective~~
- Evaluation Function
- Constraints
  - Constraint Handling
- Neighborhood and Moves
- Parameter Tuning
- Stopping Criterion

Encoding

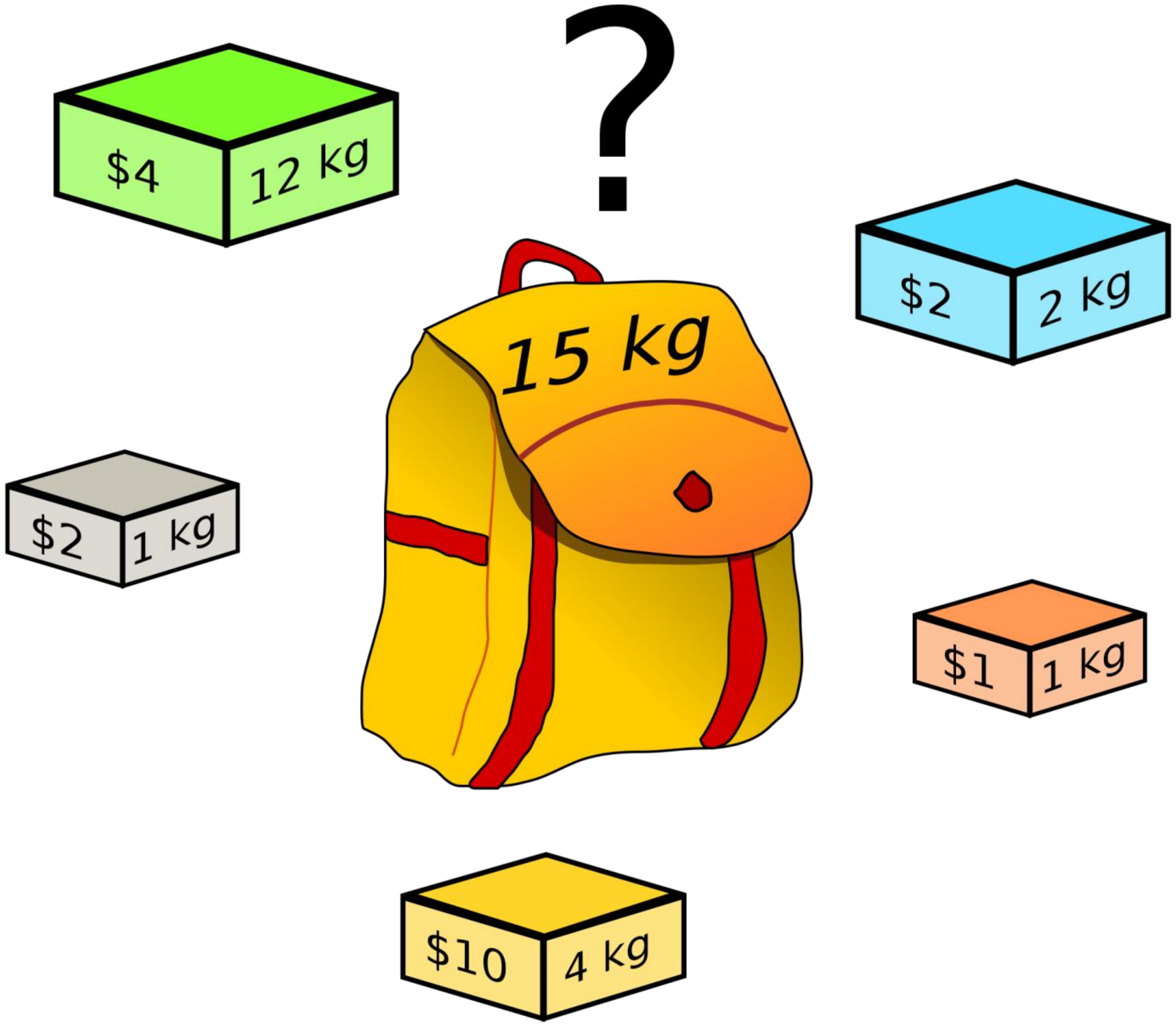
$$f(x,y) = (a-x)^2 + b(y-x^2)^2$$



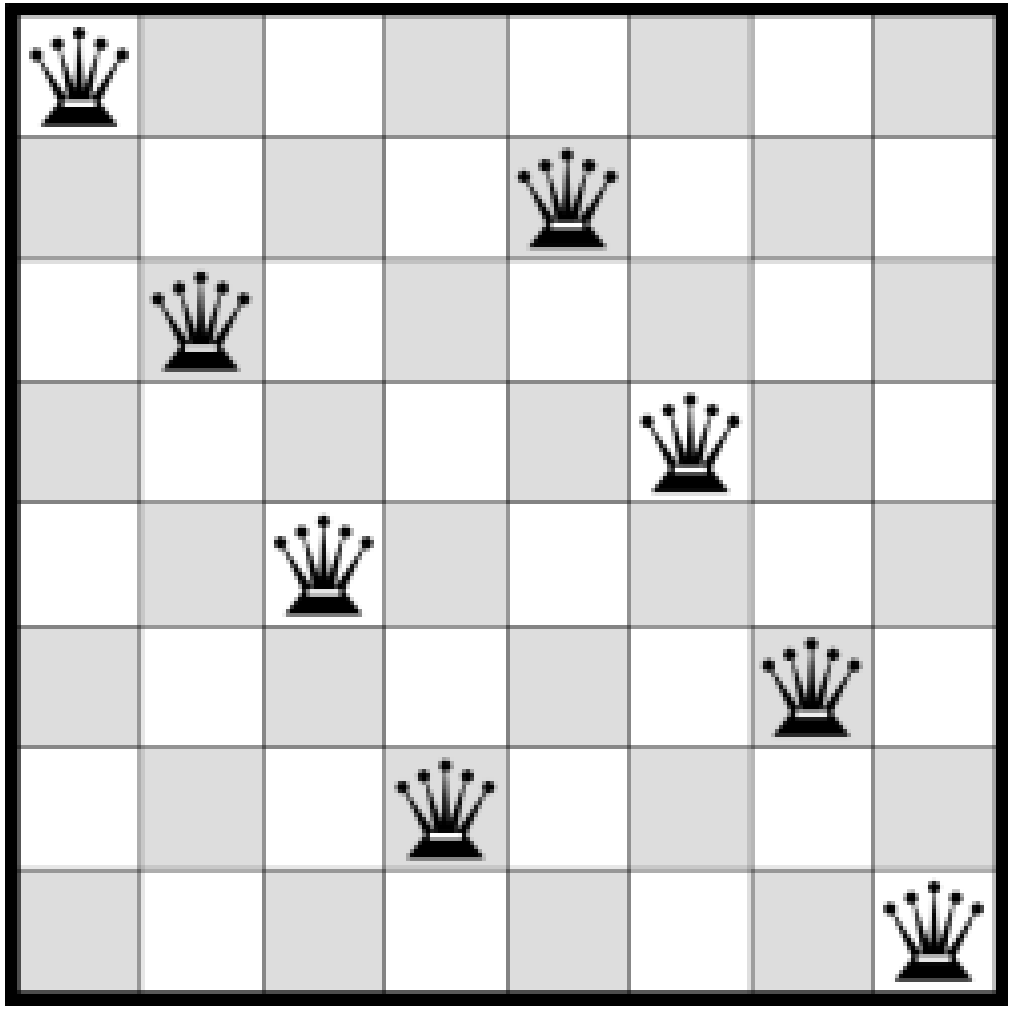
Encoding



Encoding

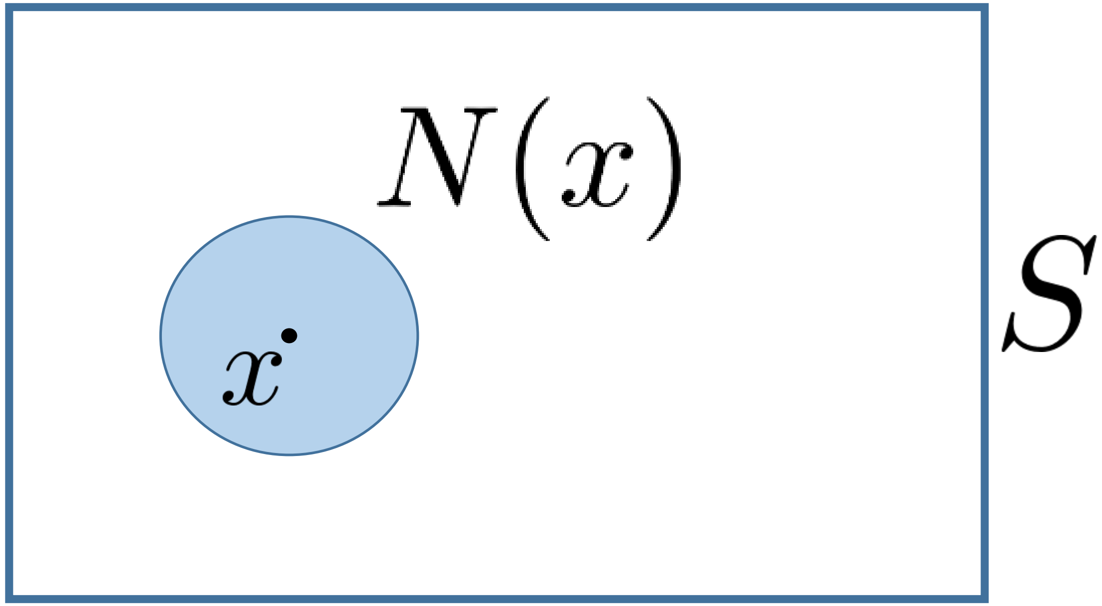


The n-queens problem is to place *n* queens on an *n*×*n* chessboard so that no two queens threaten each other.



Neighborhoods

Consider a region of the search space that's "near" some particular point:

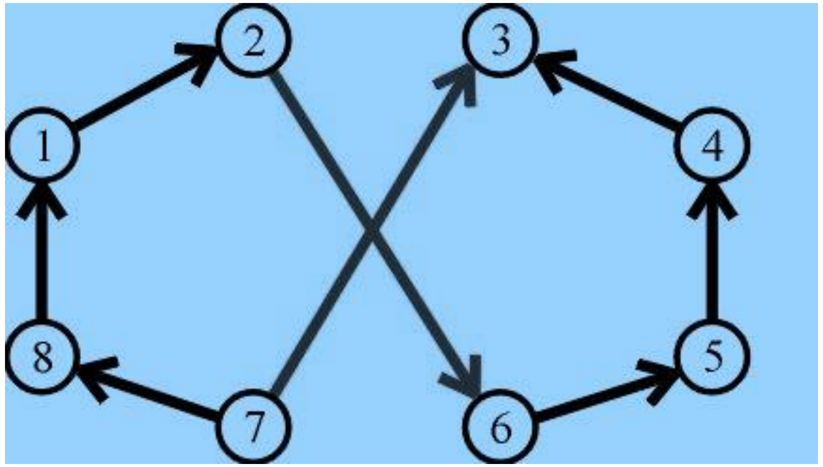


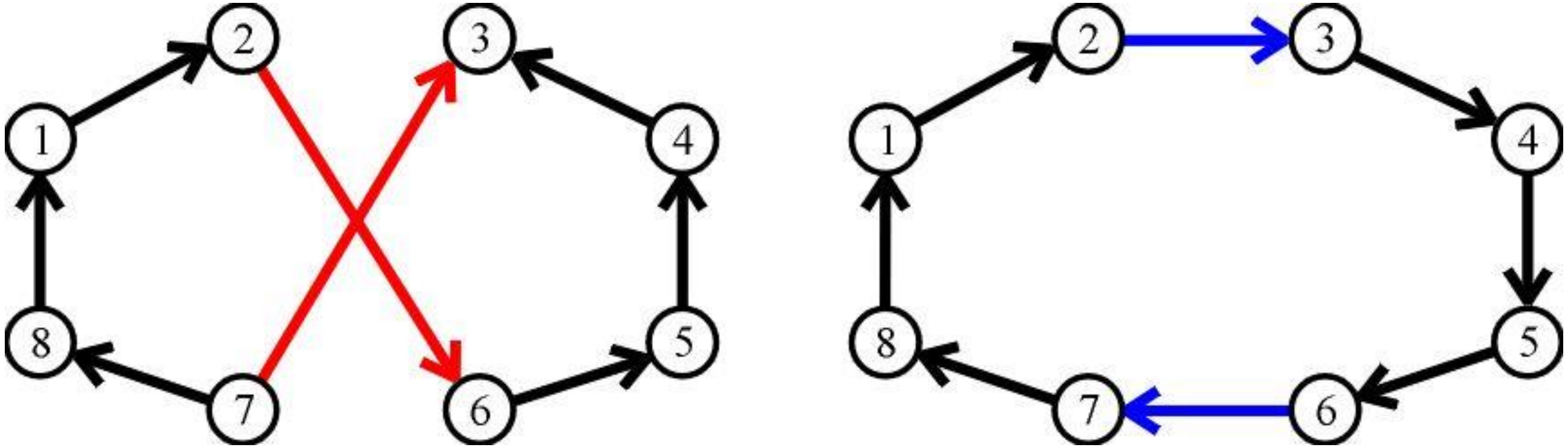
Neighborhood *N(x)* of *x* is a set of all points of search space *S* that are *close* in some measurable sense to the given point *x*.

neighborhood operator

- Neighborhoods may be defined by *distance measures* (e.g., Euclidean, Hamming, Jaccard)
- Neighborhoods may be defined by an *operation* on a solution
  - Often simple operations
    - Remove an element
    - Add an element element
    - Interchange two or more elements of a solution

Example Neighborhood for TSP: 2-opt





https://www.youtube.com/watch?v=UGGPZnAUjPU

Example Neighborhood for TSP: 2-opt

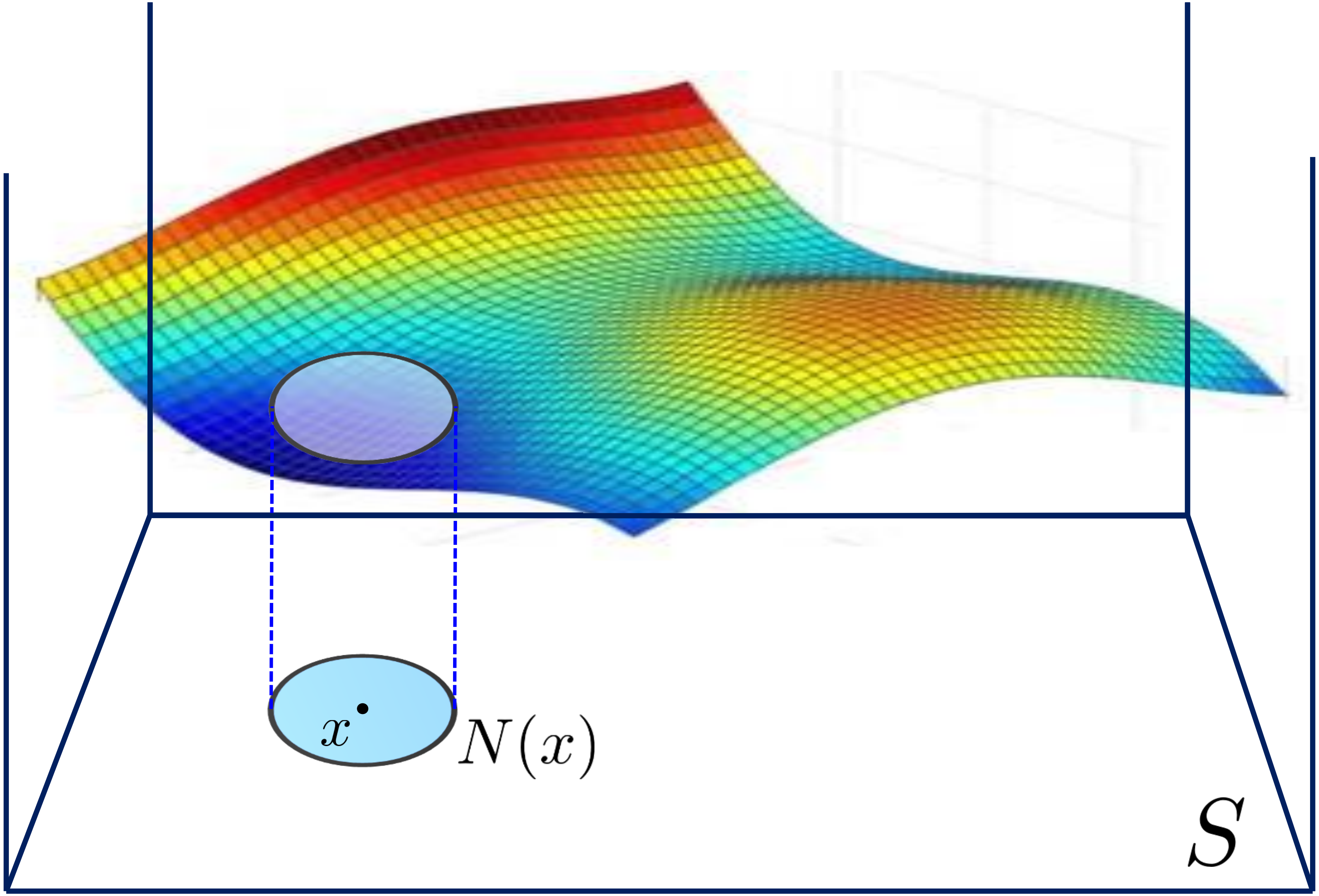
What is the size of the 2-opt neighborhood for the TSP?

Encoding is a how we represent a solution; it helps us to think about "neighborhoods" of solutions.

A neighborhood is all solutions that are somehow "close" to the solution.

We must determine a way to evaluate a candidate solution.

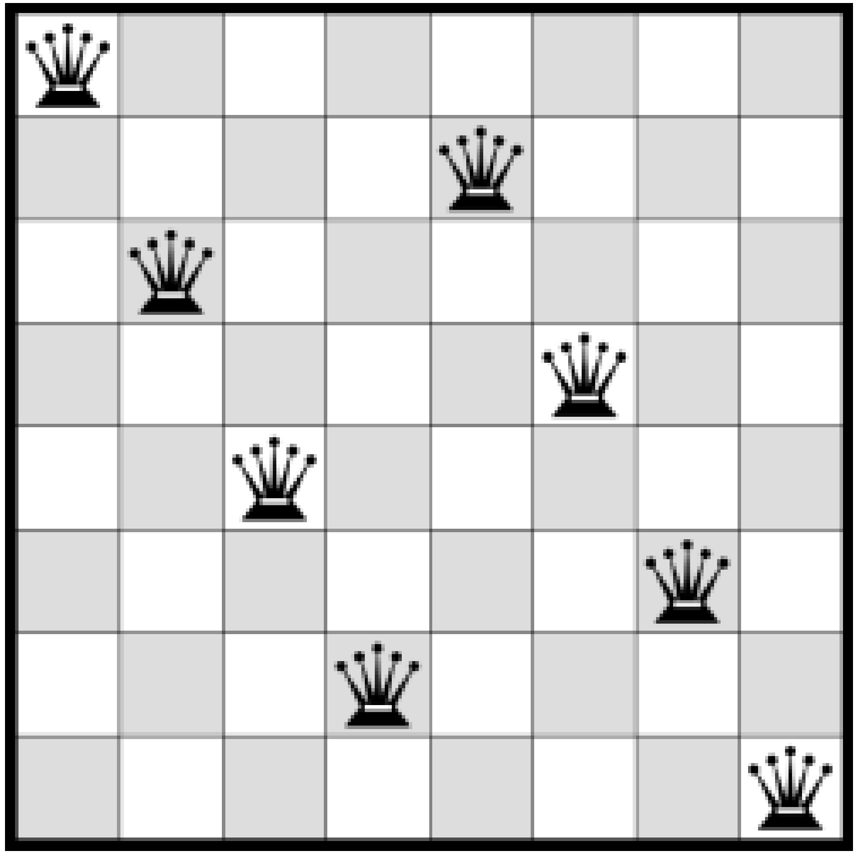
A good neighborhood definition is fundamentally important!


x
$N(x)$
S

The n-queens problem is to place *n* queens on an *n*×*n* chessboard so that no two queens threaten each other.

Describe a N(x) for the n-queens problem.

What is the size of N(x)?



How might you evaluate a solution of the 8-queens problem?

8-queens Problem

- **Encoding**: solution x is a 8d vector of integers representing queens positions in column 1-8 respectively

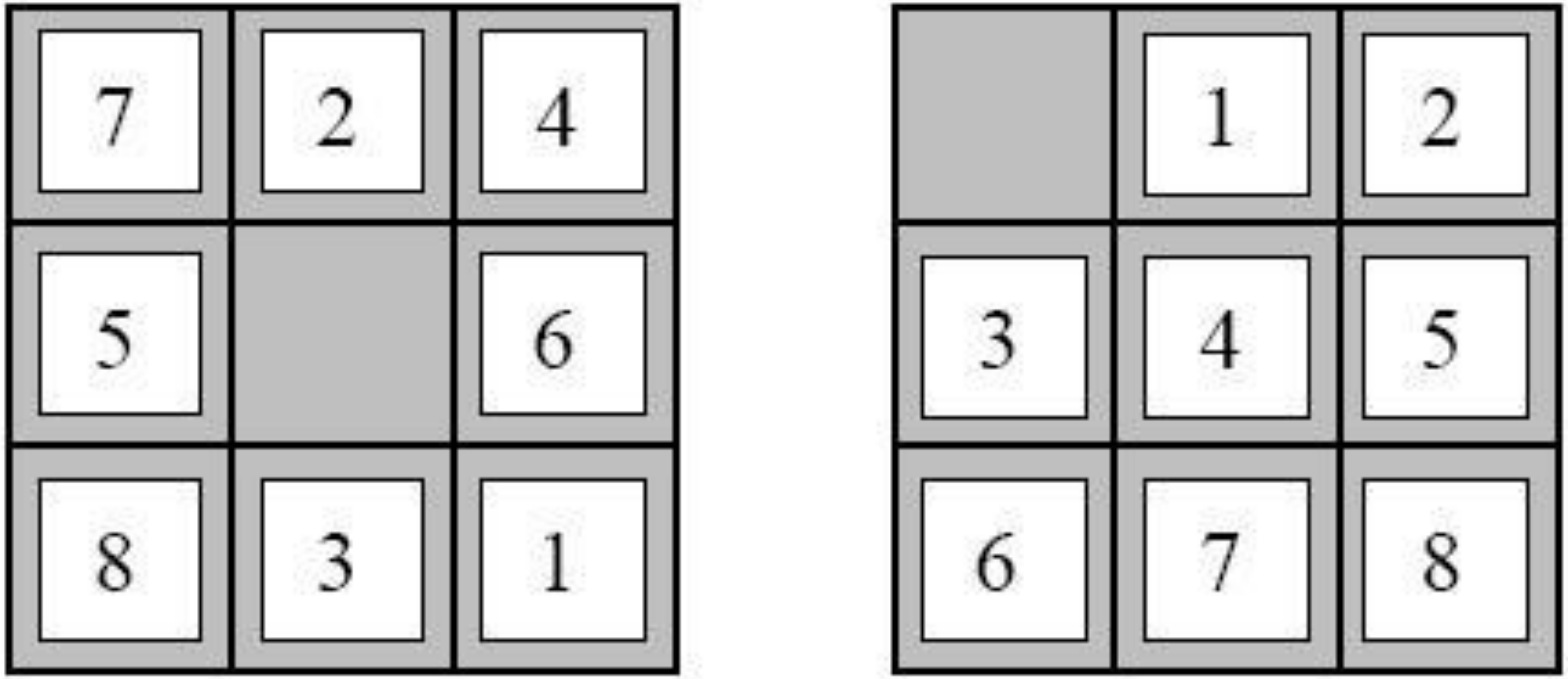
- **Neighborhood**: all solutions generated by moving a single queen to another square in the same column.

  Size of N(x): 8 * 7 = 56 solutions

- **Evaluation function**: f(x) = number of queens that attack each other in solution s.

8-puzzle

http://mypuzzle.org/sliding



Start State Goal State

Hill climbing

Like climbing Everest in thick fog with amnesia…

Hill Climbing (or descending)

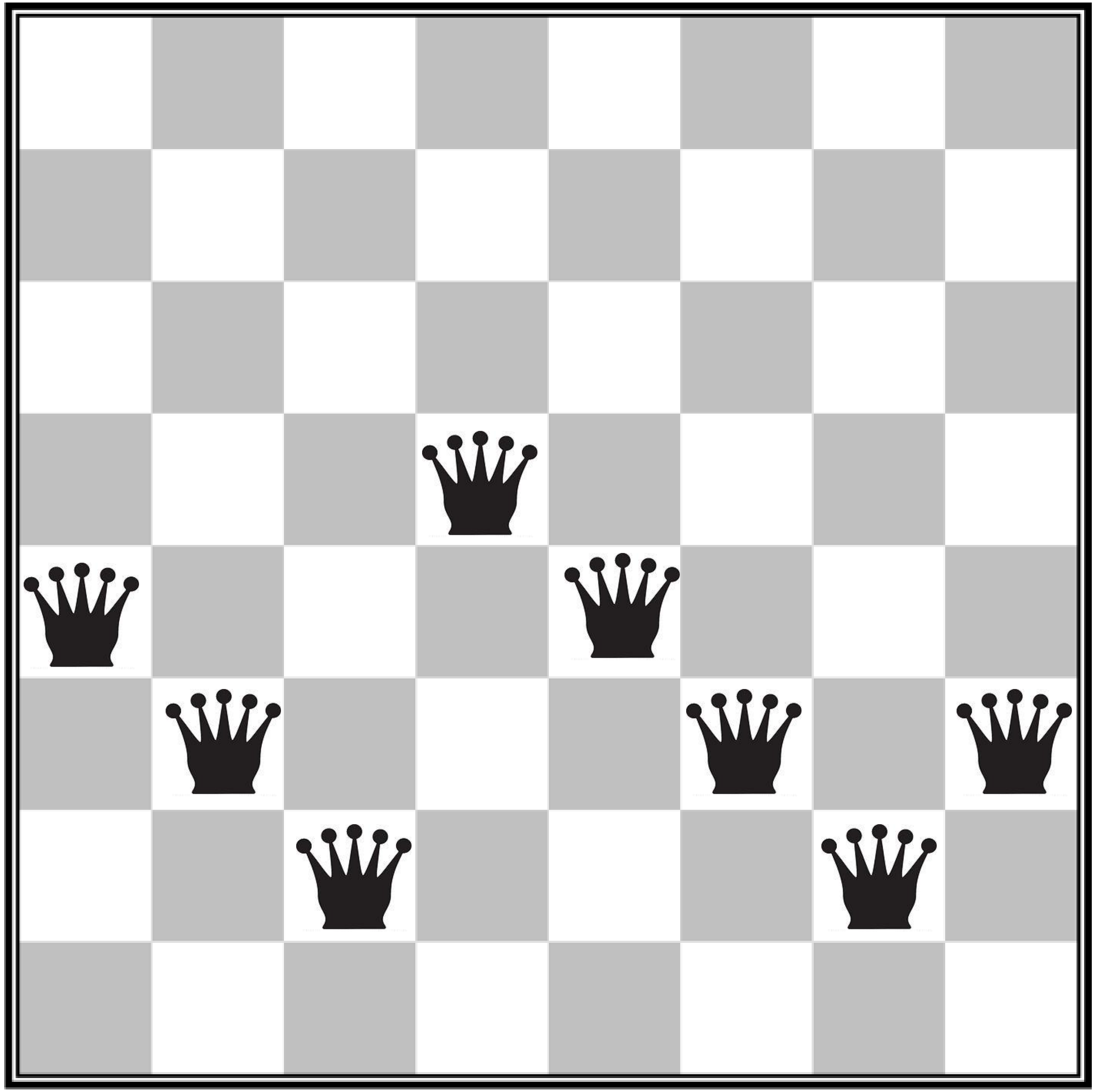
Very simple idea: Start from some solution $s \in S$, move to a neighbor $t \in S$ with better score. Repeat.

Designing the neighborhood is critical. This is the real ingenuity!

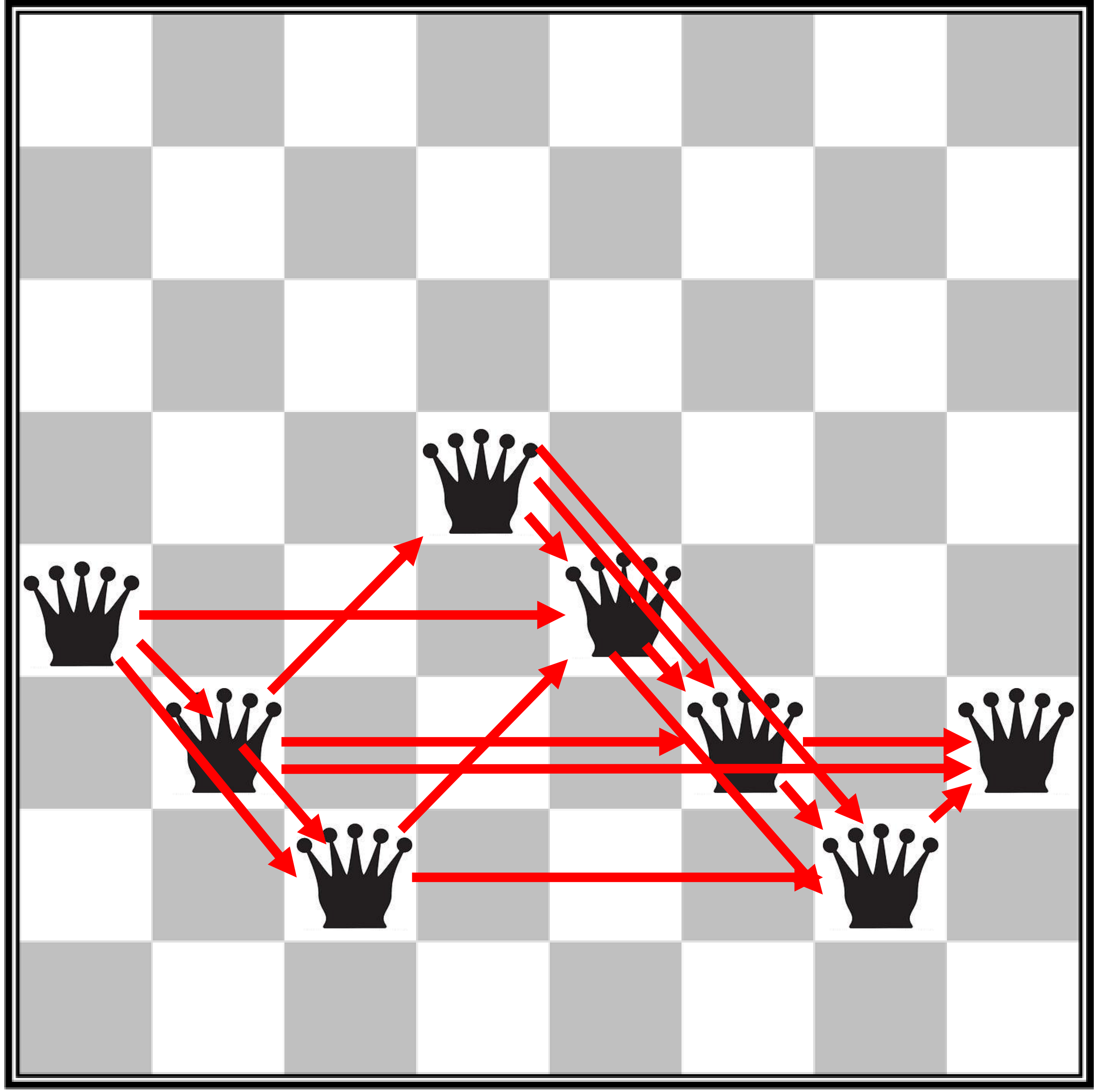
- Neighborhood must be small enough to be efficient.
- Problems tend to have structures.

Question: Pick which neighbor?

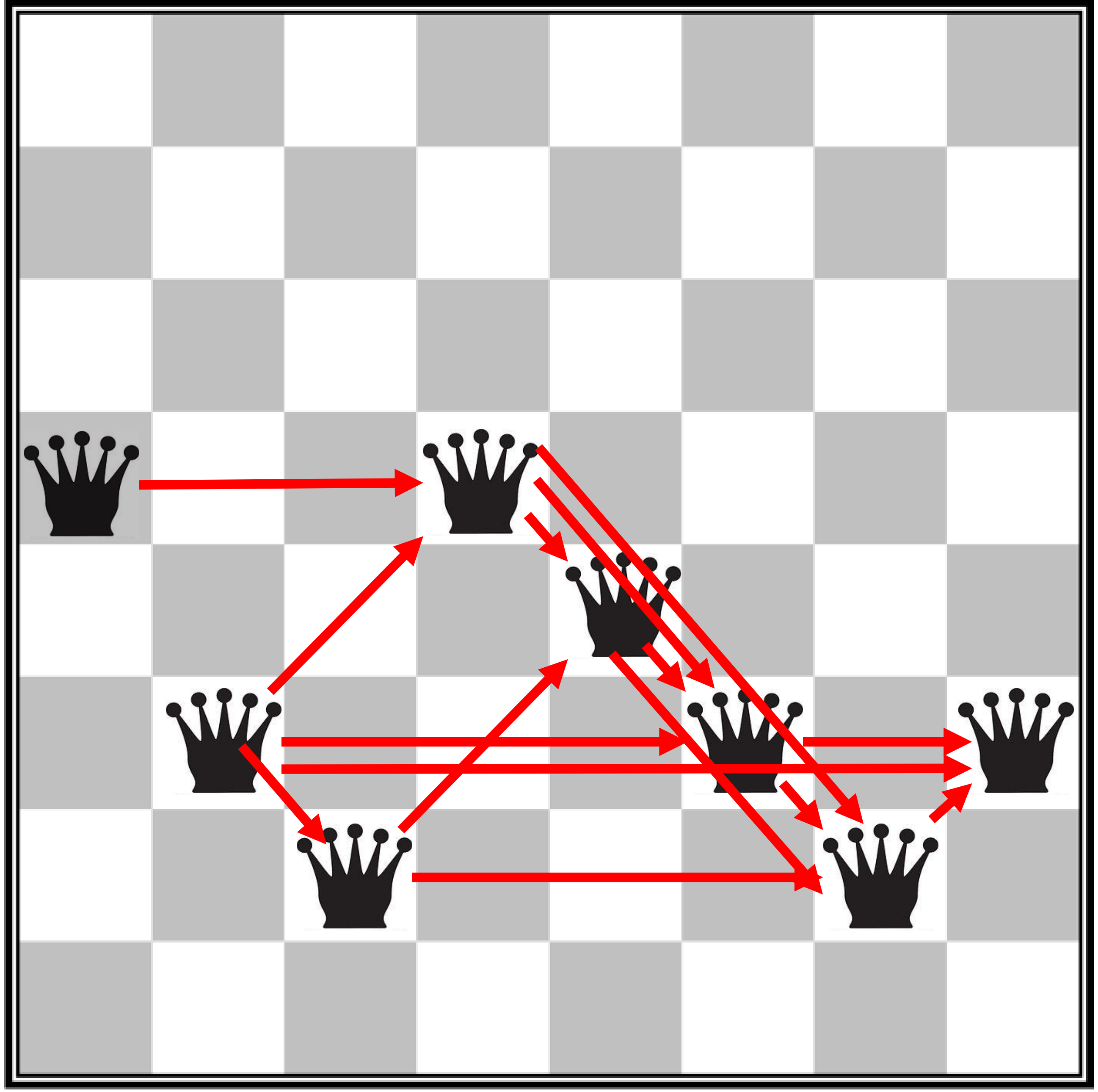
Question: What if no neighbor is better than current state?



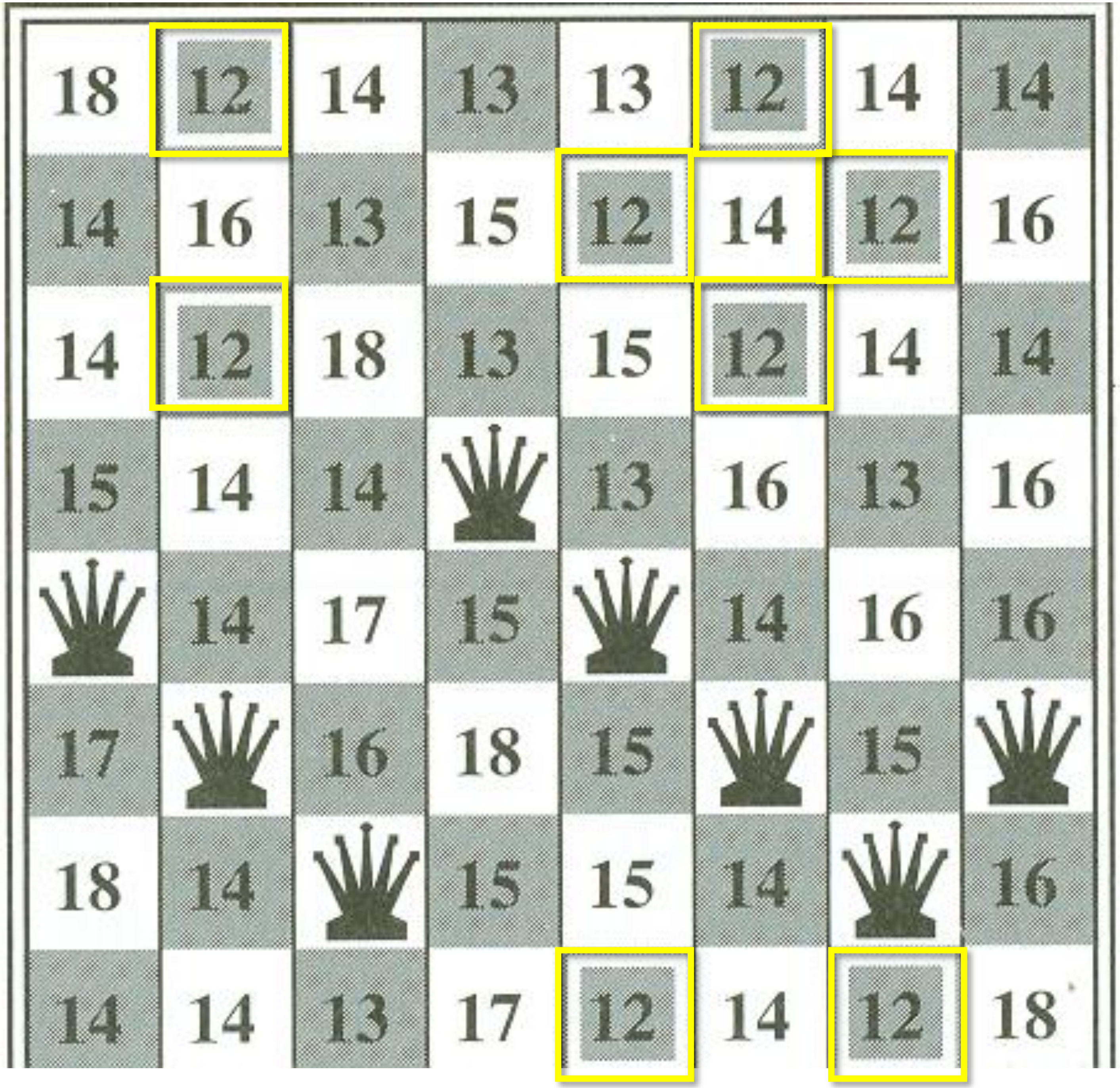
Encoding: (5, 6, 7, 4, 5, 6, 7, 6)



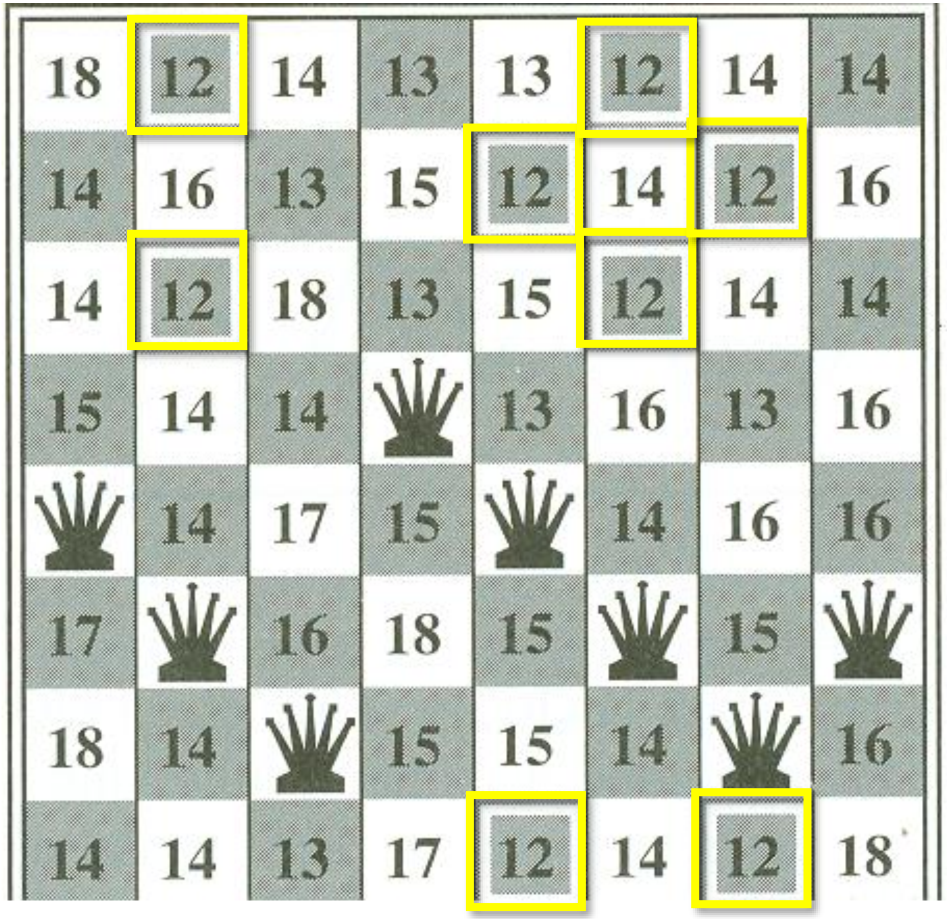
Evaluation: f(s) = 17



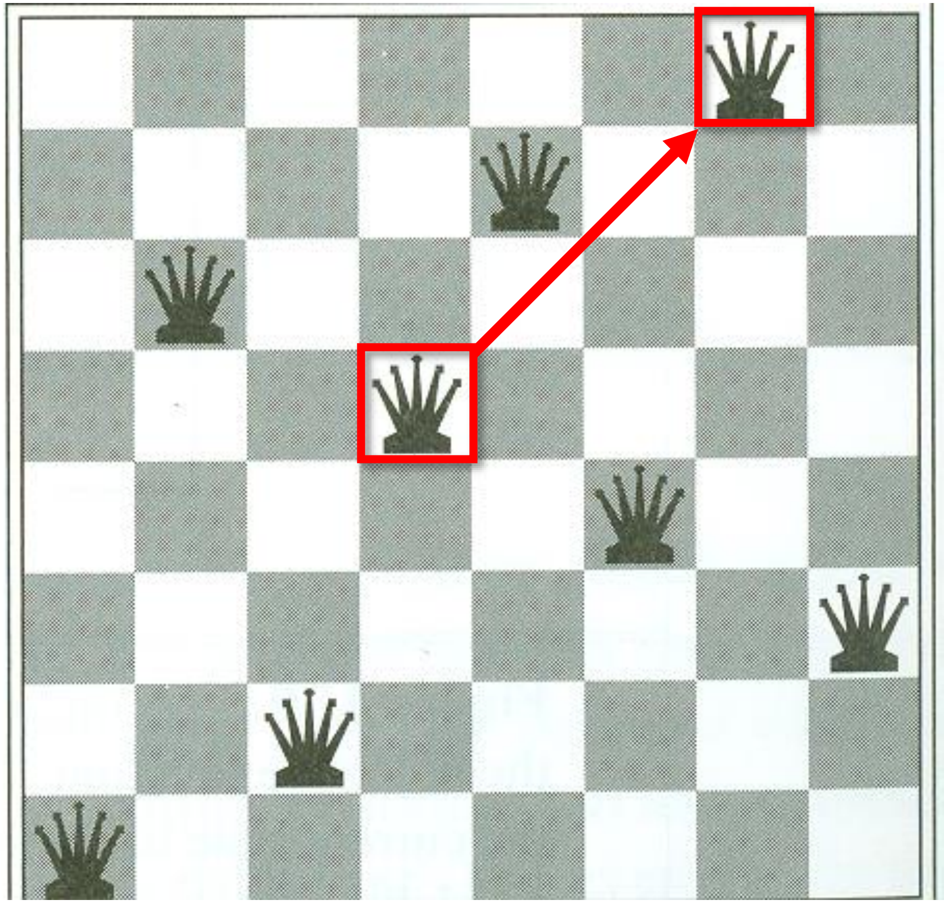
Evaluation: f(s) = 15


18 12 14 13 13 12 14 14
14 16 13 15 12 14 12 16
14 12 18 13 15 12 14 14
15 14 14 13 16 13 16
14 17 15 14 16 16
17 16 18 15 15
18 14 15 15 14 16
14 14 13 17 12 14 12 18

8-queens Problem

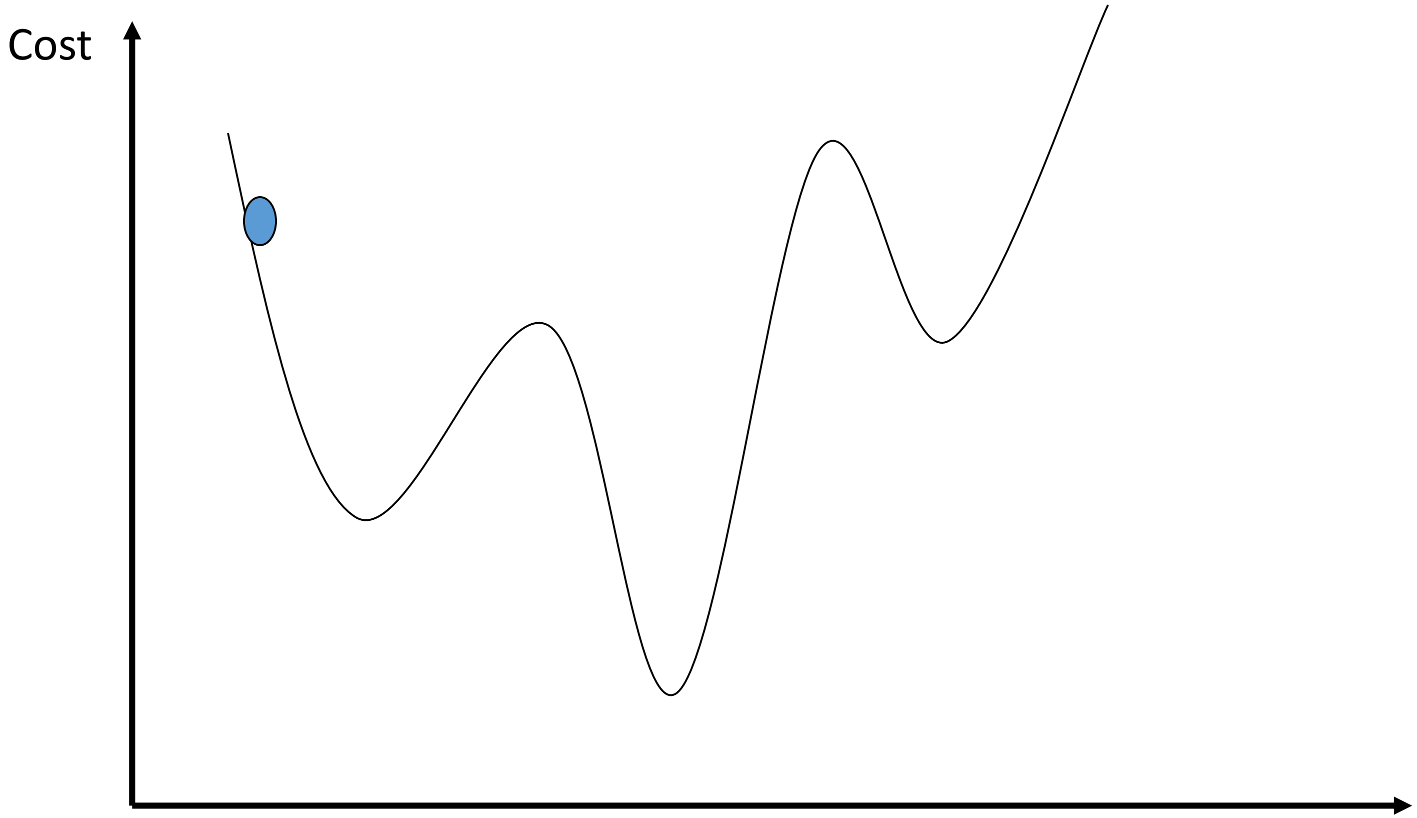


f(s) = 17 best next is 12

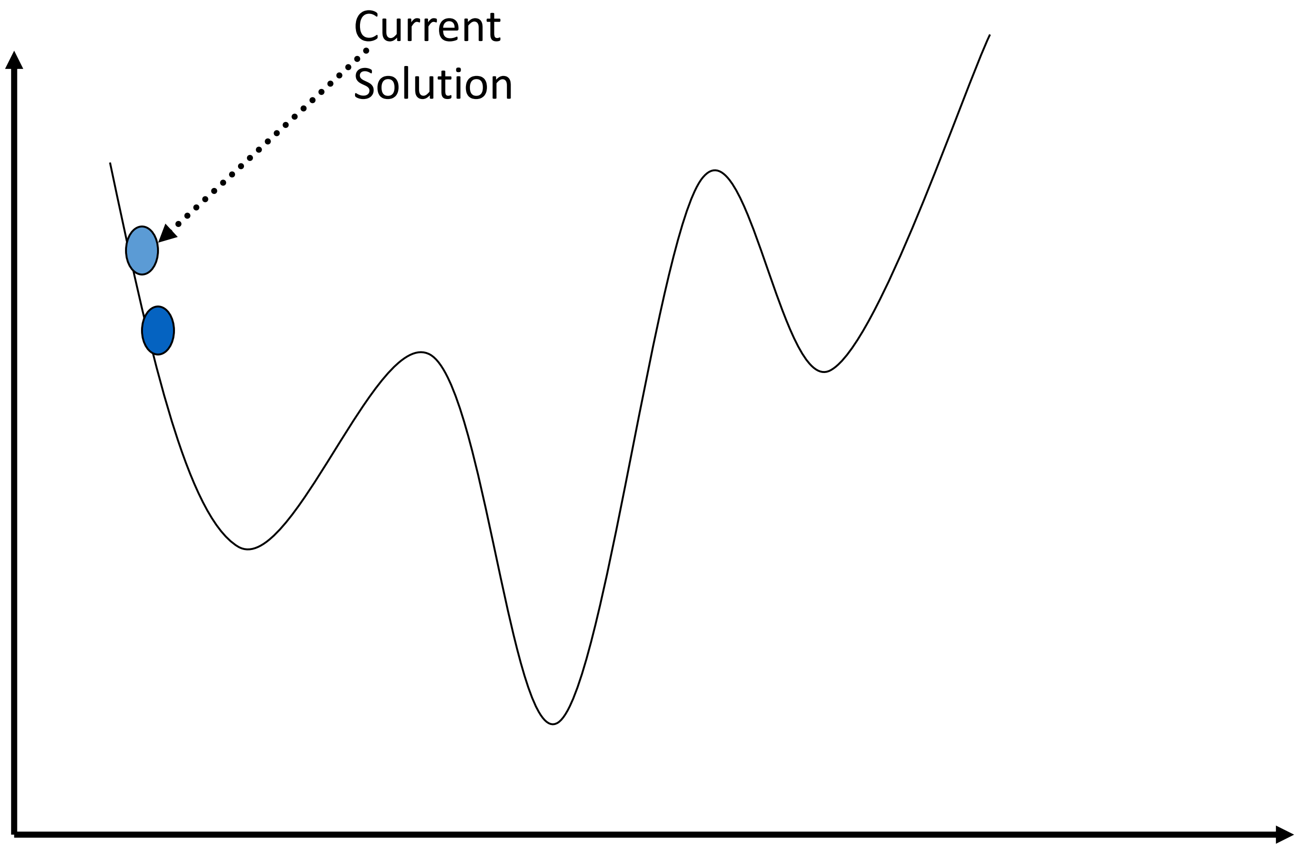


f(s)=1 [local minima]

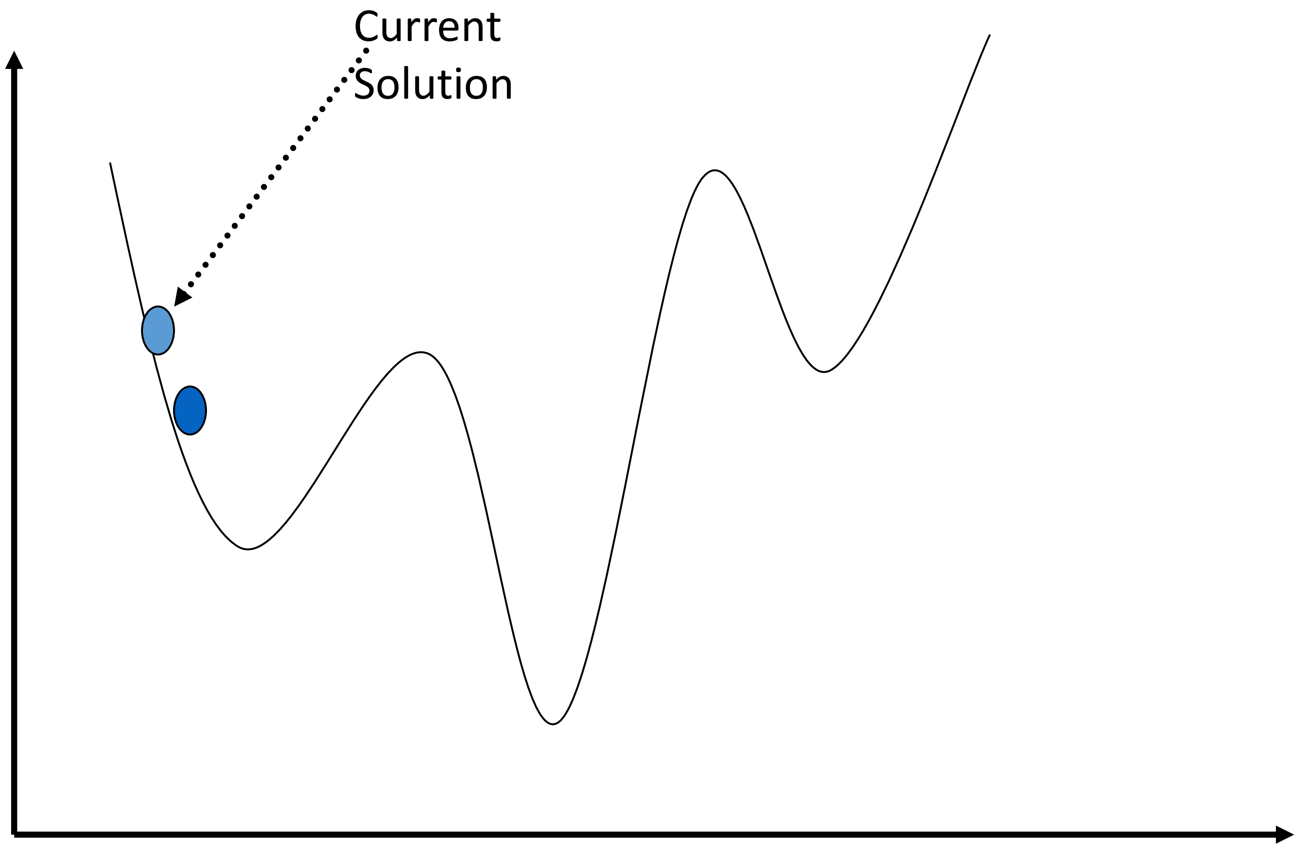
Hill Climbing



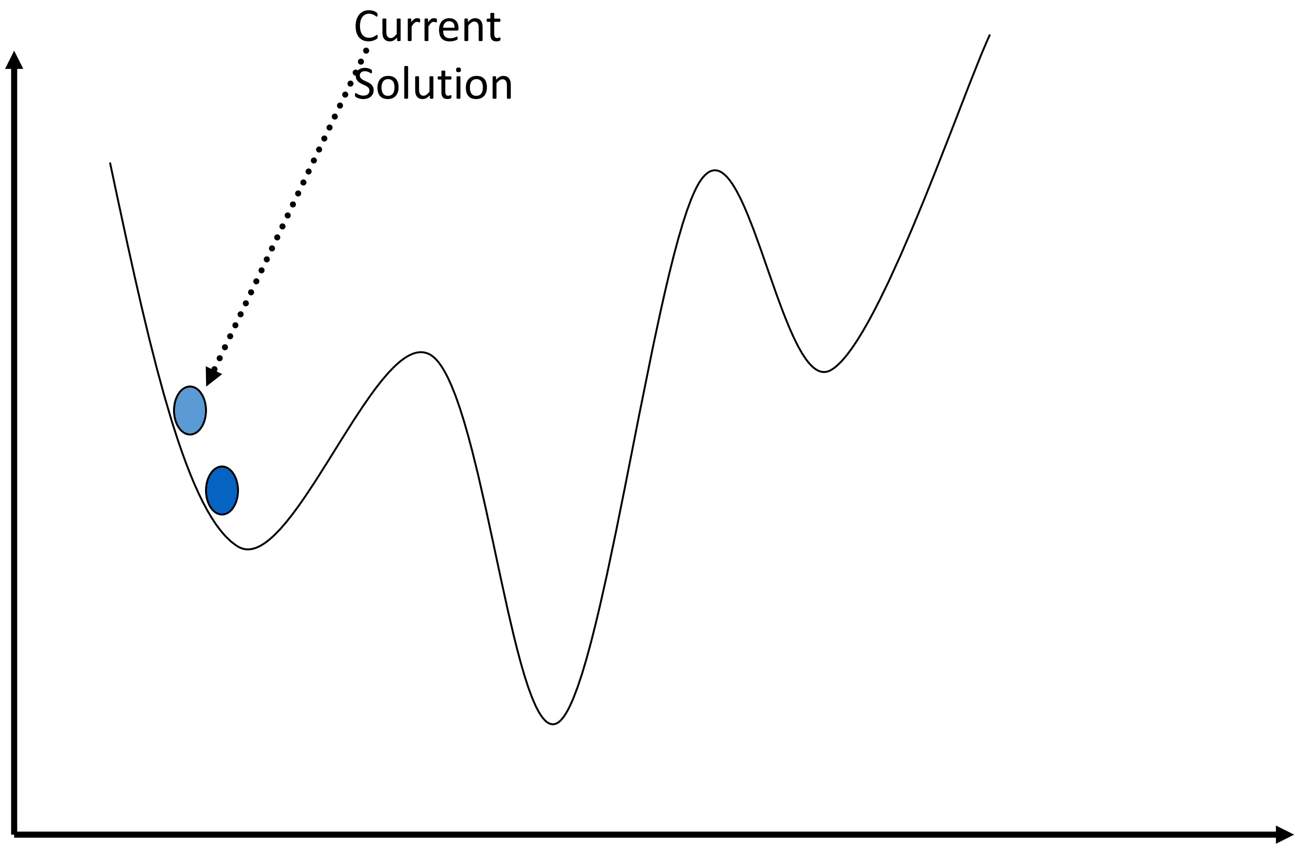
Hill Climbing



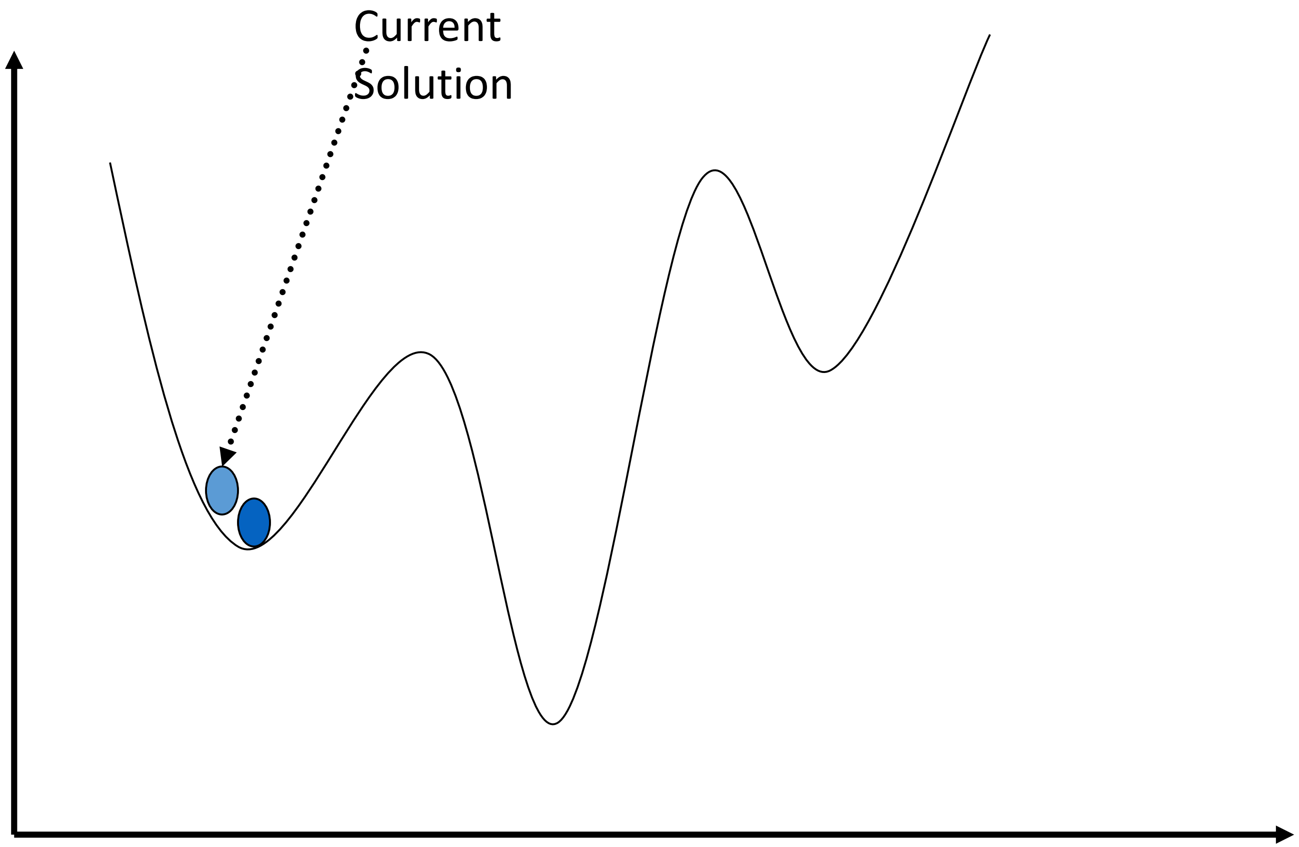
Hill Climbing



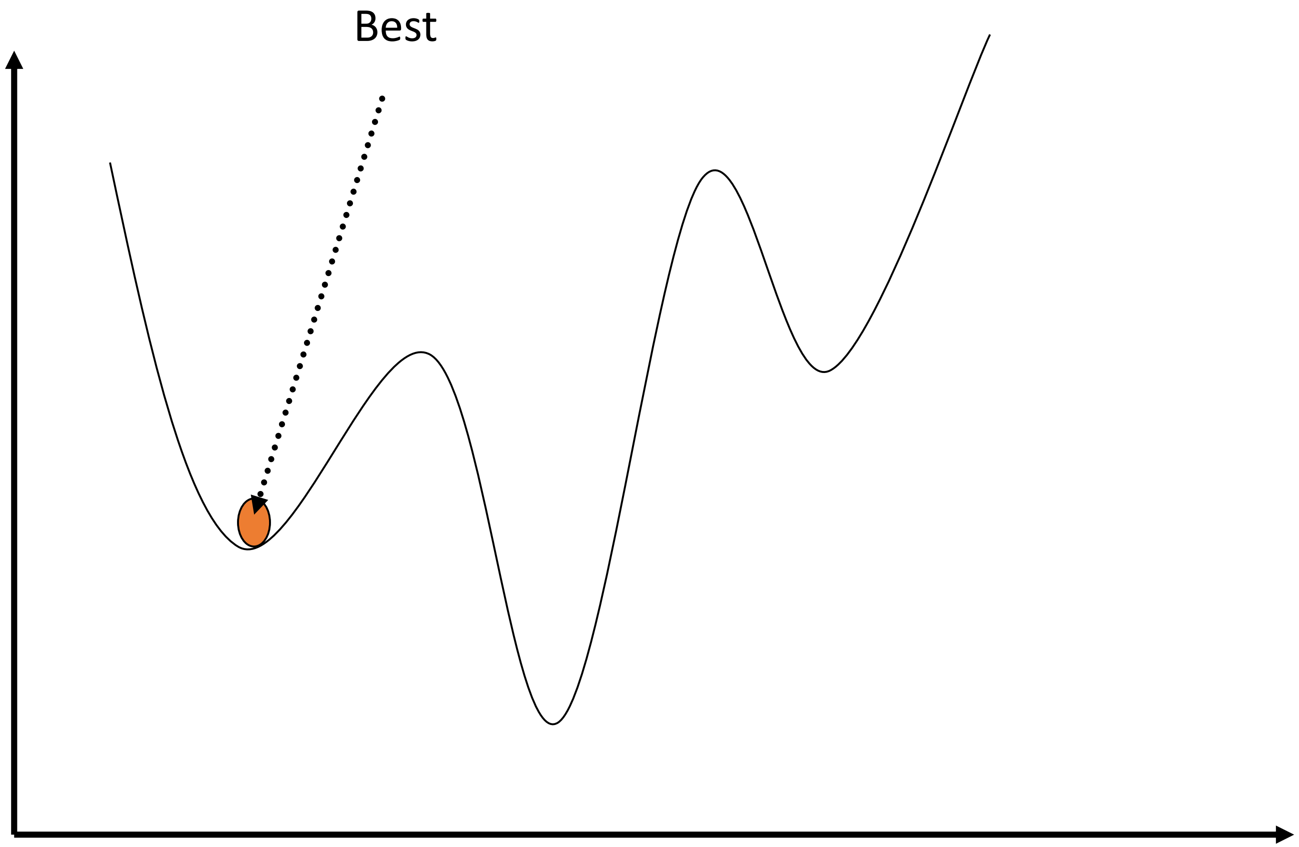
Hill Climbing



Hill Climbing



Hill Climbing



Local search:

Best Accept

a.k.a.

Steepest ascent hill climbing

Local Search 1 : Best Accept

```
input: starting solution, s_0
input: neighborhood operator, N
input: evaluation function, f
current ⇐ s_0
done ⇐ false
while done = false do
    best_neighbor ⇐ current
    for each s ∈ N(current) do
        if f(s) < f(best_neighbor) then
            best_neighbor ⇐ s
        end if
    end for
    if current = best_neighbor then
        done ⇐ true
    else
        current ⇐ best_neighbor
    end if
end while
```

Local search:

First Accept

a.k.a.
Simple hill climbing

Local Search 2 : First Accept

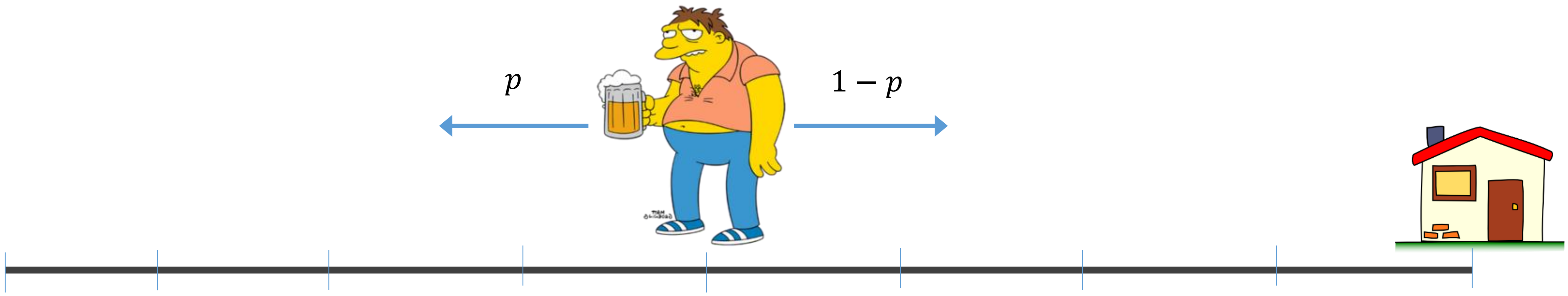
```
input: starting solution, s_0
input: neighborhood operator, N
input: evaluation function, f
current ⇐ s_0
done ⇐ false
while done = false do
    best_neighbor ⇐ current
    for each s ∈ N(current) do
        if f(s) < f(best_neighbor) then
            best_neighbor ⇐ s
            exit the for-loop
        end if
    end for
    if current = best_neighbor then
        done ⇐ true
    else
        current ⇐ best_neighbor
    end if
end while
```

Hill Climbing Comments

- **Pro**: Very fast!
- **Cons**:
  - local maxima/minima will cause HC to stop searching.
  - plateaus: landscape space with a broad flat region gives the HC search algorithm no direction: it either stops or becomes a *random walk*.
- **Variants**:
  - Hill Climbing with random walk
  - Hill Climbing with random restarts
  - Local Beam Search
  - Stochastic Beam Search

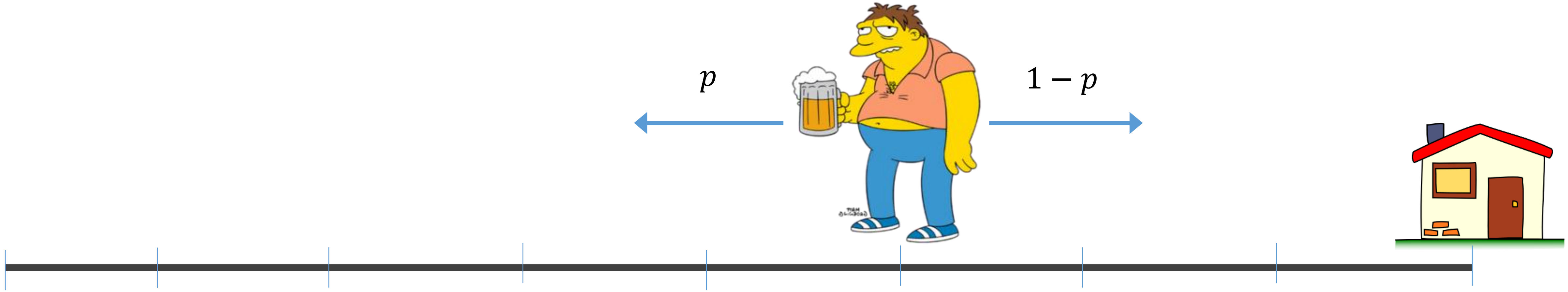
Hill climbing with Random Walk

- Random walk
  - Consider a drunken individual (Barney) stumbling out of a bar one night wanting to go home.
  - For a 1D random walk, Barney stumbles one step to the left with probability p and to the right with probability $1 - p$



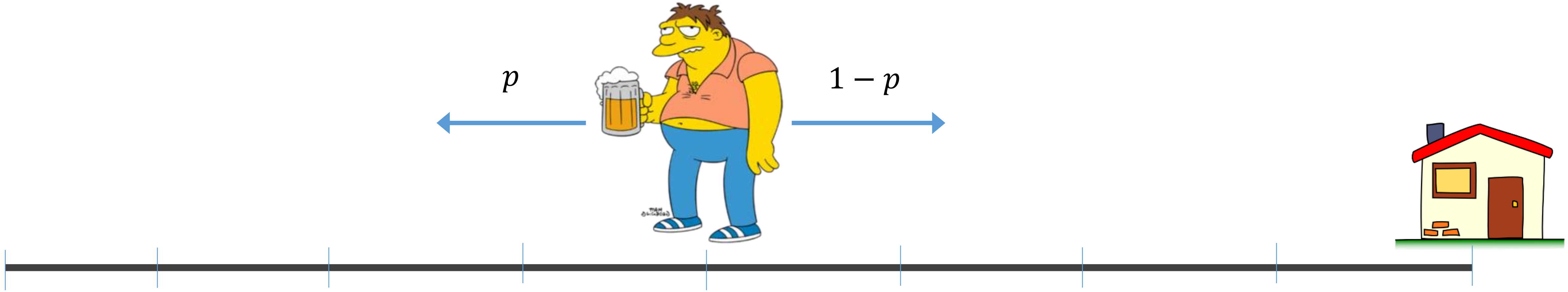
Hill climbing with Random Walk

- Random walk
  - Consider a drunken individual (Barney) stumbling out of a bar one night wanting to go home.
  - For a 1D random walk, Barney stumbles one step to the left with probability p and to the right with probability $1 - p$



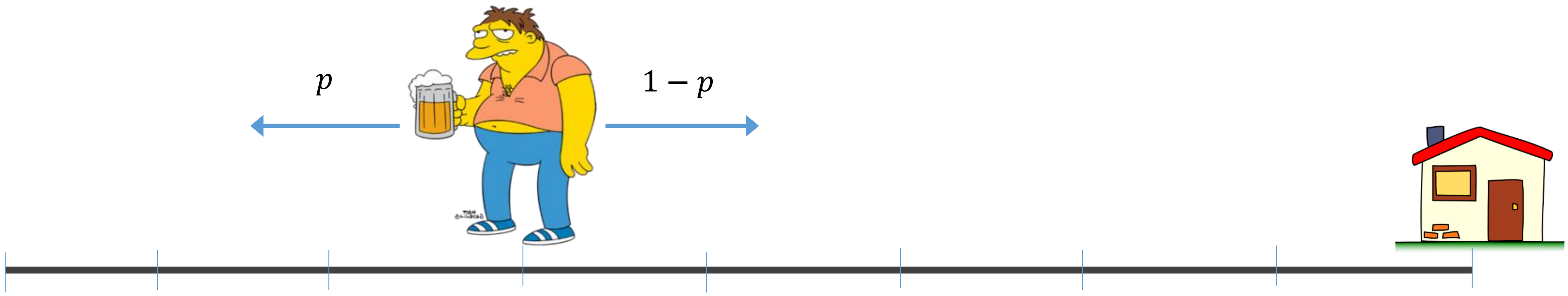
Hill climbing with Random Walk

- Random walk
  - Consider a drunken individual (Barney) stumbling out of a bar one night wanting to go home.
  - For a 1D random walk, Barney stumbles one step to the left with probability p and to the right with probability $1 - p$



Hill climbing with Random Walk

- Random walk
  - Consider a drunken individual (Barney) stumbling out of a bar one night wanting to go home.
  - For a 1D random walk, Barney stumbles one step to the left with probability p and to the right with probability $1-p$



Hill climbing with Random Walk

- At each iteration with probability p, the best neighborhood move is made (hill climbing as usual)
- However, with probability $1 - p$, the algorithm randomly selects a neighbor and moves to it (random walk)

Stochastic Hill climbing

- Another way to introduce randomness in hill climbing
- Stochastic hill climbing: Choose probabilistically from among the improving moves
  - the probability of move can be based on the level of improvement

Hill climbing with random restarts

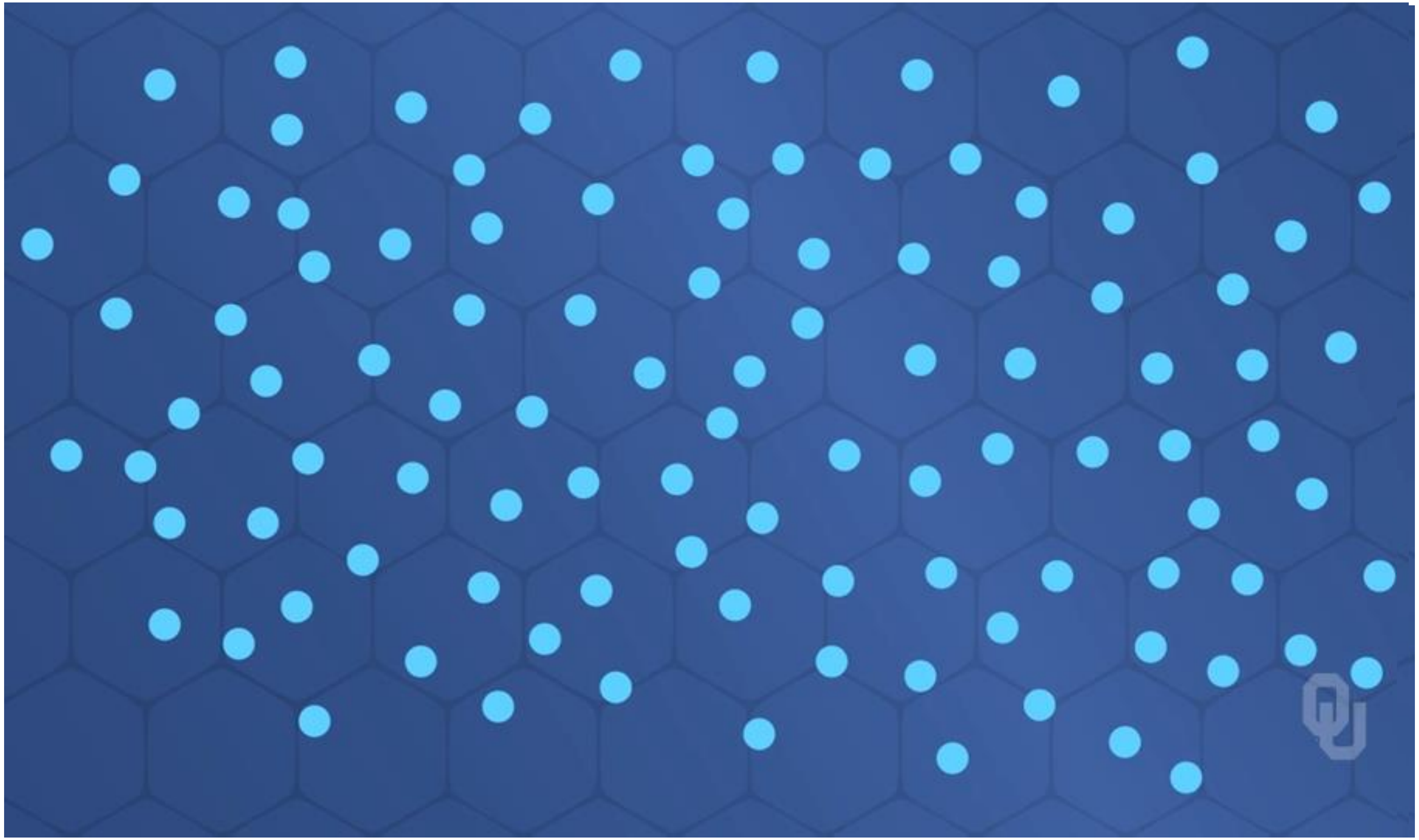
- Easiest and maybe best way to improve hill climbing
- Use the most promising element of hill climbing to its advantage – its speed
- Perform hill climbing many times (e.g., 1,000’s of times) at different random locations each time
- After all hill climbing runs are completed, choose the best overall solution found

Local Beam Search

- Similar to hill climbing with restarts
- Run multiple hill climbing starts but in parallel and share information among the searches

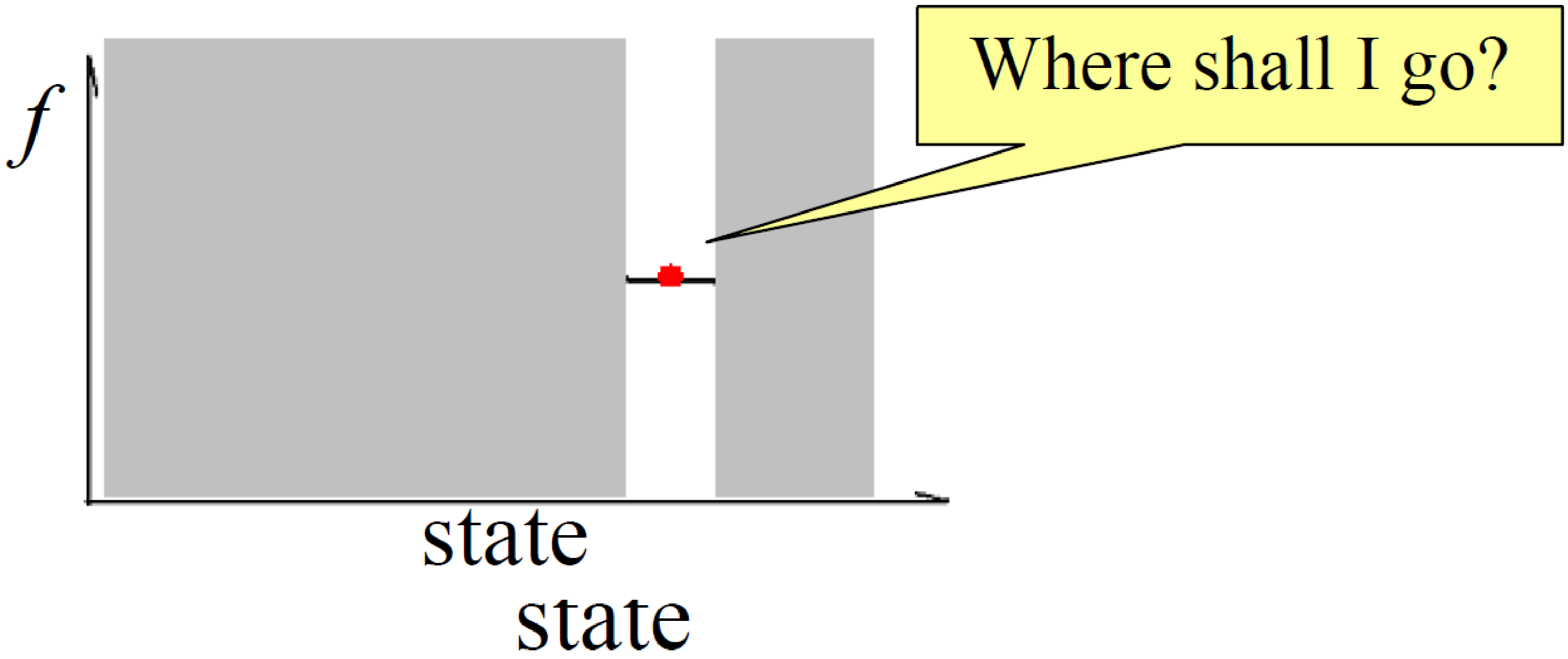
Local Beam Search

- e.g., choose 3 hill climbing searches in parallel, from different starting locations (s_1, s_2, s_3)
- Local beam search chooses the top 3 solutions from across all 3 neighborhoods for next move



Stochastic Beam Search

- Variation of local beam search with more randomness
- A blend between stochastic hill climbing and hill climbing with restarts
- Instead of choosing top k candidates, the k moves are performed probabilistically, with better moves having a higher probability.


Where shall I go?
f
state
state

Main Challenge in Local Search

How can we avoid stopping at a local optimum?

Metaheuristic

Meta: in an upper level

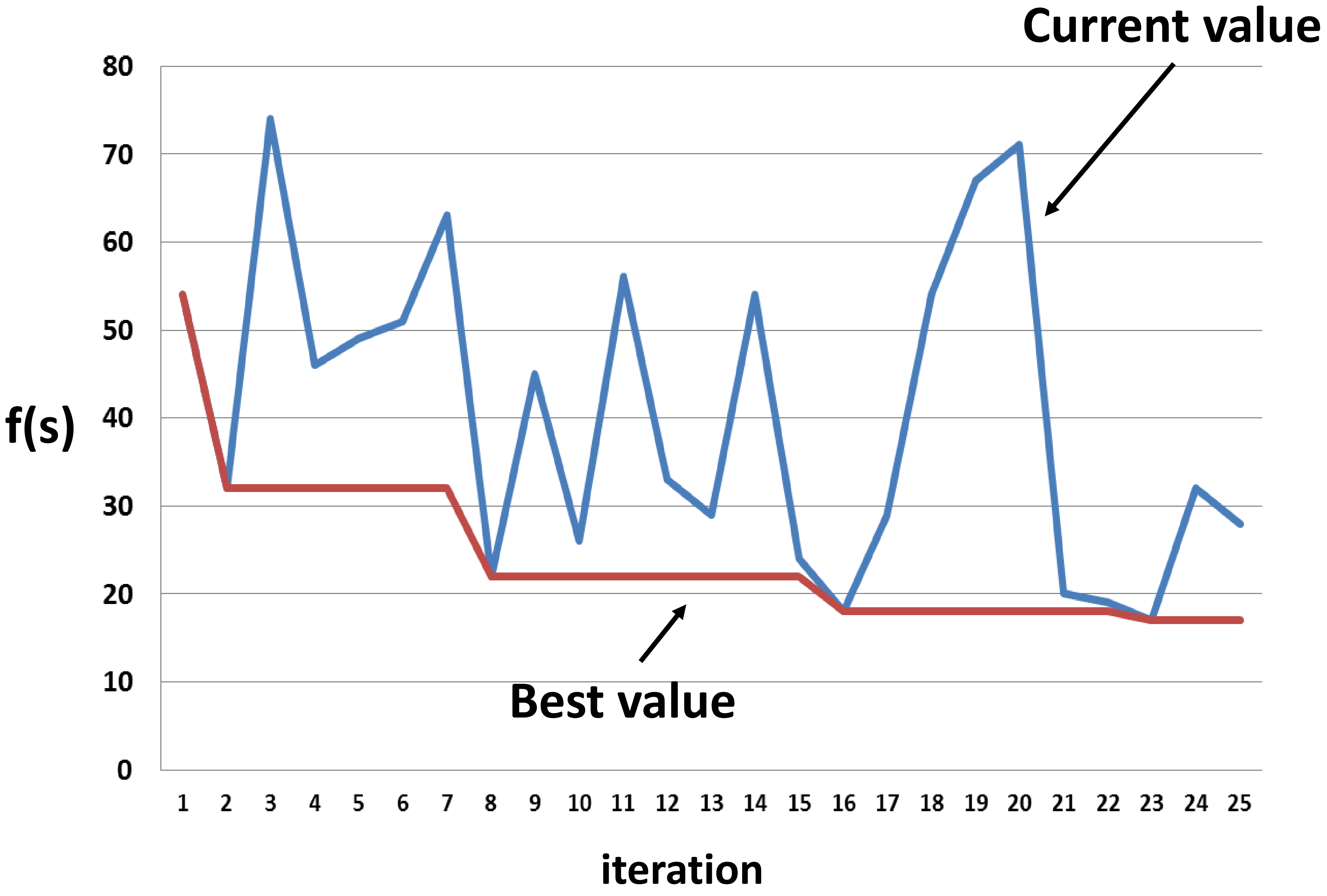
Heuristic: to find

A metaheuristic is defined as an iterative generation process *which guides a subordinate heuristic* by combining (in an intelligent way) various strategies for exploring and exploiting the search space (including learning strategies) to efficiently find near-optimal solutions.

Fundamental Properties of Metaheuristics

- Metaheuristics "orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of solution space."
- Metaheuristic algorithms do not guarantee optimality and usually non-deterministic
- Metaheuristics are not problem-specific

Typical Search Trajectory



Metaheuristic Examples

- **Simulated annealing** (Kirkpatrick et al. 1983)
- **Tabu search** (Glover 1980s)
- **Variable Neighborhood search** (Mladenovic 1997)
- **Genetic algorithms** (1960s/1970s), Evolutionary strategy (Rechenberg & Swefel 1960s), Evolutionary programming (Fogel et al. 1960s)
- **Ant colony optimization** (Dorigo 1992), Genetic programming (Koza 1992),
- **Particle swarm optimization** (Kennedy & Eberhart 1995)
- **Guided Local Search** (Voudouris 1997)

And more…

- Scatter Search (SS)
- Adaptive Memory Procedures (AMP)
- Iterative Local Search (ILS)
- Threshold Acceptance methods (TA)
- Greedy Randomized Adaptive Search Procedure (GRASP)
- Memetic Algorithms (MA)
- Bees algorithm, Artificial Bee Colony (ABC), Bee Hive Optimization
- Bacteria Swarm Foraging Optimization (BSFO)
- The Harmony Method
- The Great Deluge Method
- Shuffled Leaping-Frog Algorithm
- Squeaky Wheel Optimization