Bio-Inspired Algorithms

Simulated Annealing

Daniel Vaz
daniel.ramosvaz@esiee.fr

2026-03-12

Local Search

Last Lecture

Local search:

- Repeated local improvement
- Need solution representation and neighborhood
- Focused on exploitation, little or no exploration

Greedy local search:

- Choose neighbor that minimizes fitness
- Inefficient, but very general
- Gets trapped in local minima
- Examples: minimization, TSP, scheduling, maze solving
- Variations, optimization, and improvements

Local Search

Algorithm LocalSearch

Input: domain D, fitness f

Output: solution $x \in D$

1: $x_0 := \textsc{InitialSolution}(D)$
2: $i := 1$
3: **while** not $\textsc{Finished}$ **do**
4: $\quad N := \textsc{Neighbors}(x, D)$
5: $\quad x_i := \textsc{Selection}(N, x)$ *// May keep the same solution*
6: $\quad i := i + 1$
7: **end while**
8: **return** x_{i-1}

Greedy Local Search

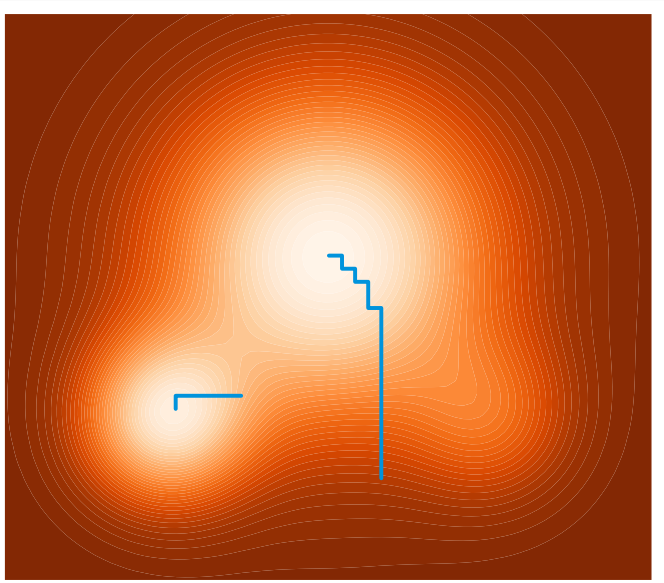
Algorithm GreedyLocalSearch

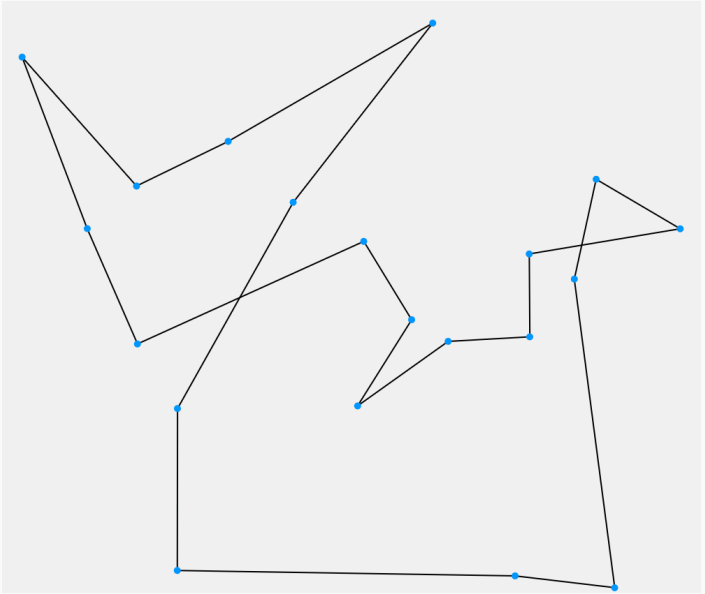
Input: domain D, fitness f

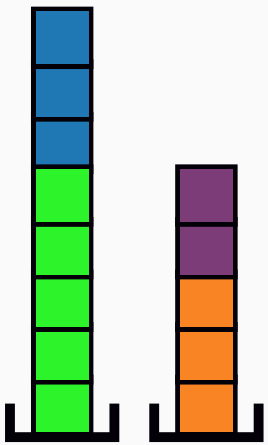
Output: solution $x \in D$

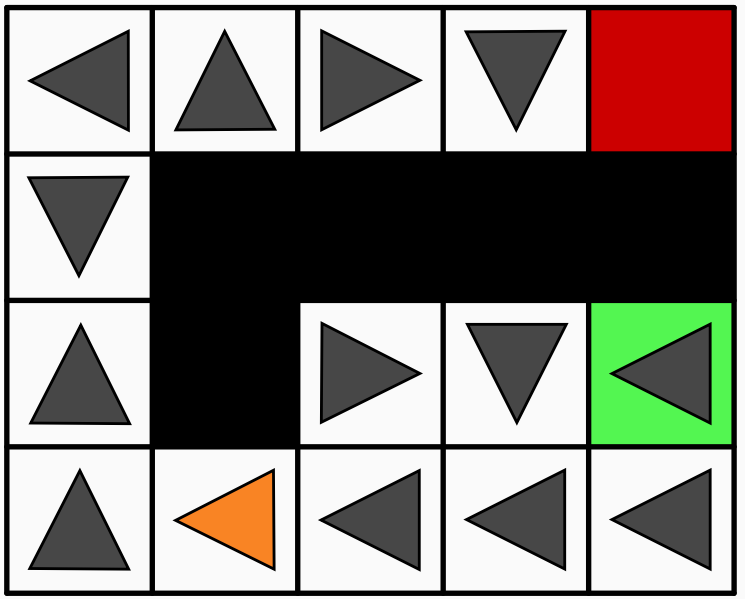
1: $x_0 := \textsc{RandomSolution}(D)$
2: $i := 1$
3: **while true do**
4: $\quad N := \textsc{Neighbors}(x, D)$
5: $\quad x_i := \text{argmin}\{f(c) : c \in N\}$
6: $\quad$ **if** $f(x_i) \geq f(x_{i-1})$ **then**
7: $\quad\quad$ **break**
8: $\quad$ **end if**
9: $\quad i := i + 1$
10: **end while**
11: **return** x_i

Applications of Local Search

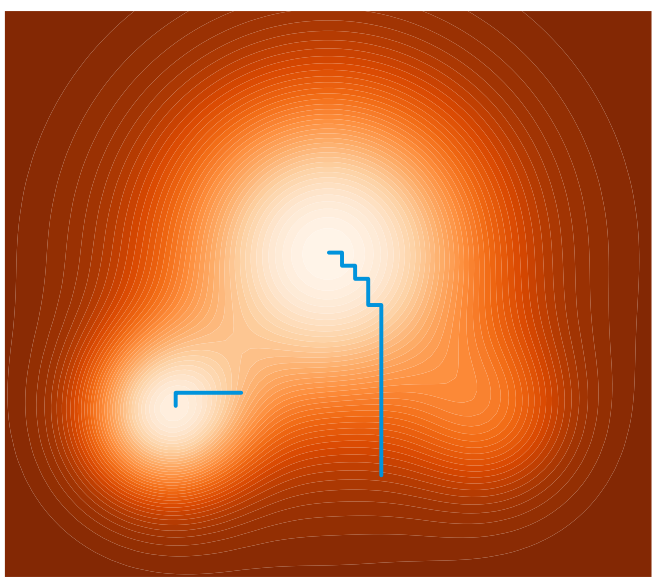








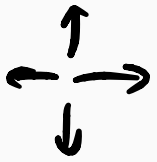
Applications of Local Search



Solution: $x \in \mathbb{R}^d$

Moves: 1. $x_i' = x_i \pm \lambda$

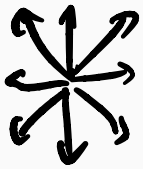
$x_j' = x_i$



numb. $2d$

2. $x_j' = x_j + \{-\lambda, 0, \lambda\}$

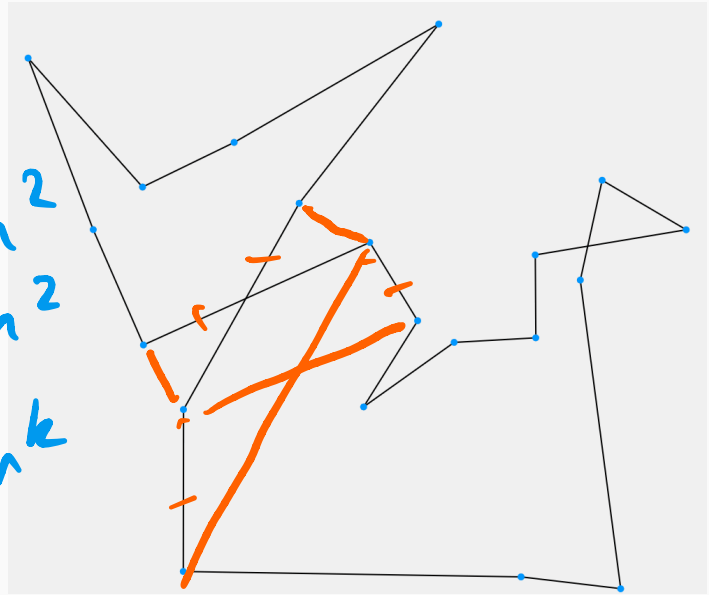
numb. 3^d



Applications of Local Search

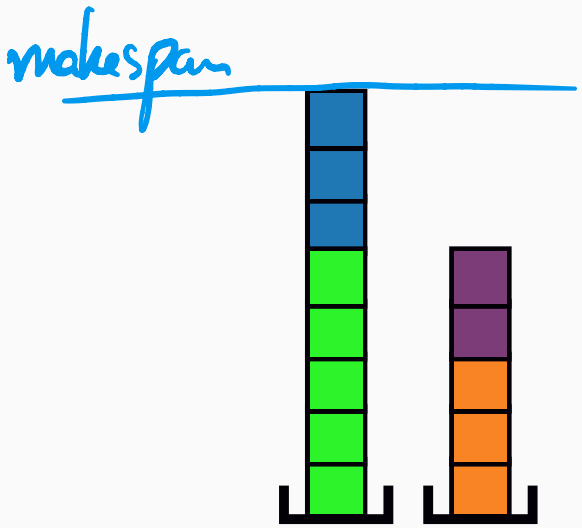
Solution: seq. vertices

Moves: 1. Switch 2 vertices
2. 2-opt: exchange 2 edges
3. k-opt: exchange k edges



Improvements: a. Compute cost by difference.
b. Uncross fast.

Applications of Local Search



Solution: assignment vector ✓
v_i: student for task i

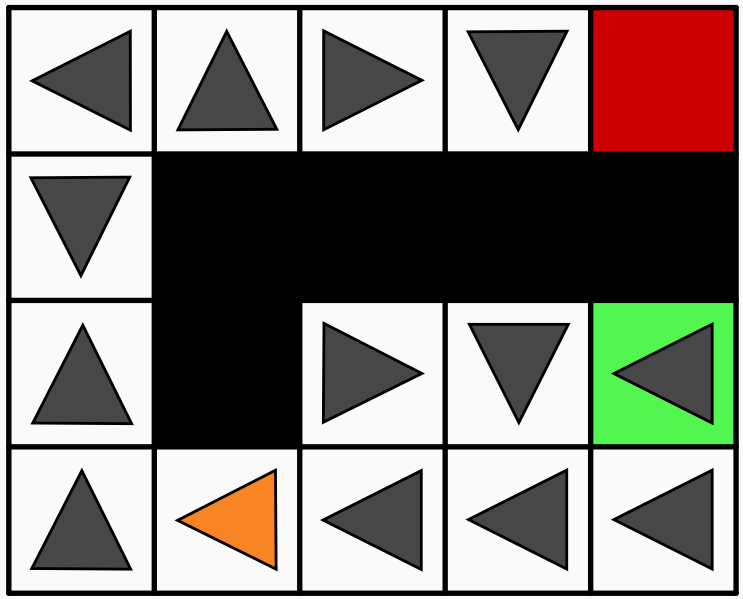
2-approximation algorith

Applications of Local Search

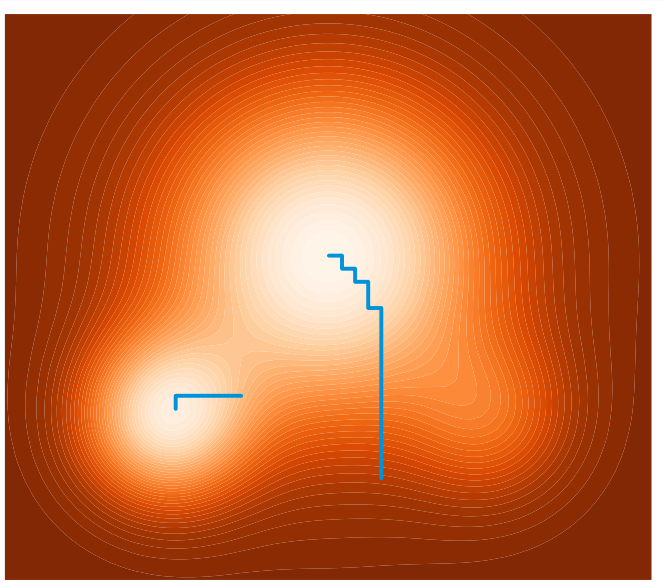
Fitness: Number of steps to the goal

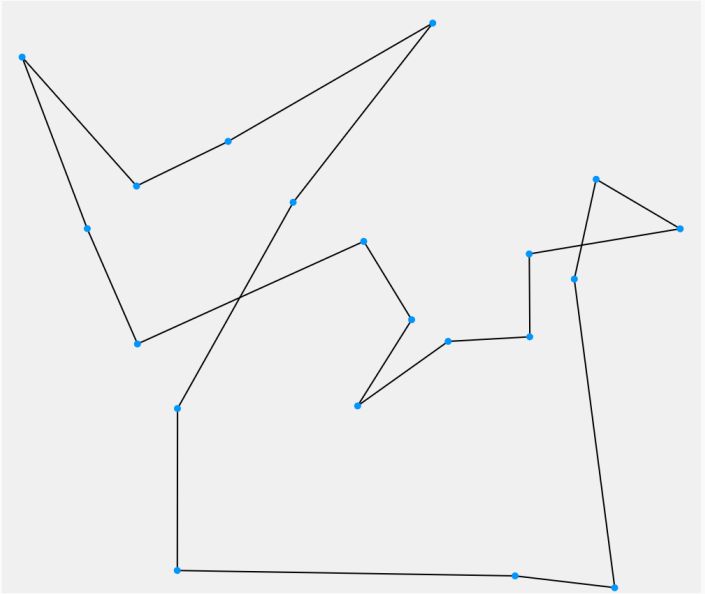
if infeasible:

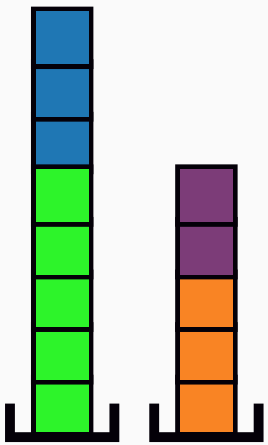
n + #unvisited

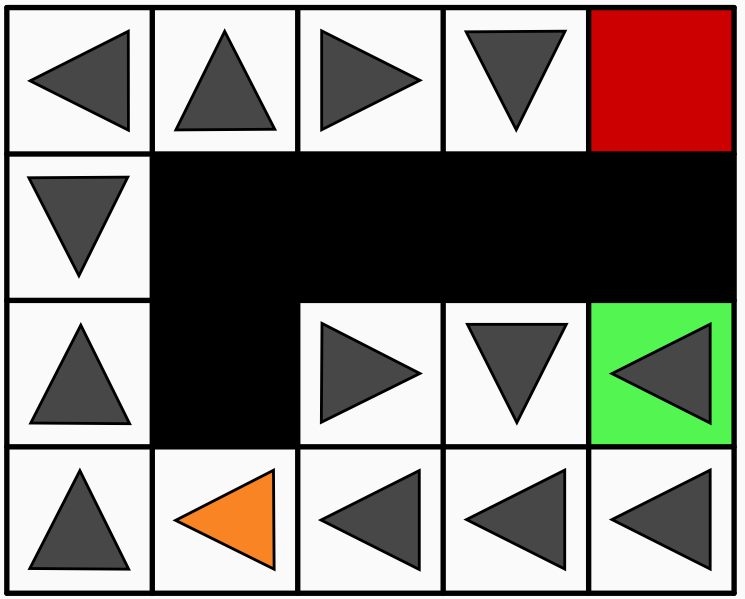


Applications of Local Search









Variations

Choice of neighbor:

- Best possible
  - Very inefficient
- A random improving neighbor
  - Random behavior not always desired
  - Multiple tries?
  - When to stop?
- First improving move
  - Biased towards first neighbors
  - Inefficient in the worst case
  - Threshold?

Running time optimization

Can we improve the running time?

Fitness evaluation

- Repeated many times
  - Even small improvements count
- Idea: Compute difference between solutions
  - TSP: length of changed edges
  - Often $O(1)$ instead of $O(n)$

Neighbor set

- Running time is proportional to its size
- If possible, discard bad neighbors
  - If sure that there is no improvement
- Example: TSP in $\mathbb{R}^2$
  - Always uncross edges first

Local Minima

Local minima: big problem for local search

- Change the fitness
- Change the neighborhood
- Change the strategy

Strategies: increase exploration

- Random restart: once stuck, generate new random solution
- Simulated annealing: accept bad moves "sometimes"
- Tabu search: change the neighborhood dynamically

Local Search

Random Restart

Random Restart

Idea

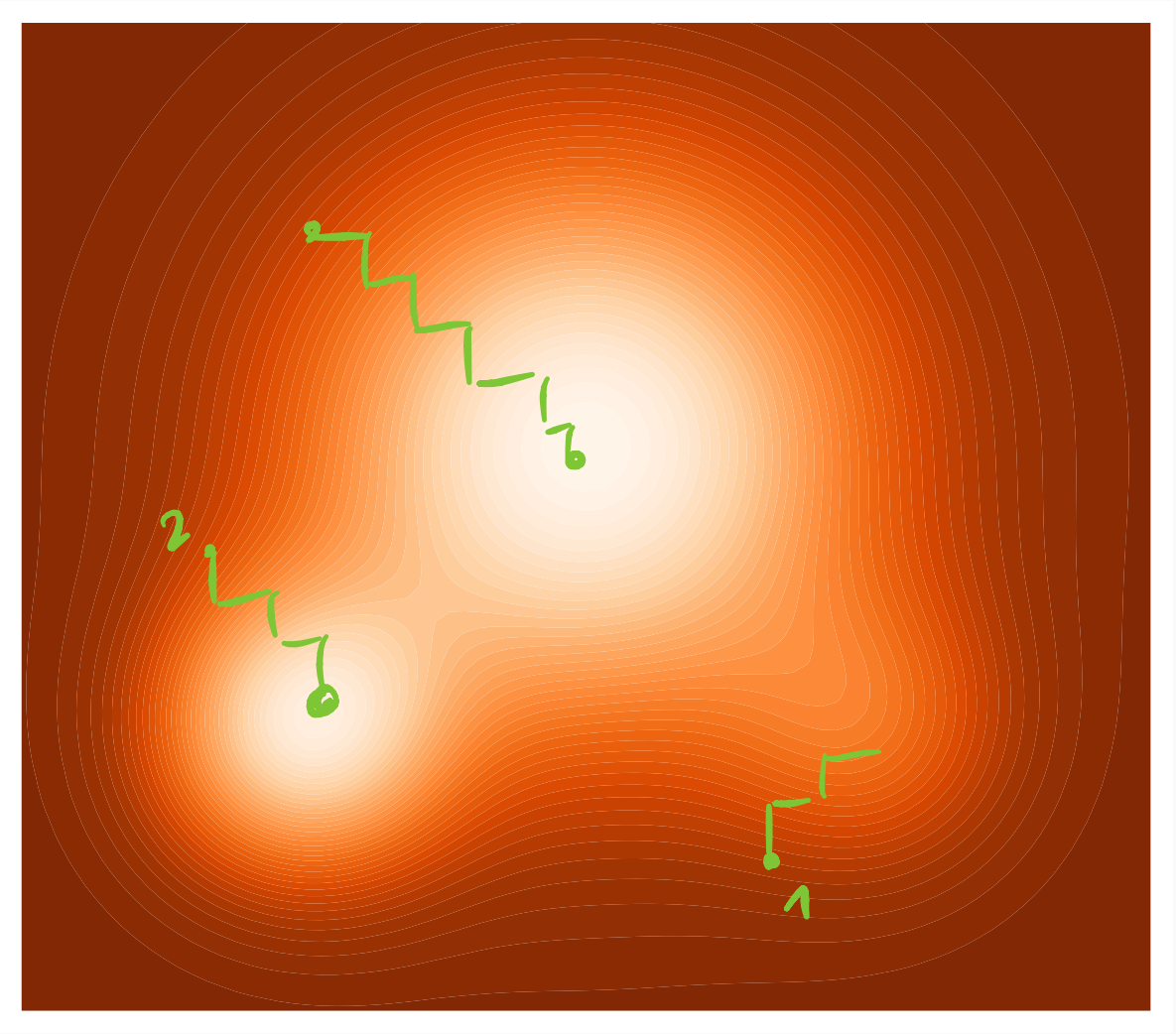
- Restart algorithm with new random solution
- Keep best global and local solutions
- Introduces some exploration

When to restart?

- When local minimum is achieved
- Every k iterations
- Every t seconds

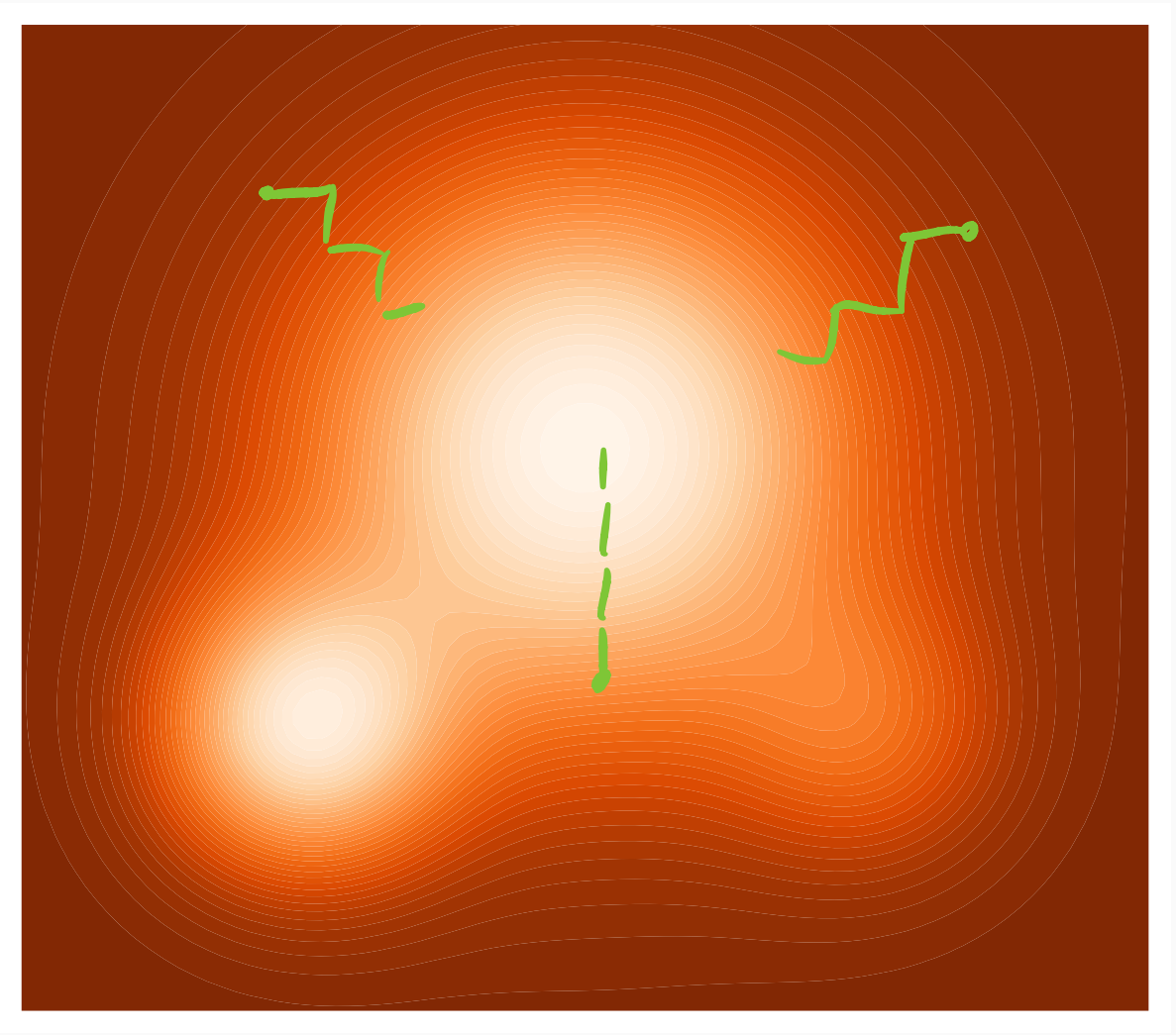
Example

Restart at local minimum



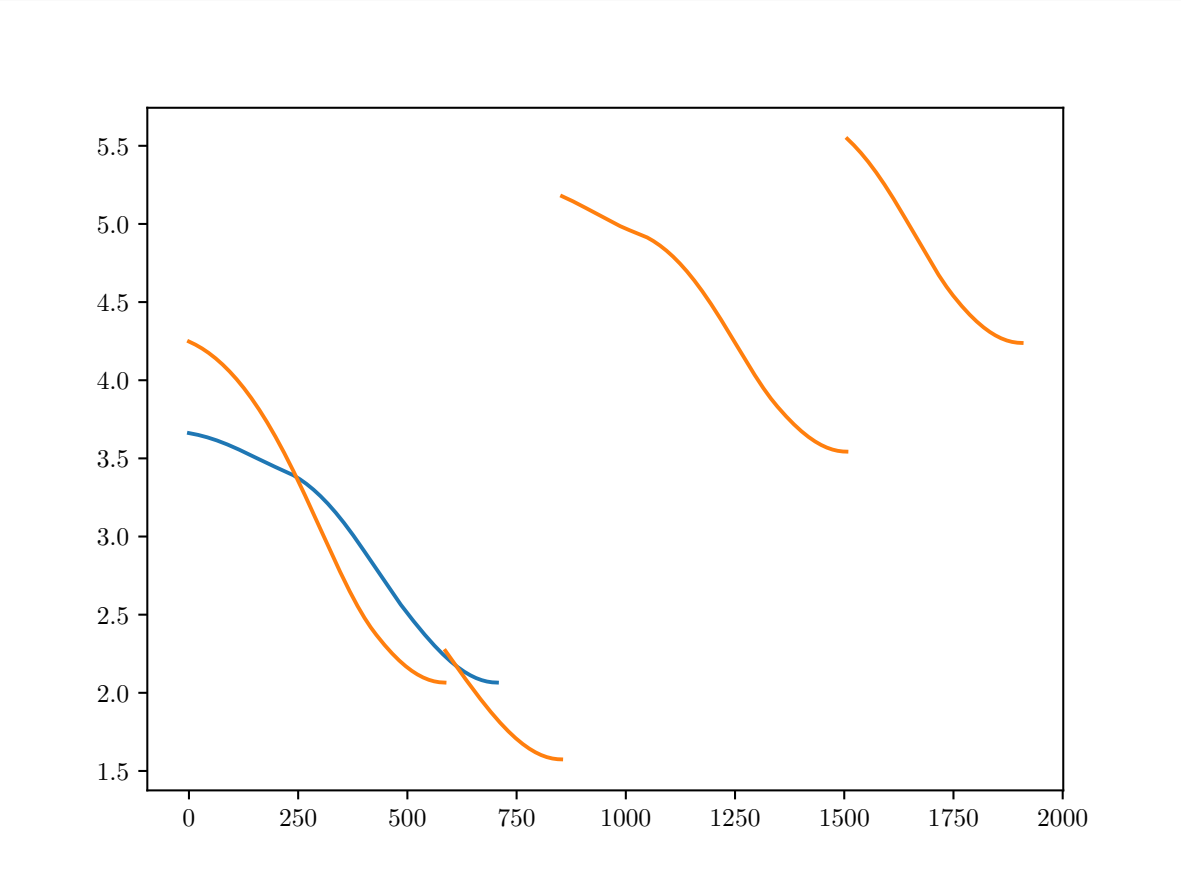
Example

Restart every k iterations



Results

For the Ackley function:



Simulated Annealing

Simulated Annealing

Technique for randomized search

- Introduced by Kirkpatrick, Gelatt, Vecchi for TSP (1983)

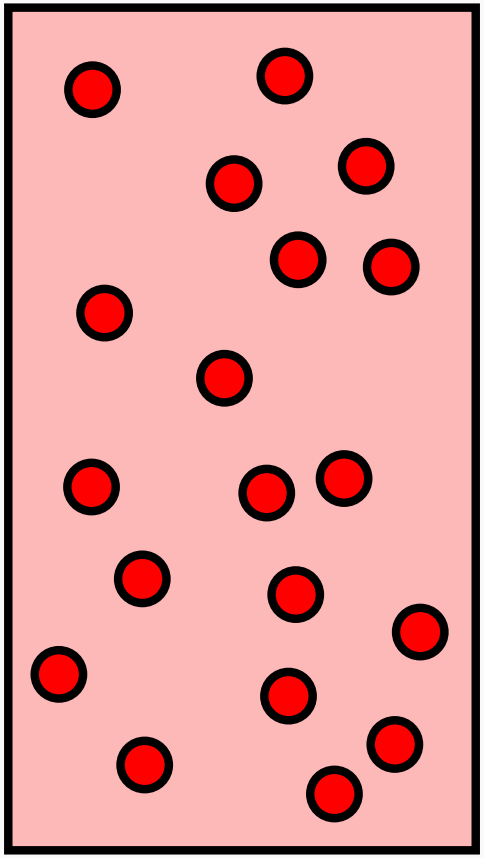
Principle

- Local search with some exploration
- Accept bad moves with some probability
- Allows escaping from local optima

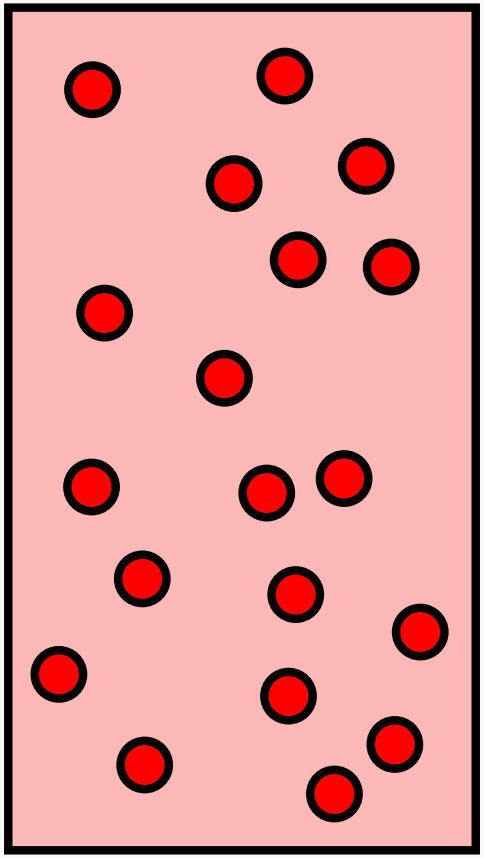
Inspiration

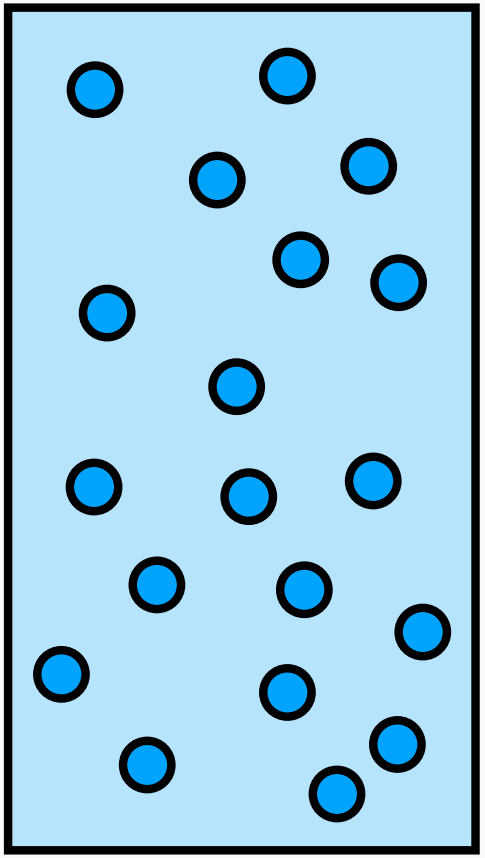
- Crystallization of metals at high temperatures
- Atoms escape from structure, which might improve it
- As the metal cools, the structure settles

Simulated Annealing



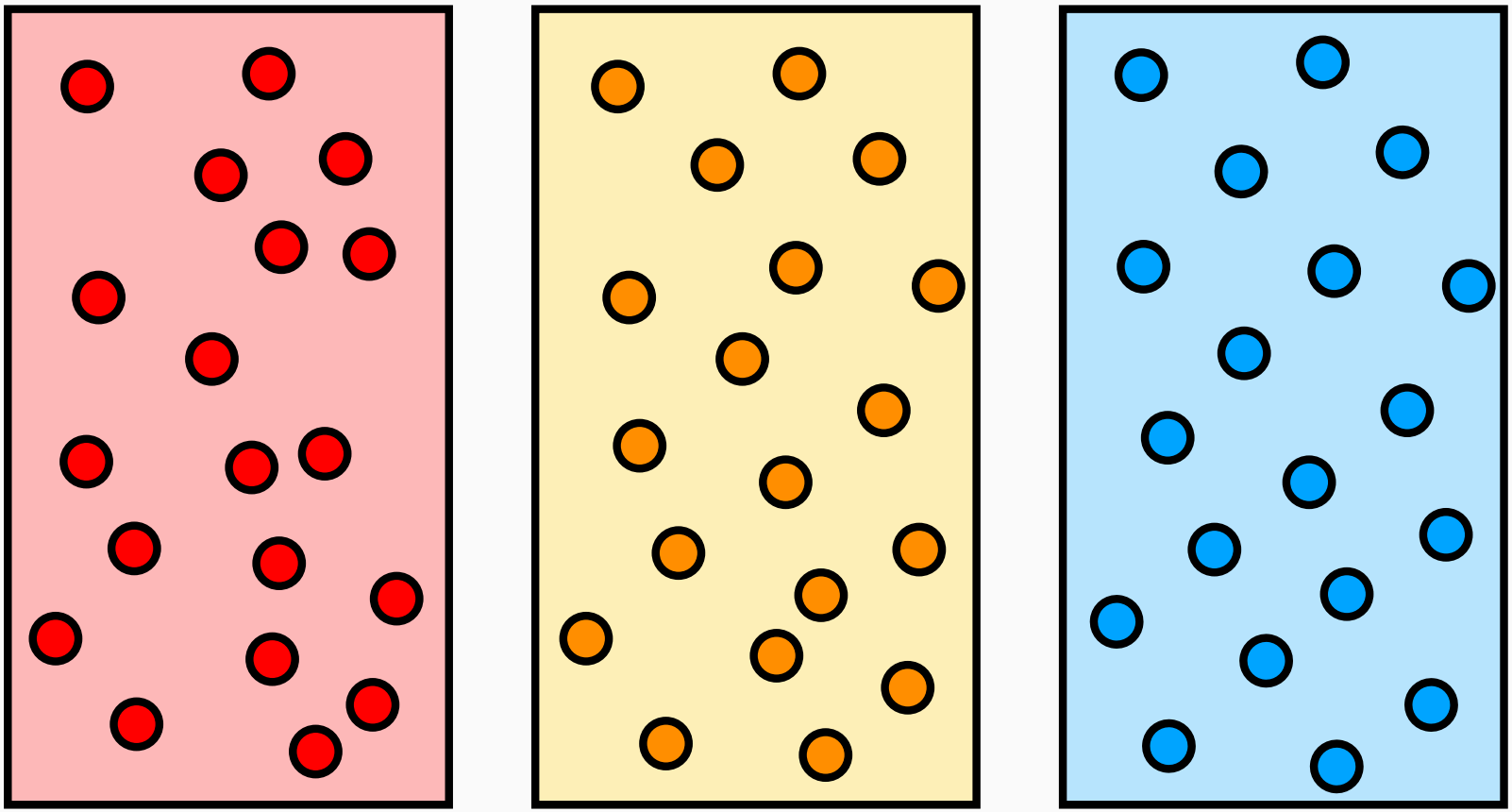
Simulated Annealing





Rapid cooling

Simulated Annealing



Slow cooling

Simulated Annealing

Principle

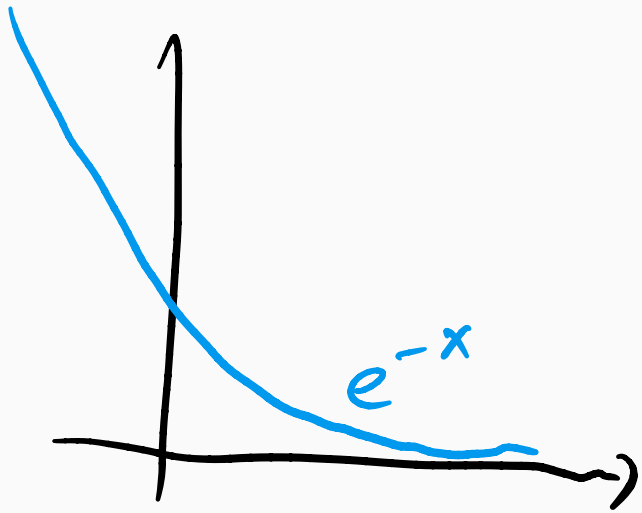
- Accept bad moves with some probability

Details

- x : ~~Best~~ solution so far
- Take a random neighbor c
- If c better than x, take $x := c$
- Otherwise, take $x := c$ with probability

$$\exp\left(-\frac{f(c) - f(x)}{T}\right)$$

- T is the temperature of the algorithm



Simulated Annealing

Details

- Accept a bad move with probability $\exp\left(-\frac{f(c)-f(x)}{T}\right)$
- T is the temperature

Some remarks:

- if c is close to x, likely to accept
- if $T = 0$, never accepts
- if T is large, always accepts

How do we make the algorithm converge?

- Decrease temperature over time!
- Temperature starts very high and decreases towards 0
- Schedule: how temperature changes

Simulated Annealing

Algorithm SimulatedAnnealing

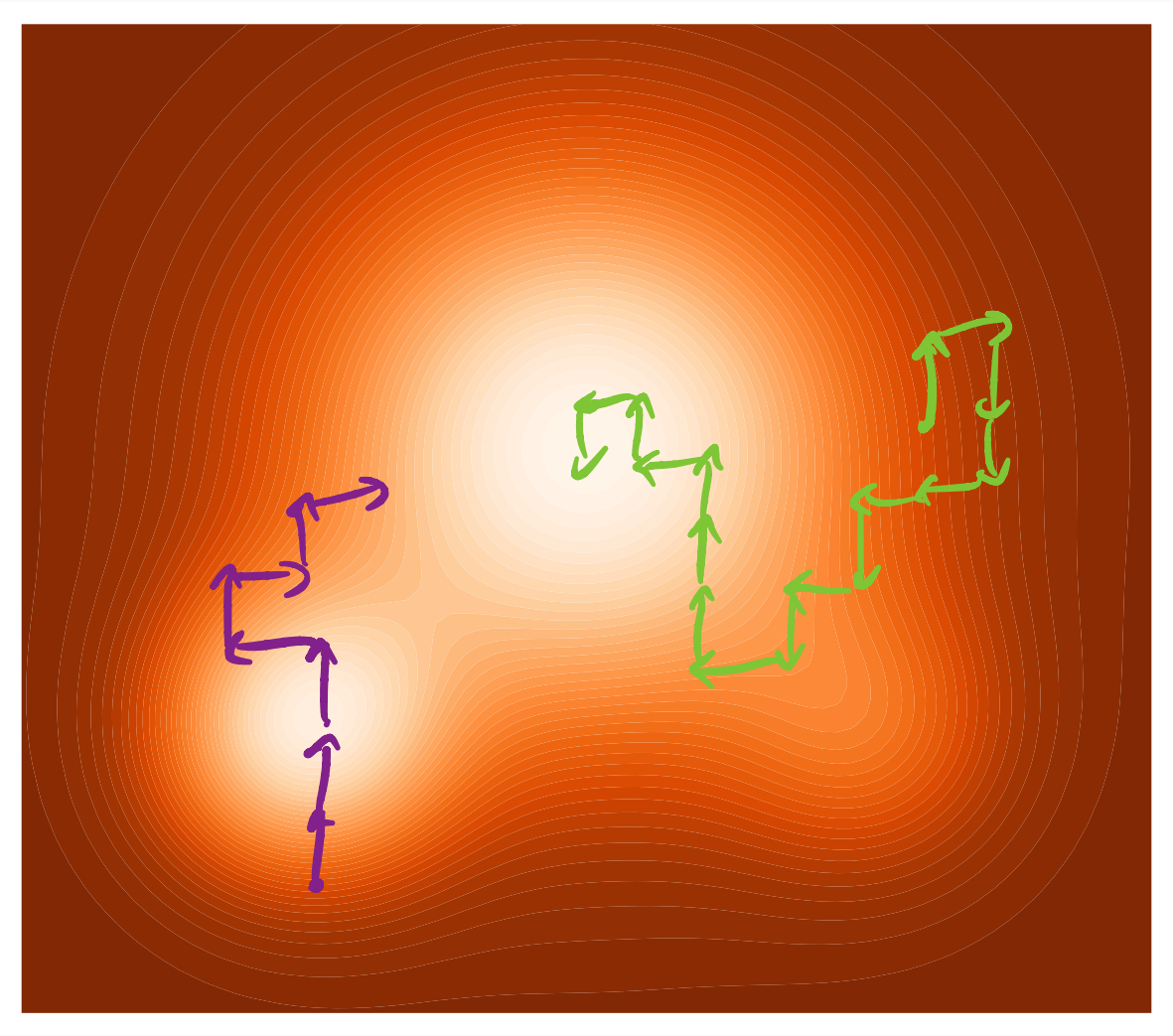
Input: domain D, fitness f, temperature T

Output: solution $x \in D$

1: $x := \textsc{RandomSolution}(D)$
2: **while** $T \not\approx 0$ **do**
3: $\quad c := \textsc{RandomNeighbor}(x, D)$
4: $\quad$ **if** $\big(f(c) < f(x)\big) \vee \big(\textsc{random}() \leq e^{-(f(c)-f(x))/T}\big)$ **then**
5: $\quad\quad x := c$
6: $\quad$ **end if**
7: $\quad T := \textsc{UpdateTemp}(T)$
8: **end while**
9: **return** x

$\in [0,1]$

Example



Simulated Annealing

Convergence

Convergence

Does this algorithm converge?

- Yes (if temperature tends to 0)
- Randomized, so hard to analyze
- Can show it converges with some probability

Big theoretical results:

- Follows statistical distribution of fitness
- Approaches the optimum with the right schedule

Convergence

Statistical equilibrium

- Assuming we run a large number of iterations with constant T
- Will end at a solution c with probability $\sim e^{-f(c)/T}$
- Statistical distribution follows the fitness

Asymptotic convergence

- Converges to an optimum solution
- As the temperature tends towards 0
- And sufficient rounds are given to "each temperature"

Convergence

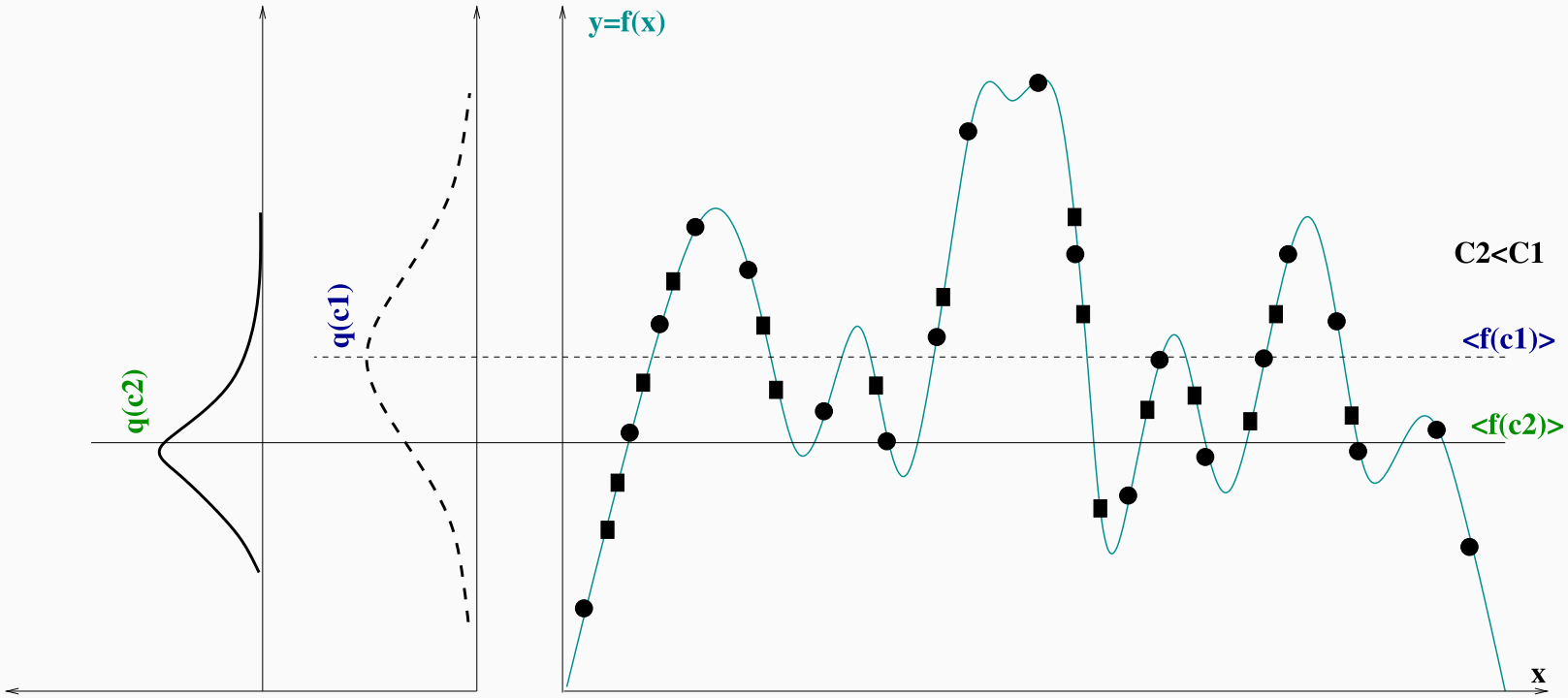


Figure 2: Distribution of the objective-function values at some high temperature c_1 and at a lower temperature c_2.

Delahaye et al. *Simulated annealing.* Handbook of Metaheuristics

Simulated Annealing

Schedules

Simple Schedules

How do we reduce T?

- Any function that goes to 0
- E.g. $T_i = T_0/i$

What is the initial temperature?

- Experimentally verified
- Should be close to the fitness values

For more complicated schedules, convenient to talk about stages

Simulated Annealing by Stages

Algorithm SimulatedAnnealingStages

Input: domain D, fitness f, temperature T

Output: solution $x \in D$

1: $x := \textsc{RandomSolution}(D)$
2: **while** $T \not\approx 0$ **do**
3: **for** $i = 1$ to $\textsc{LengthStage}(T)$ **do**
4: $c := \textsc{RandomNeighbor}(x, D)$
5: **if** $\big(f(c) < f(x)\big) \vee \big(\textsc{random}() \leq e^{-(f(c)-f(x))/T}\big)$ **then**
6: $x := c$
7: **end if**
8: **end for**
9: $T := \textsc{UpdateTemp}(T)$
10: **end while**
11: **return** x

Schedules

Initial temperature

1. Set experimentally
2. Increase as power of 2 until bad moves always accept
    - Try 30 random solutions, each with a few random neighbors

Temperature reduction

1. Geometric: $T_k = \alpha T_{k-1}$
2. Polynomial: $T_k = T_0/(k+1)$
    - Or equivalently $T_k = T_{k-1}/\left(1 + \frac{T_{k-1}}{T_0}\right)$
3. Logarithmic: $T_k = T_0/\log(k+2)$
    - Usually with stage length 1

$T_k = T_0 \cdot \alpha^k$

Schedules

Length of the stages

1. Experimental
2. Around neighborhood size

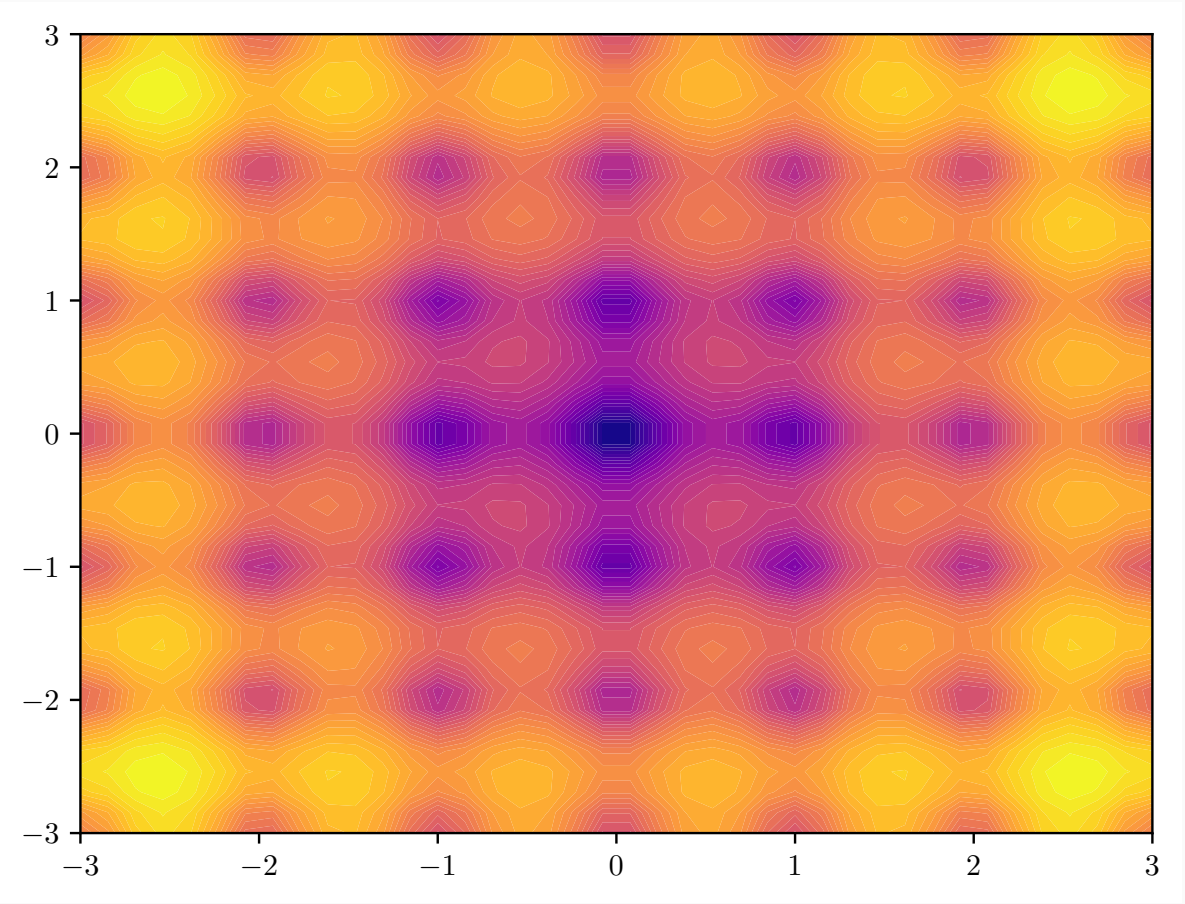
Stopping criterion

1. Temperature close to 0 0.001
2. Little improvement over several iterations

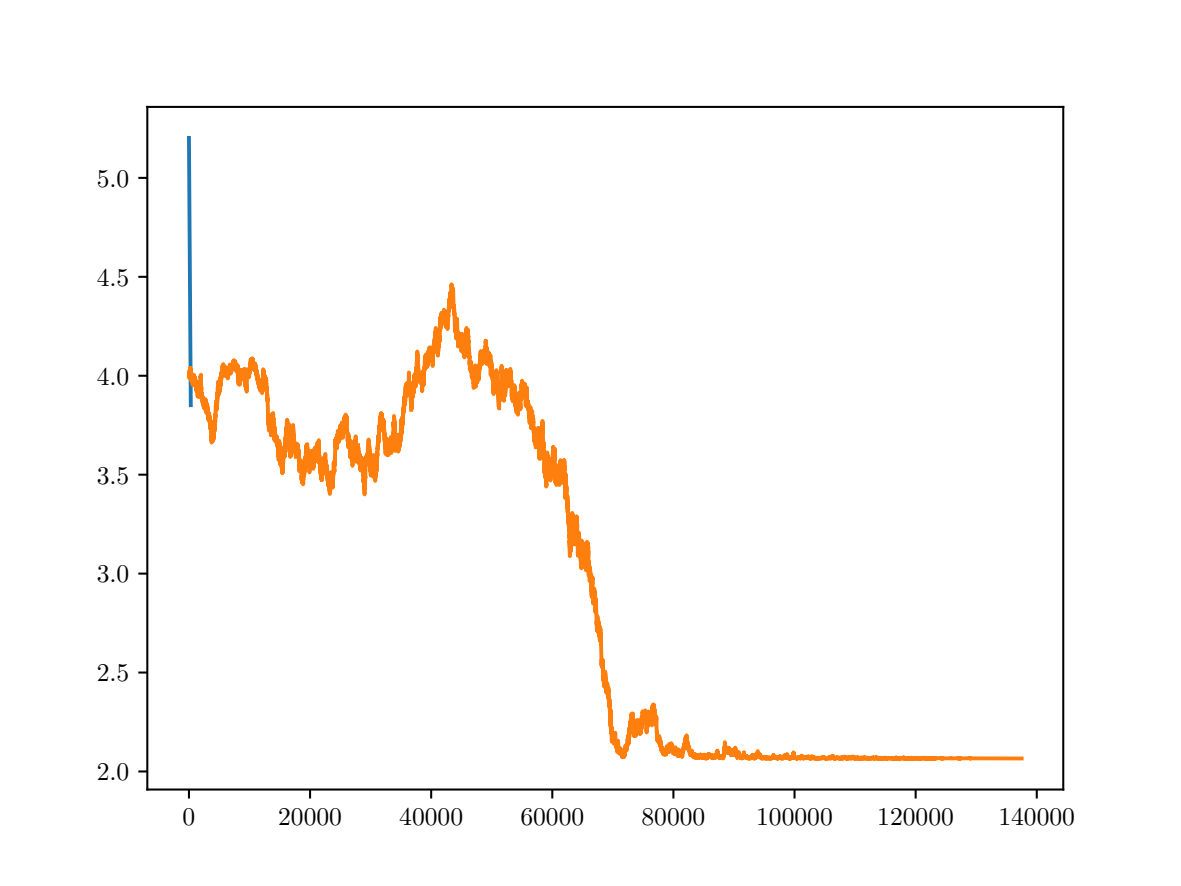
Simulated Annealing

Example

Example



Results


5.0
4.5
4.0
3.5
3.0
2.5
2.0
0
20000
40000
60000
80000
100000
120000
140000

Results

Parameters:

- Initial temperature: $T_0 = 100$
- Decay: geometric, $T_k = 0.99\,T_{k-1}$
- Stopping criterion: $T_k \leq 0.0001$
- Length of stages: 100

Simulated Annealing

Summary

Simulated annealing

Method that allows bad moves, depending on the temperature

Advantages

- Generates good solutions
- Allows different types of customization
- Backed by theoretical convergence results

Disadvantages

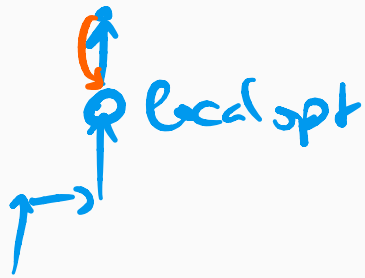
- Many parameters to configure
- Running time
    - In particular, with stages of size equal to neighborhood

Tabu Search

Tabu Search

Idea

- Deterministic algorithm that avoids repetition
- When a local optimum is found, allow best bad move



Problem

- After bad move, best move might be going back
- So forbid reversing previous moves

Tabu Search

Idea

- Deterministic algorithm that avoids repetition
- When a local optimum is found, allow best bad move

Problem

- After bad move, best move might be going back
- So forbid reversing previous moves

Principle: Tabu list

- Keeps a list of used moves and solutions
- Usually not as complete solutions
- Rather moves and attributes

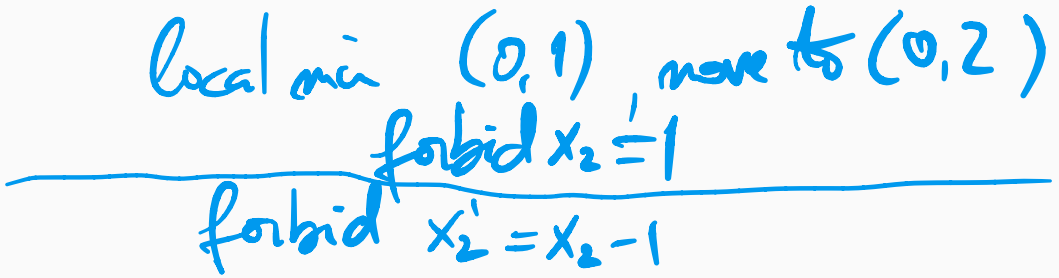
Tabu List

Tabu list

- Contains a list of forbidden moves
- Goal is to force exploration
- Takes a large amount of memory

Type of elements:

- Characteristics to avoid
- Transformations to avoid



Tabu List

Tabu list

- Contains a list of forbidden moves
- Goal is to force exploration
- Takes a large amount of memory

Type of elements:

- Characteristics to avoid
- Transformations to avoid

Different goals:

- Short-term: forbid recently considered solutions
- Intensification: intermediate rules to guide search
- Diversification: long-term rules to explore different regions

Tabu Search

Algorithm TabuSearch

Input: domain D, fitness f, number of rounds k

Output: solution $x \in D$

1: $x := \textsc{RandomSolution}(D)$
2: $T := \emptyset$
3: **for** $i = 1$ to k **do**
4: $\quad N := \textsc{Neighbors}(x, D)$
5: $\quad N := \textsc{FilterTabu}(N, T)$
6: $\quad c := \text{argmin}\{f(y) : y \in N\}$
7: $\quad T := \textsc{UpdateTabu}(T, c)$
8: $\quad x := c$
9: **end for**
10: **return** x

Example: TSP

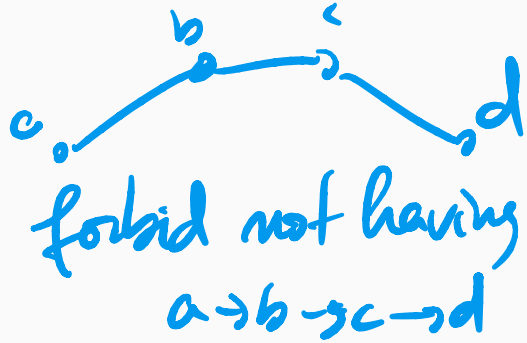
Operator: Switch two cities in the order

Short-term memory:

- Store solution
- Store the pair of cities switched

Intensification:

- Store long-lived sub-sequences
- Keep them during intensified search



Example: TSP

Operator: Switch two cities in the order

Short-term memory:

- Store solution
- Store the pair of cities switched

Intensification:

- Store long-lived sub-sequences
- Keep them during intensified search

Diversification:

- Store last time each vertex was switched
- Force these to be moved sometimes

Conclusion

Summary

Local search with random restart

- Introduces exploration by restarting from random solutions
- Greatly improves results
  - Even for comparable number of iterations

Simulated annealing

- Accept bad moves with some probability
  - Depending on the solution and temperature
- Bad moves allow escaping from local optima
- Good quality and also quite customizable

Tabu search

- Keeps a list of forbidden solutions
- To prevent cycling, but also to guide the search