Discrete Optimization

MA2827

Fondements de l'optimisation discrète

Approximate methods: local search

https://project.inria.fr/2015ma2827/

Material based on the lectures of Pascal Van Hentenryck at Coursera

Introduction

- Many important problems are provably not solvable in polynomial time (NP and harder)
- But, these results are based on the worst-case analysis
- Practical instances are often easier
- Approximate methods can often obtain good solutions

Approximate methods

- Local search (today)
- Relaxation
- Constraint programming

Outline

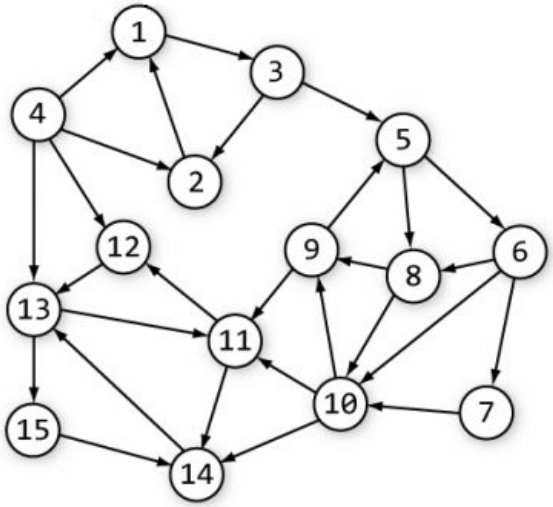
- Simple neighborhoods
  - queens on the chess board
  - warehouse location
- Travelling salesman problem
- Escaping local minima
  - randomization
  - tabu search
- Graph coloring
- Complex neighborhoods
  - sports scheduling
  - image segmentation

Local search

- Moves between configurations by performing local moves
- Works with complete assignments of the variables
- Optimization problems:
  - Start from a suboptimal configuration
  - Move towards better solutions
- Satisfaction problems:
  - Start from an infeasible configuration
  - Move towards feasibility
- No guarantees
- Can work great in practice!

Neighborhood

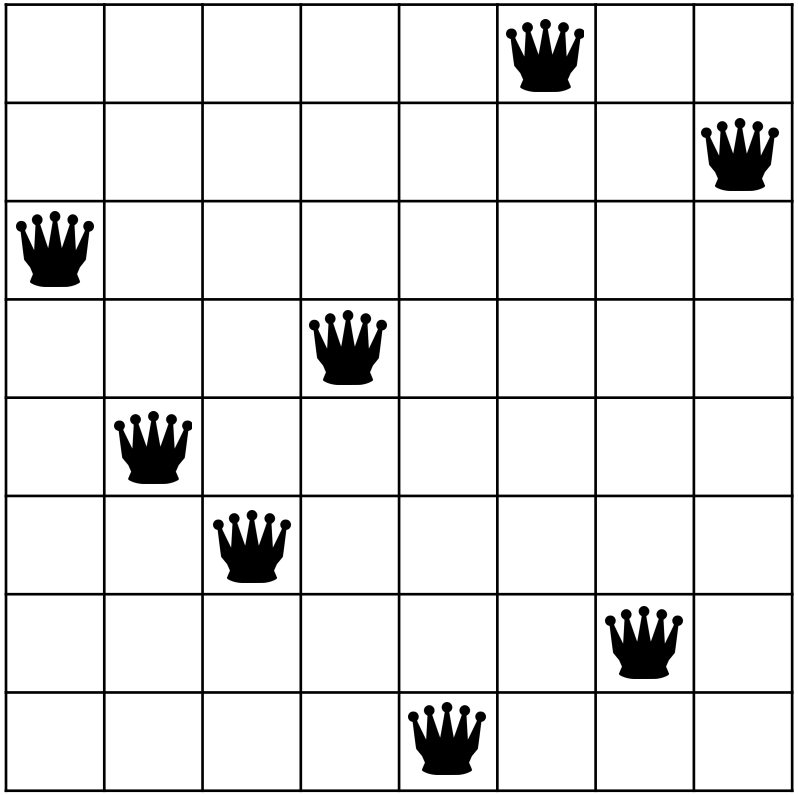
- Optimizing a function f
- Local moves define a neighborhood
  - Configuration that are close
  - N: C -> 2^C
- Local search is a graph exploration



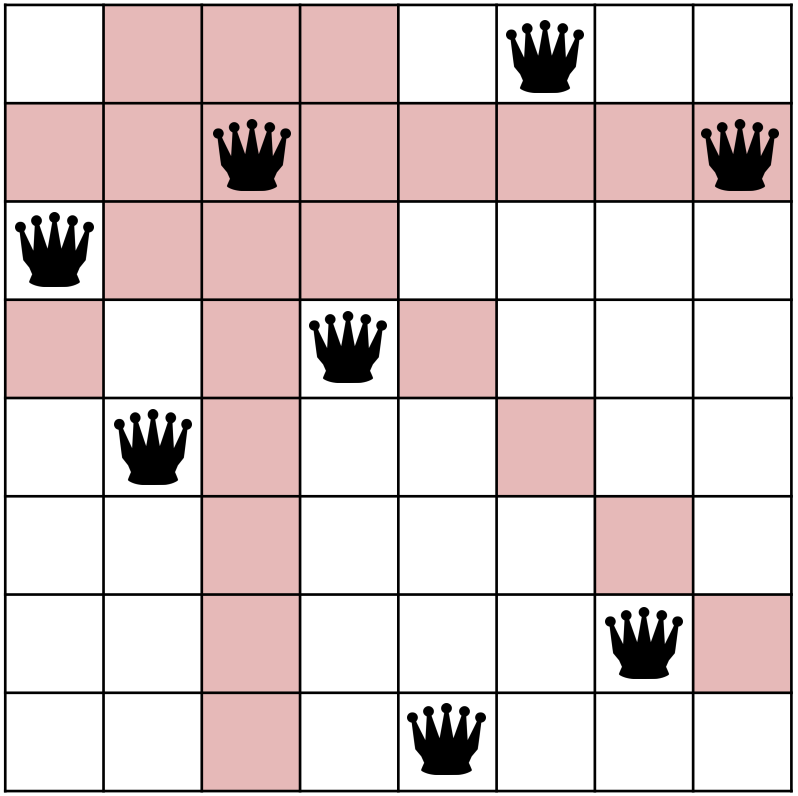
Example: satisfaction problem

Task: place 8 queens on the chess board such that they do not attack each other

Good



Bad



Example: satisfaction problem

Task: place 8 queens on the chess board such that they do not attack each other

Representation: the row of the queen in each column

Local steps: move one of the queens in its column

Example: satisfaction problem

Task: place 8 queens on the chess board such that they do not attack each other

Representation: the row of the queen in each column

Local steps: move one of the queens in its column

Local search:

– start with infeasible configuration

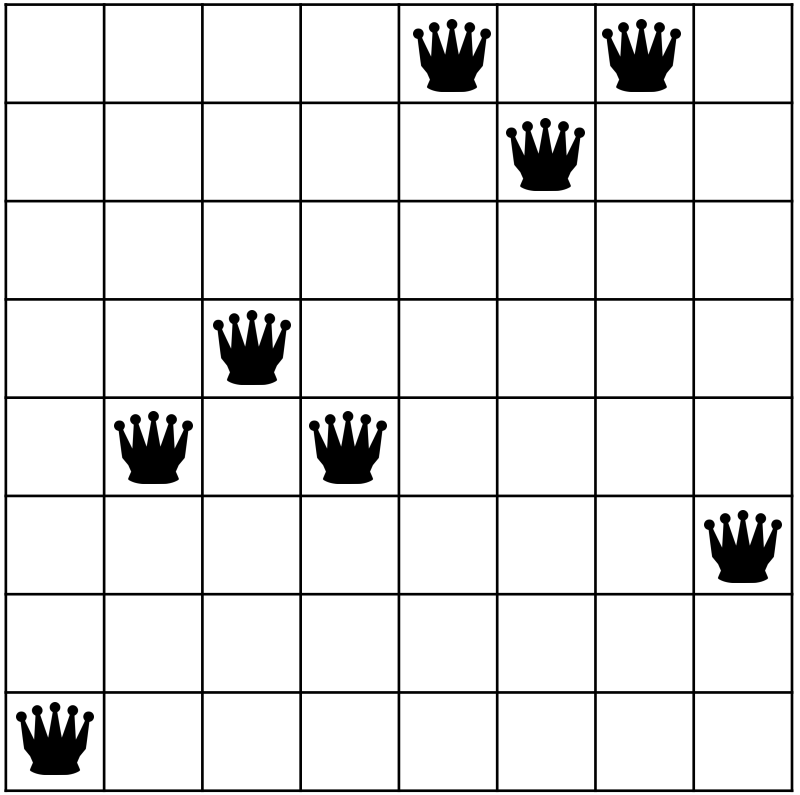
– move towards feasibility

8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Max/min conflict:

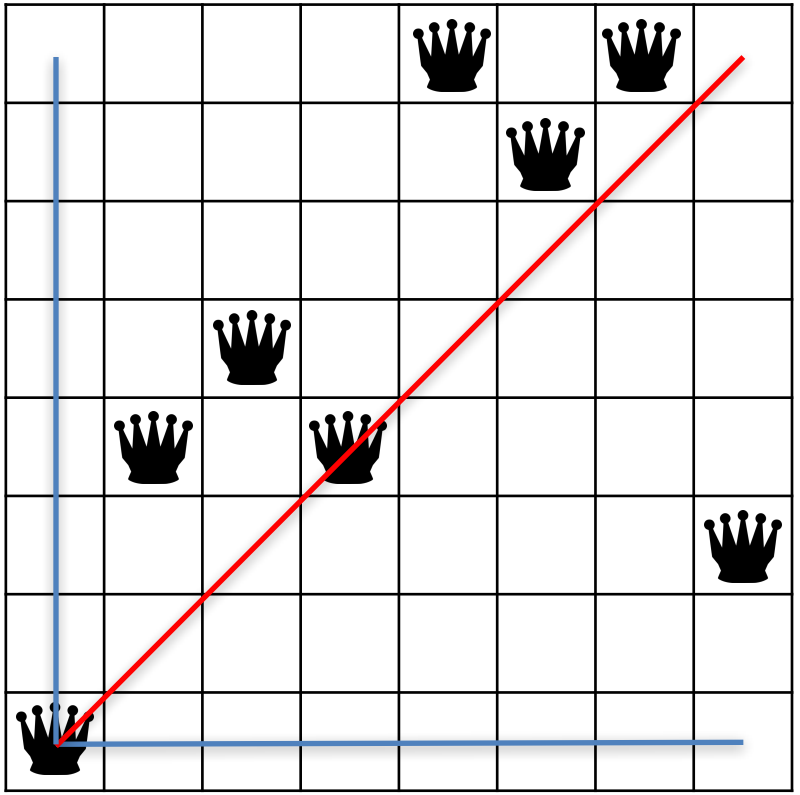
Count constraint violations



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

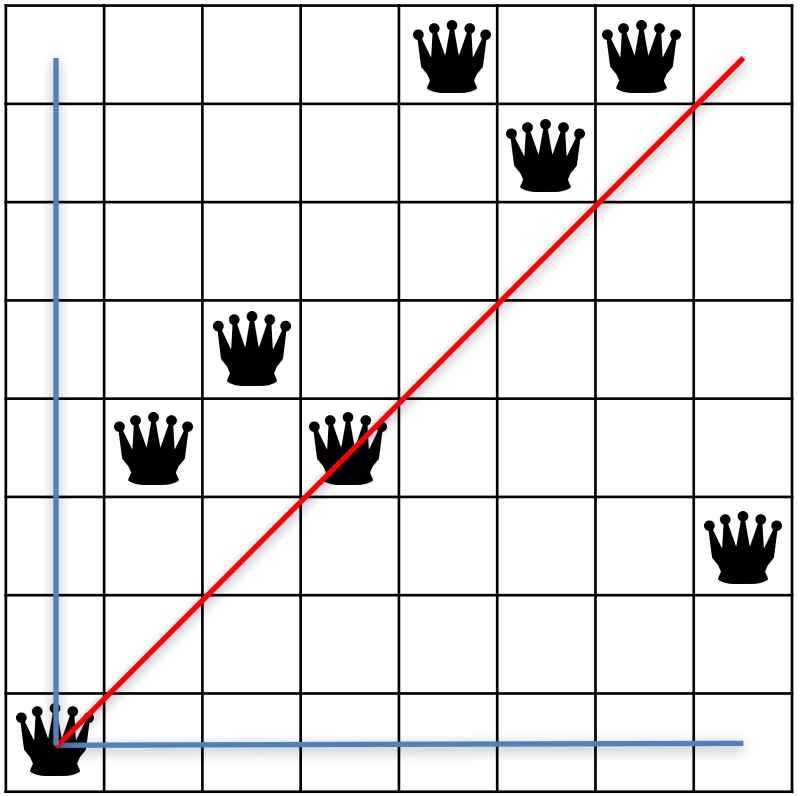
Count constraint violations



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

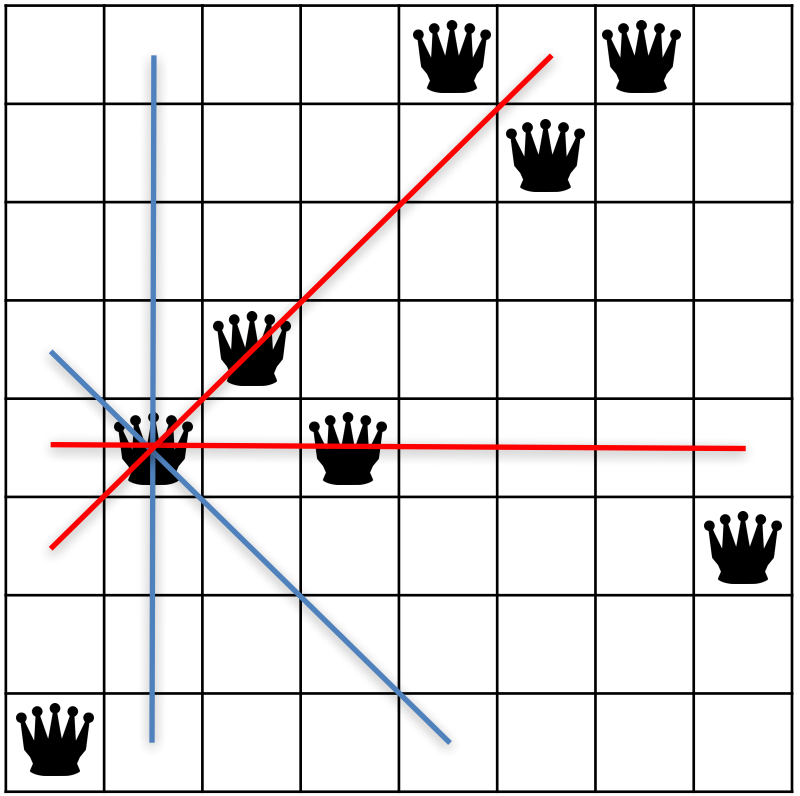


1

8 queens

Task: place 8 queens on the chess board such that they do not attack each other

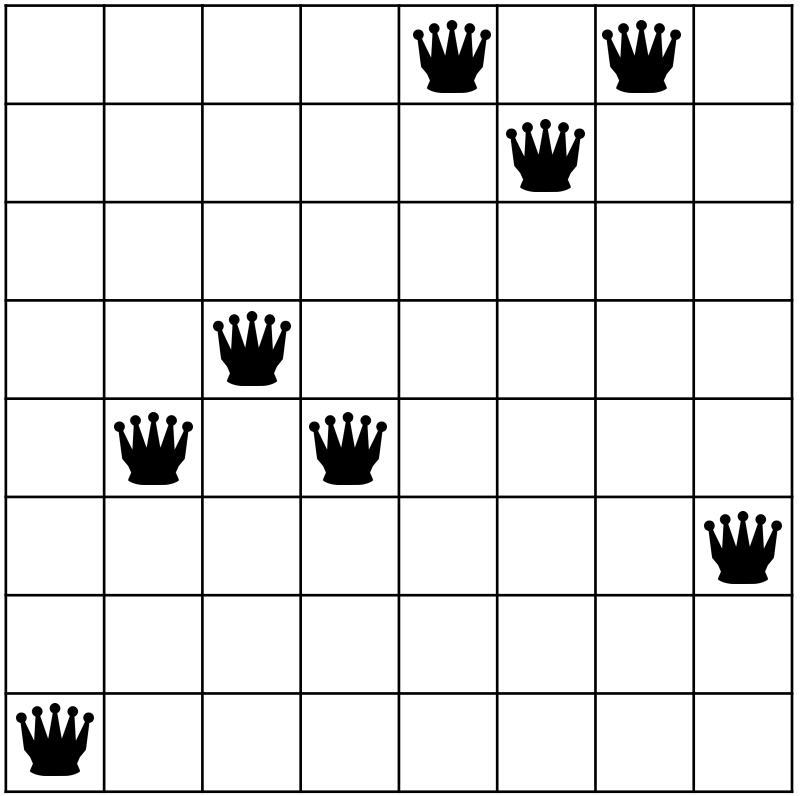
Count constraint violations



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

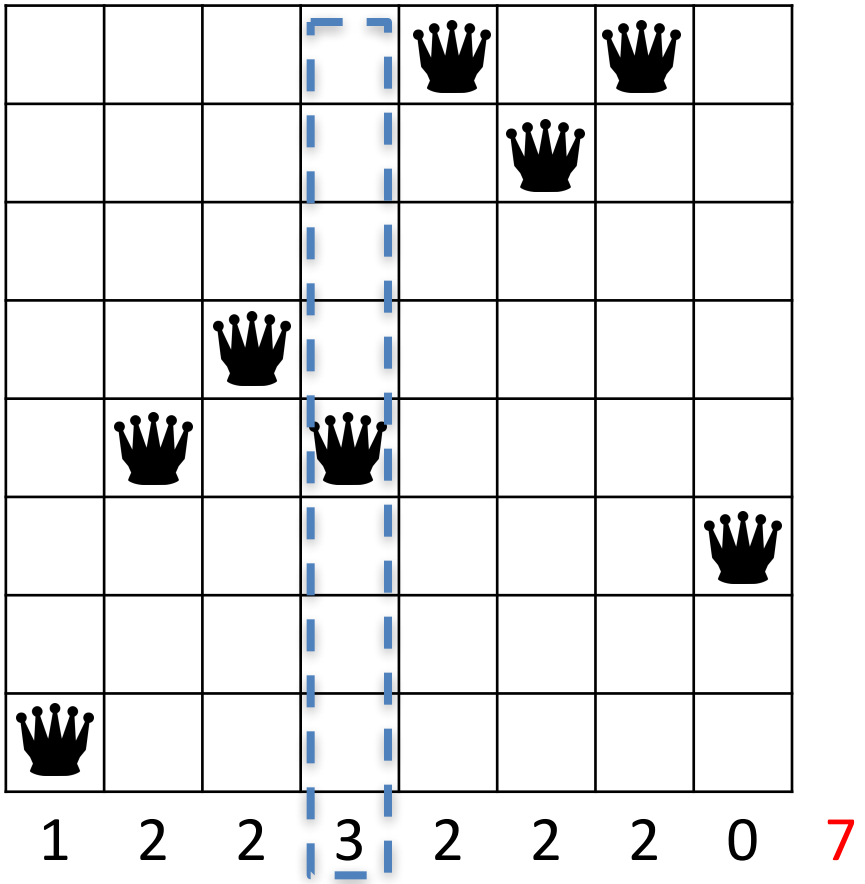


8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations



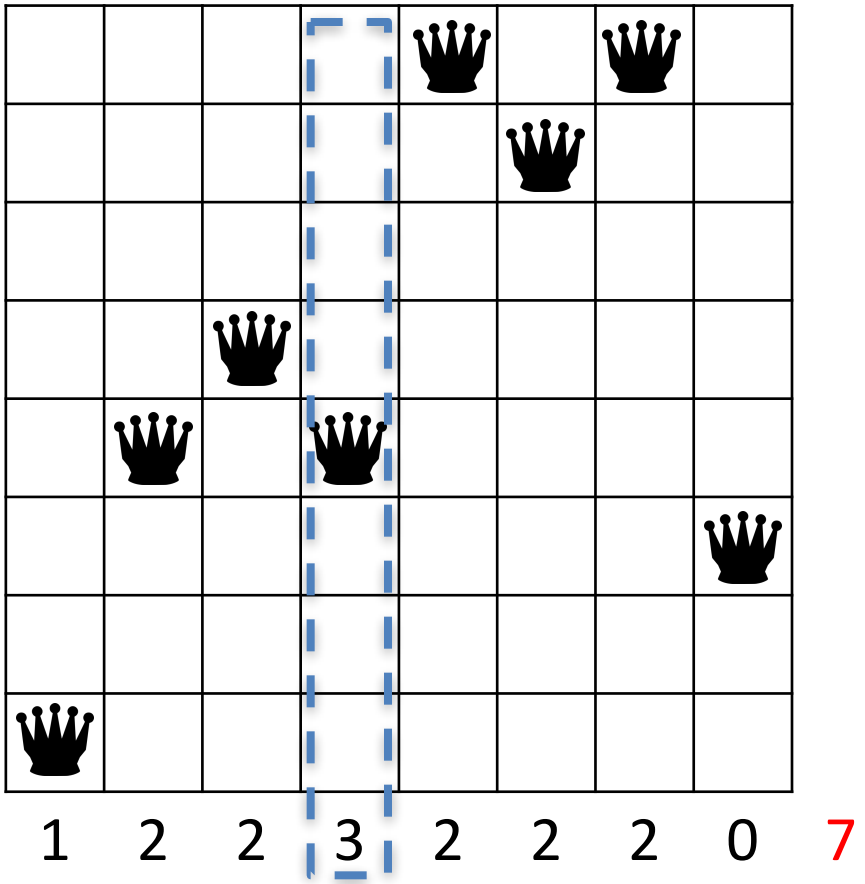
8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations

Possible moves



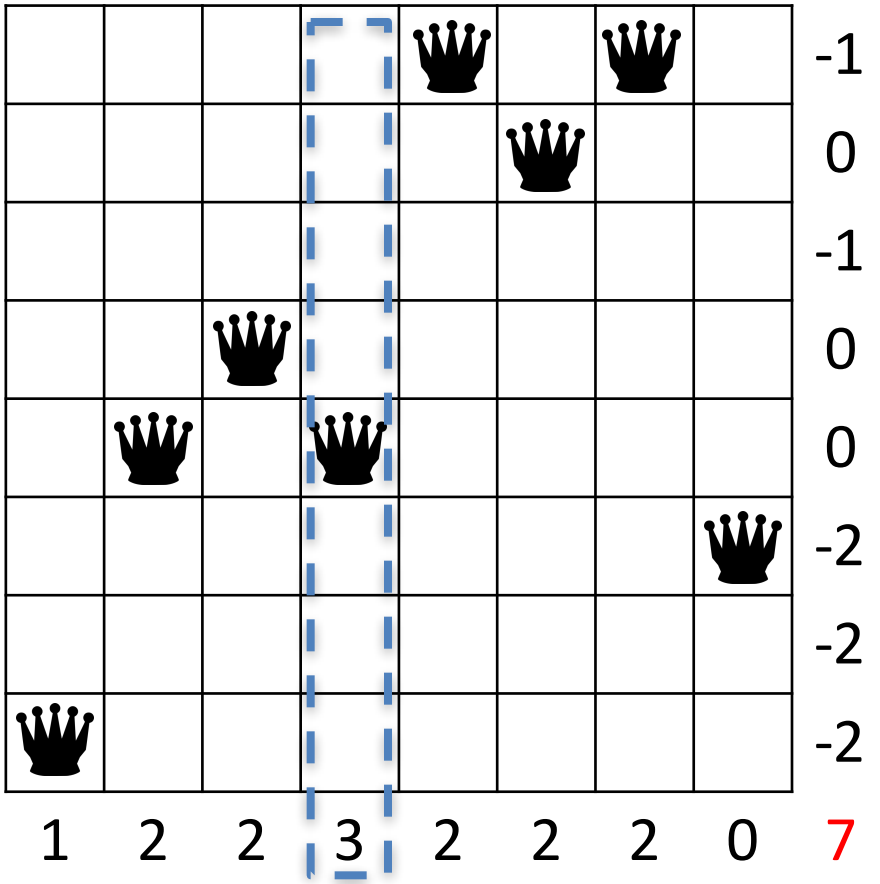
8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations

Possible moves



8 queens

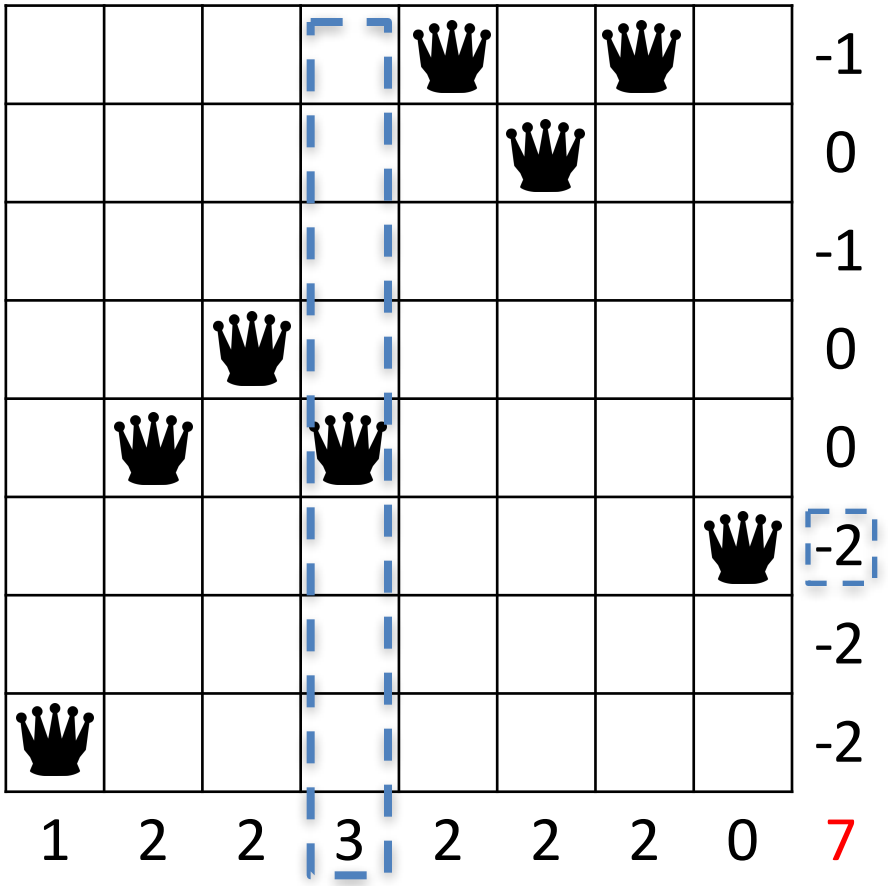
Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations

Possible moves

Find the best move



8 queens

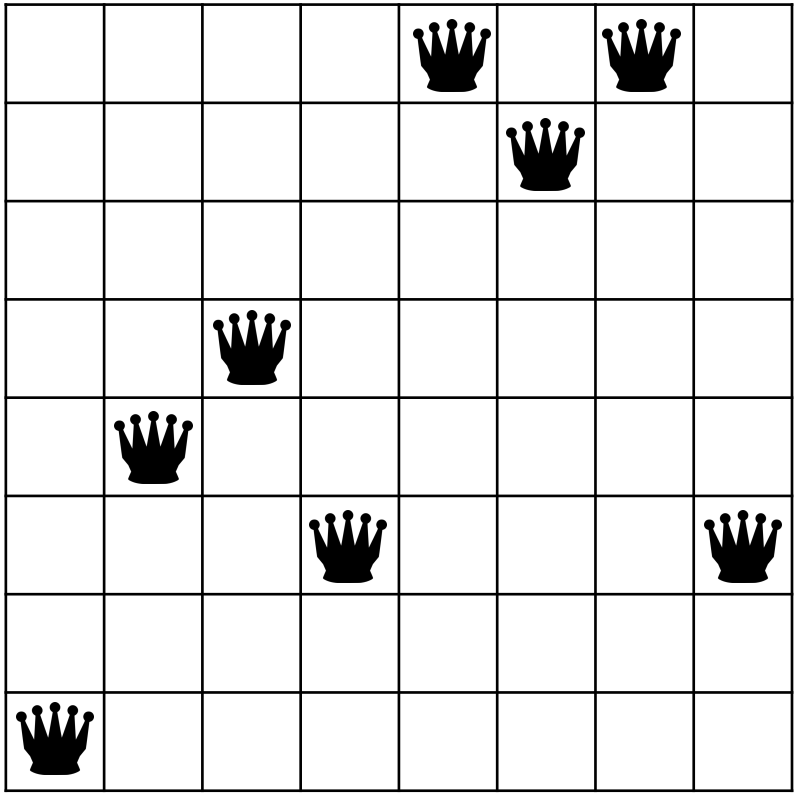
Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations

Possible moves

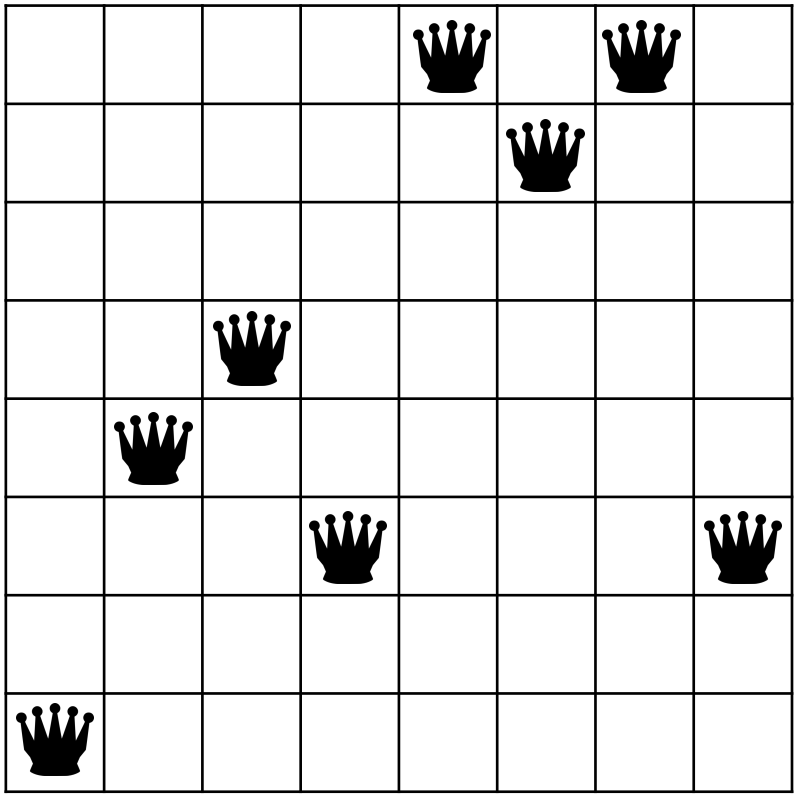
Find the best move



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

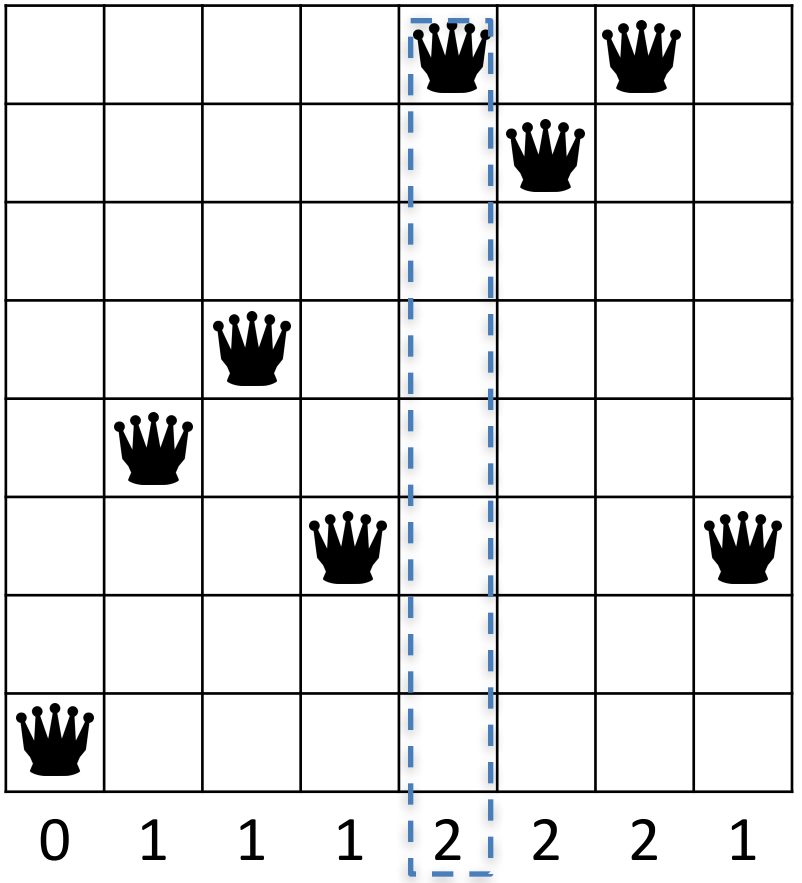


8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations



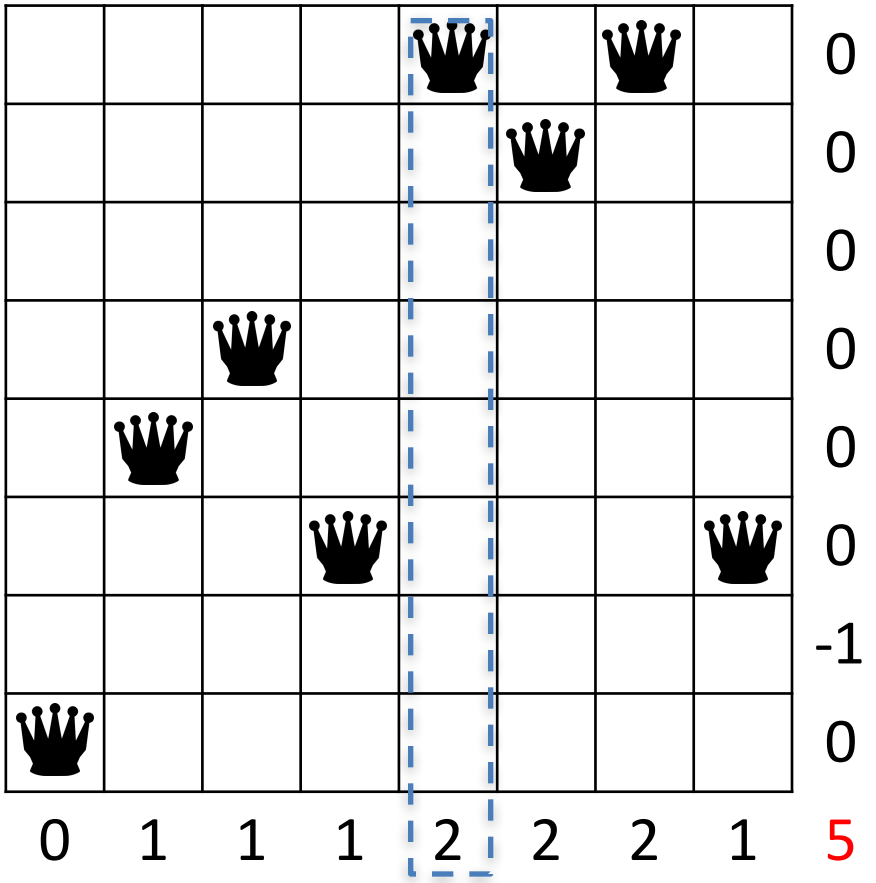
8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations

Possible moves



8 queens

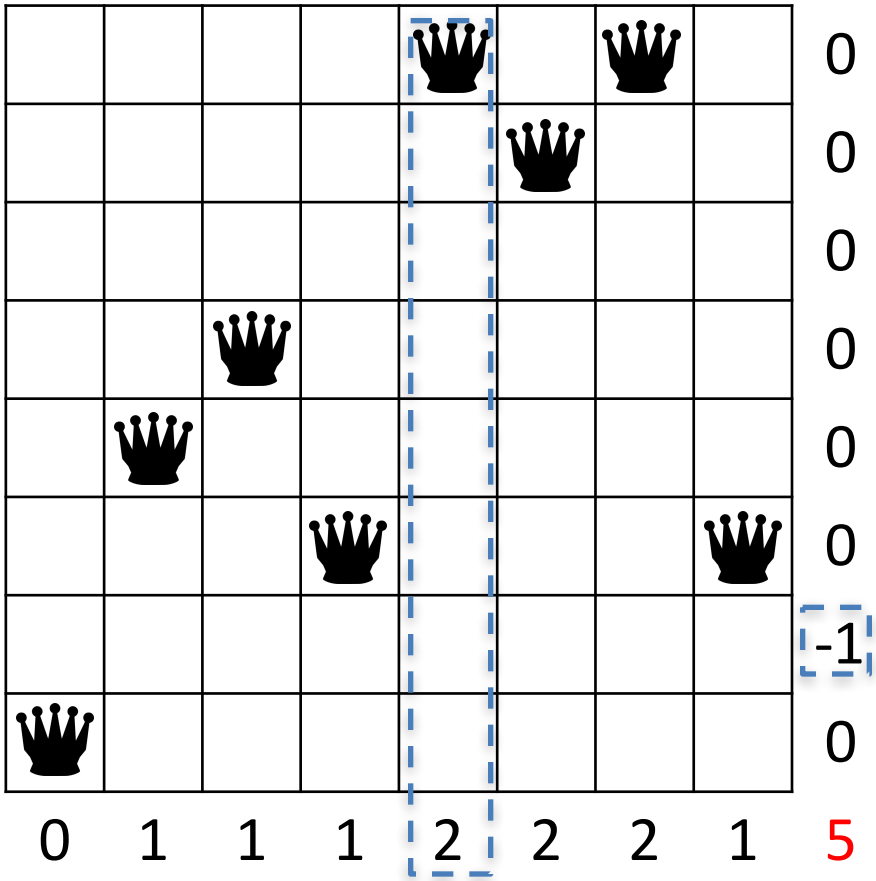
Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations

Possible moves

Find the best move



8 queens

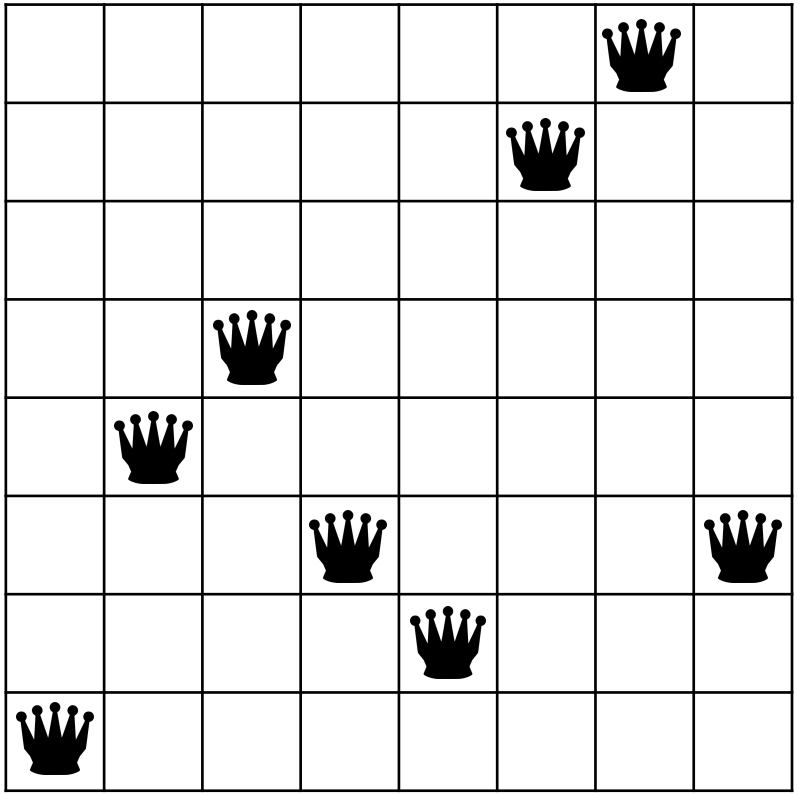
Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations

Possible moves

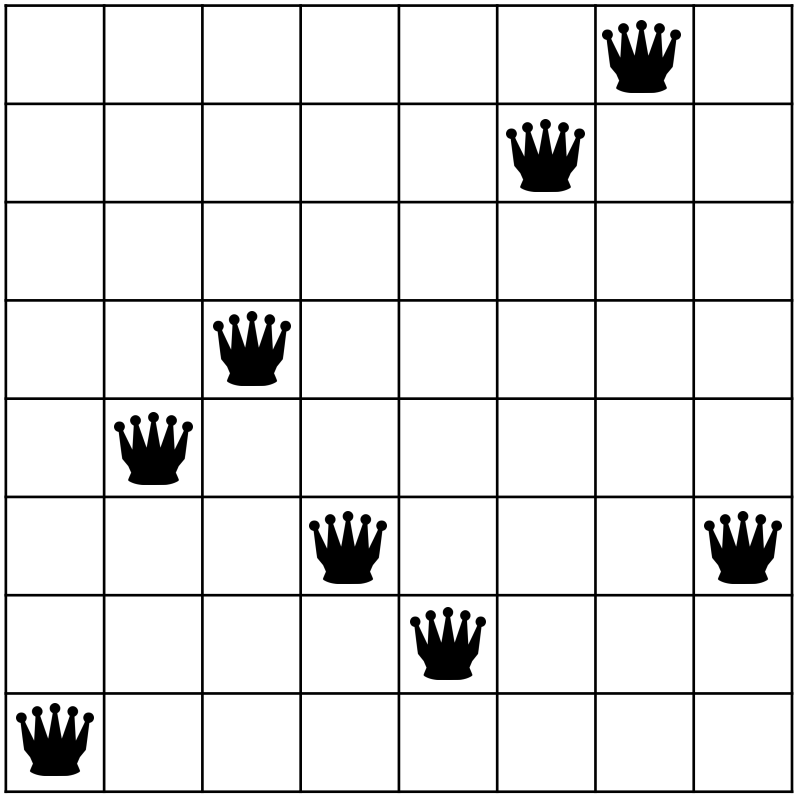
Find the best move



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

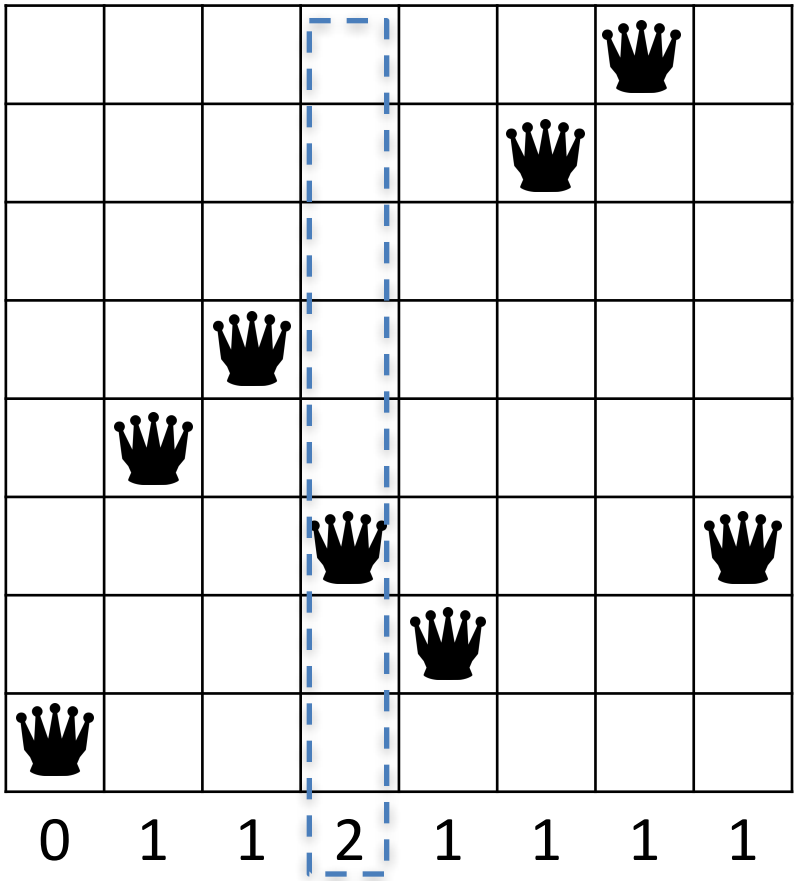


8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations



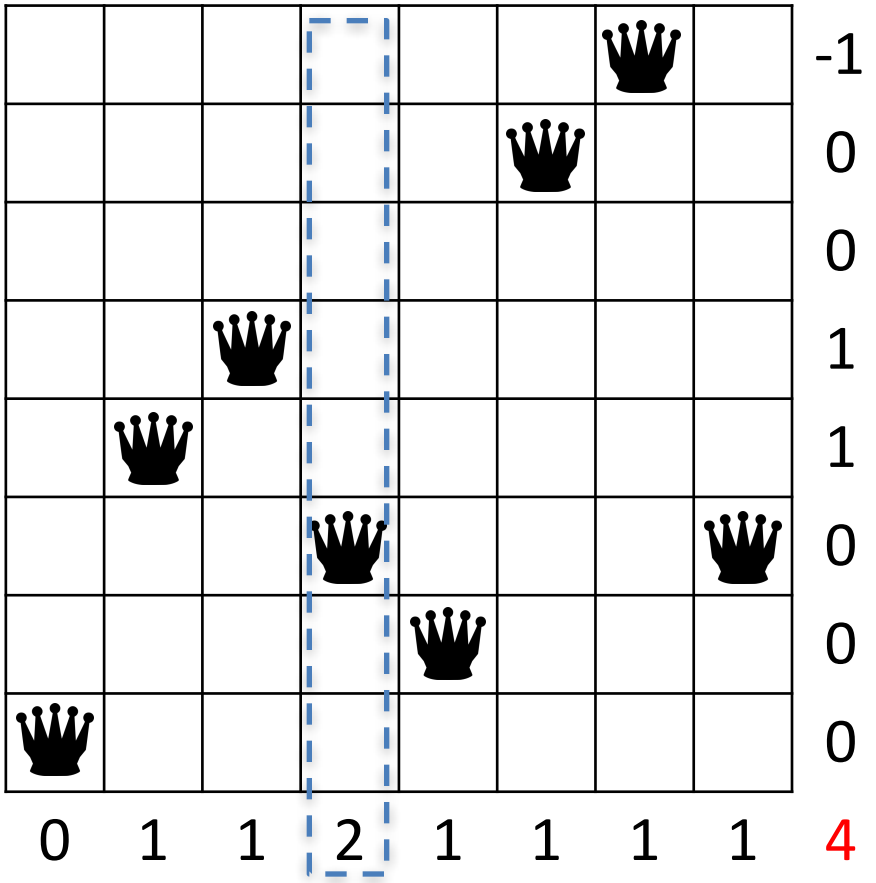
8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations

Possible moves



8 queens

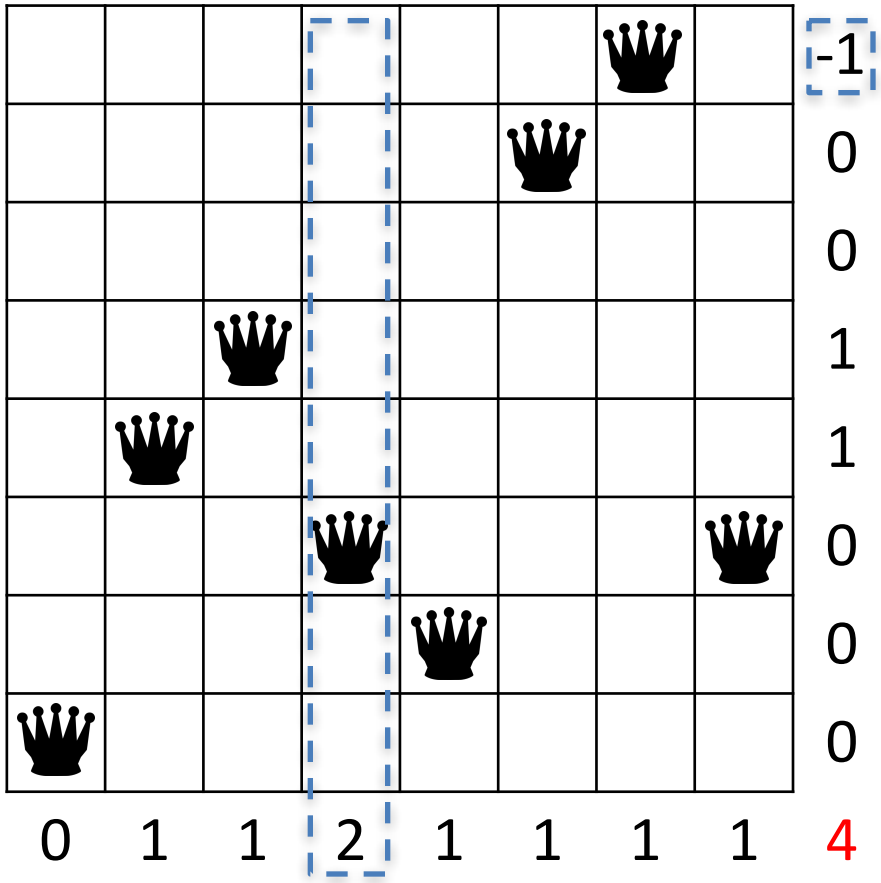
Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations

Possible moves

Find the best move



8 queens

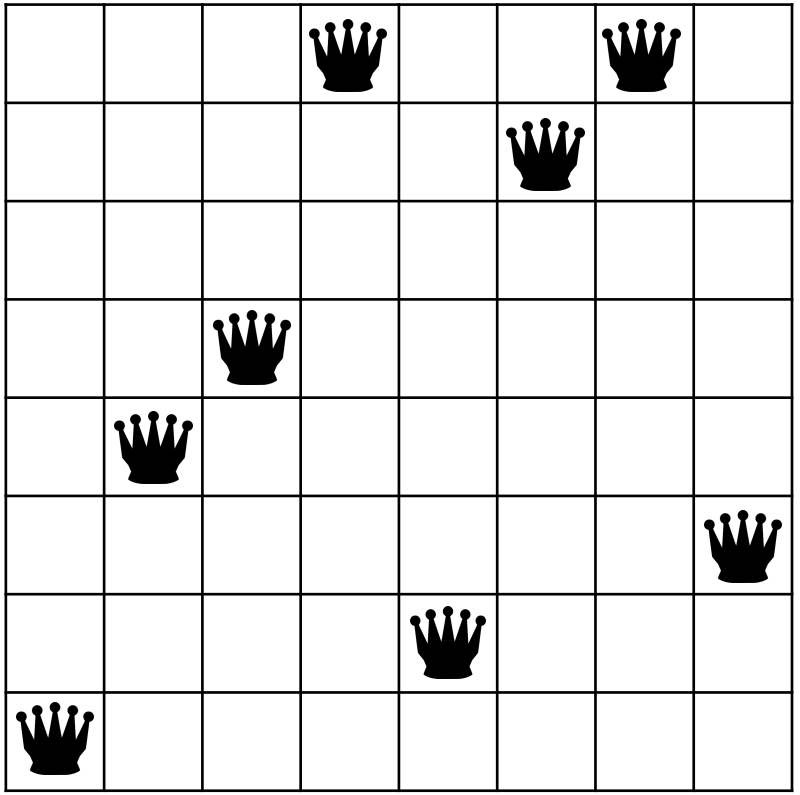
Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations

Possible moves

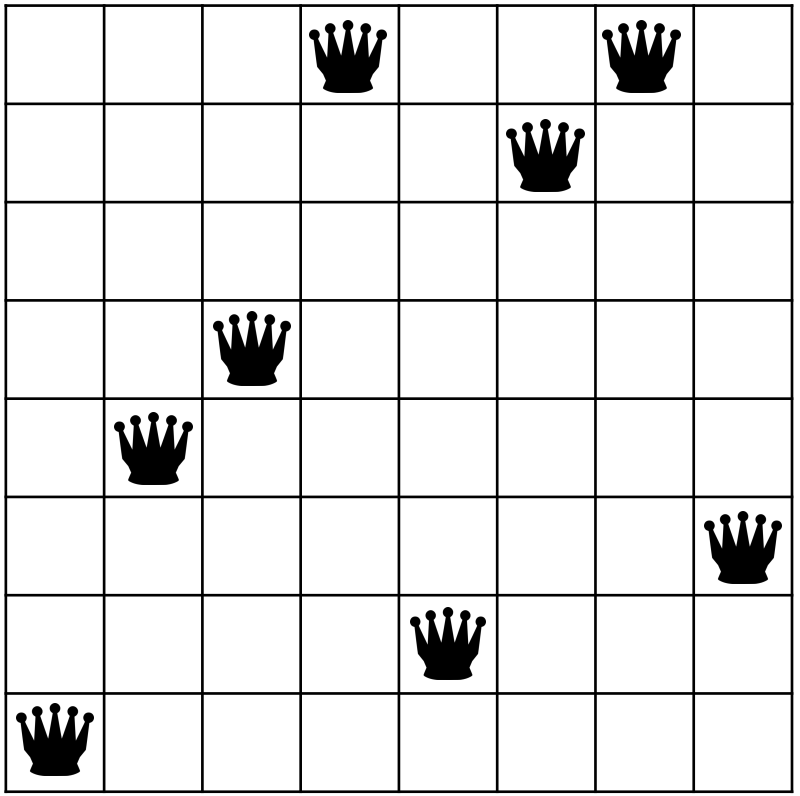
Find the best move



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

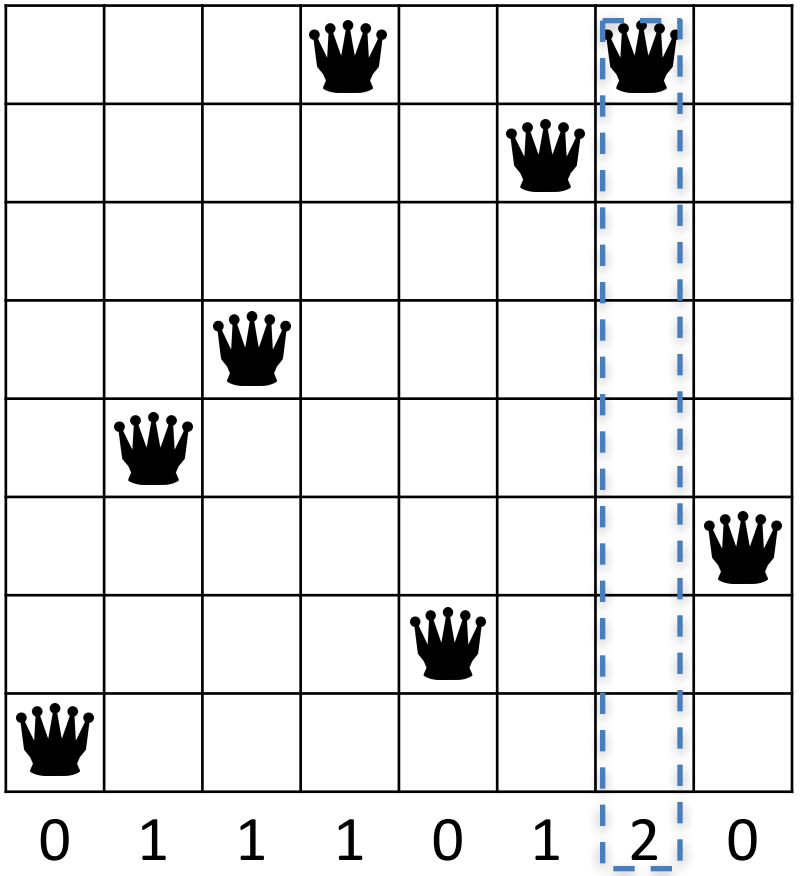


8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations



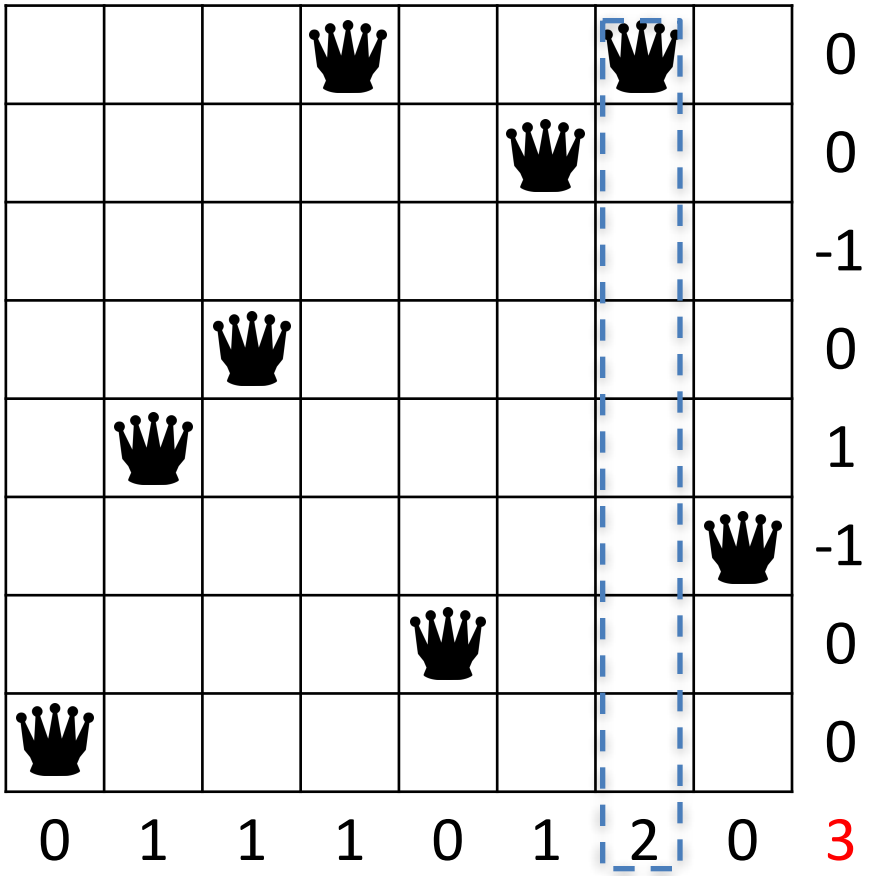
8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations

Possible moves



8 queens

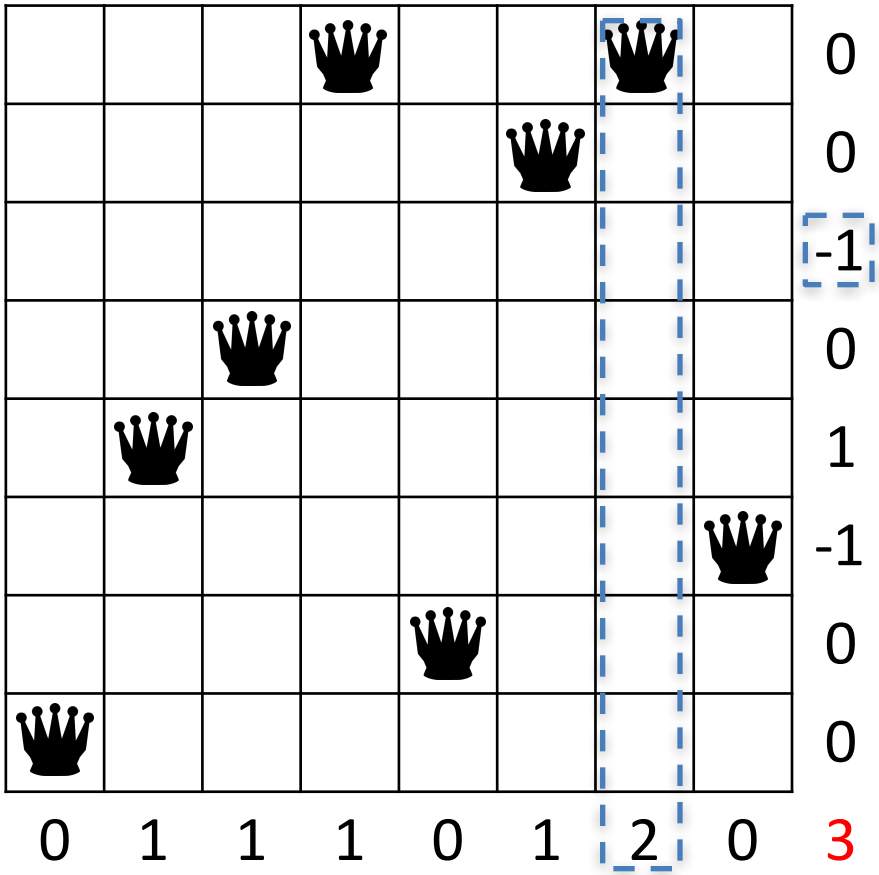
Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations

Possible moves

Find the best move



8 queens

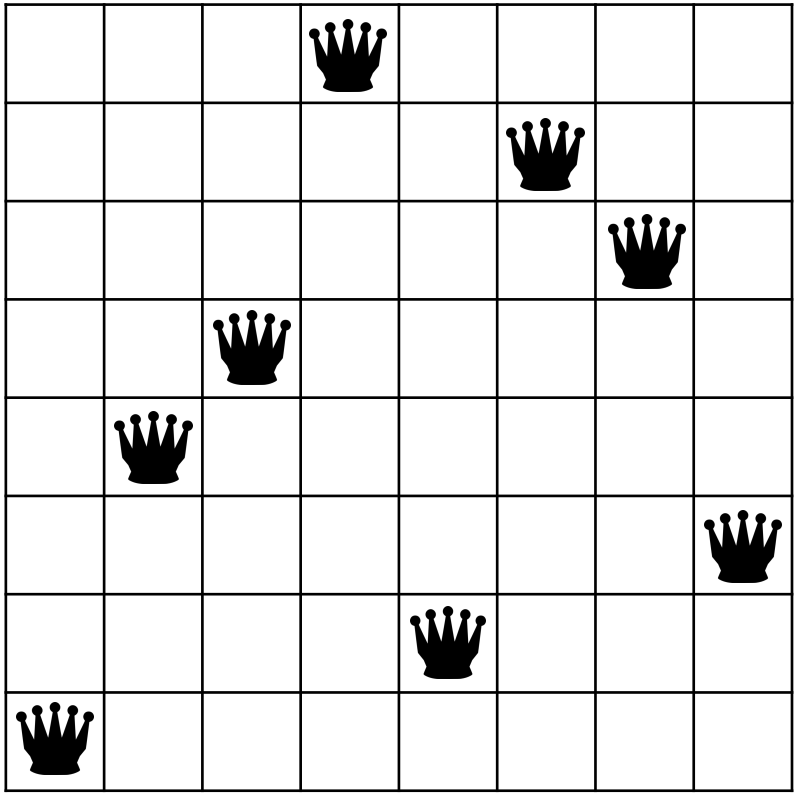
Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Queen with most violations

Possible moves

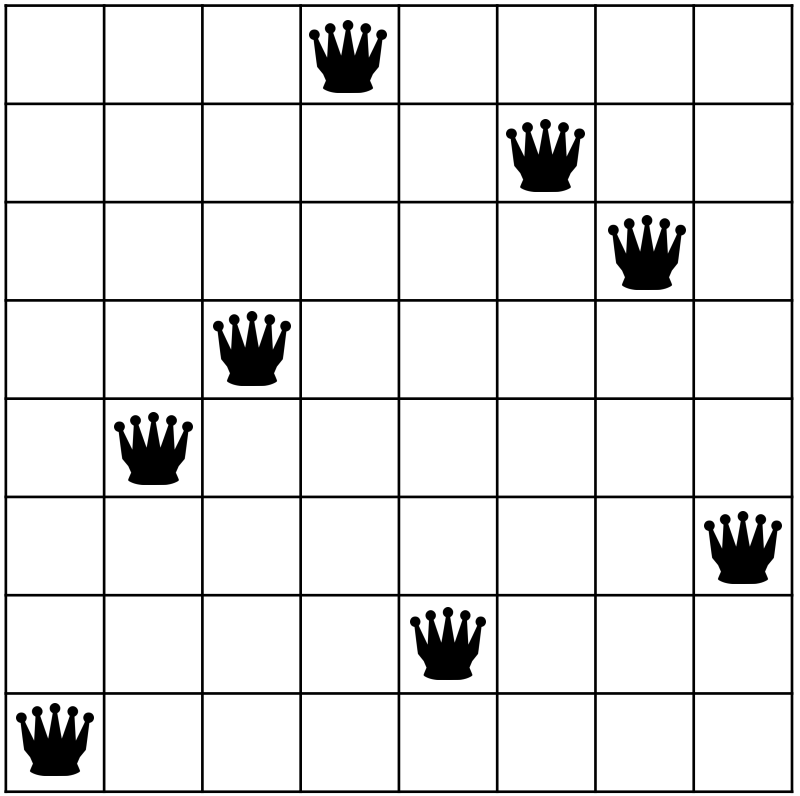
Find the best move



8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

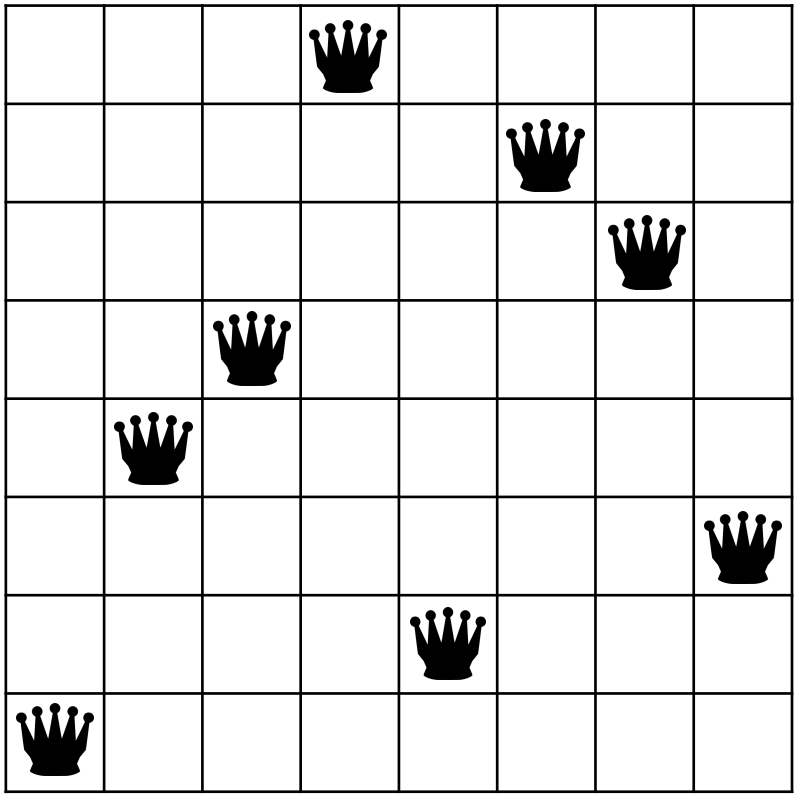


8 queens

Task: place 8 queens on the chess board such that they do not attack each other

Count constraint violations

Local minimum!



Local search: no guarantees

Local minima:

no step improves the objective

No guarantees for global optimality

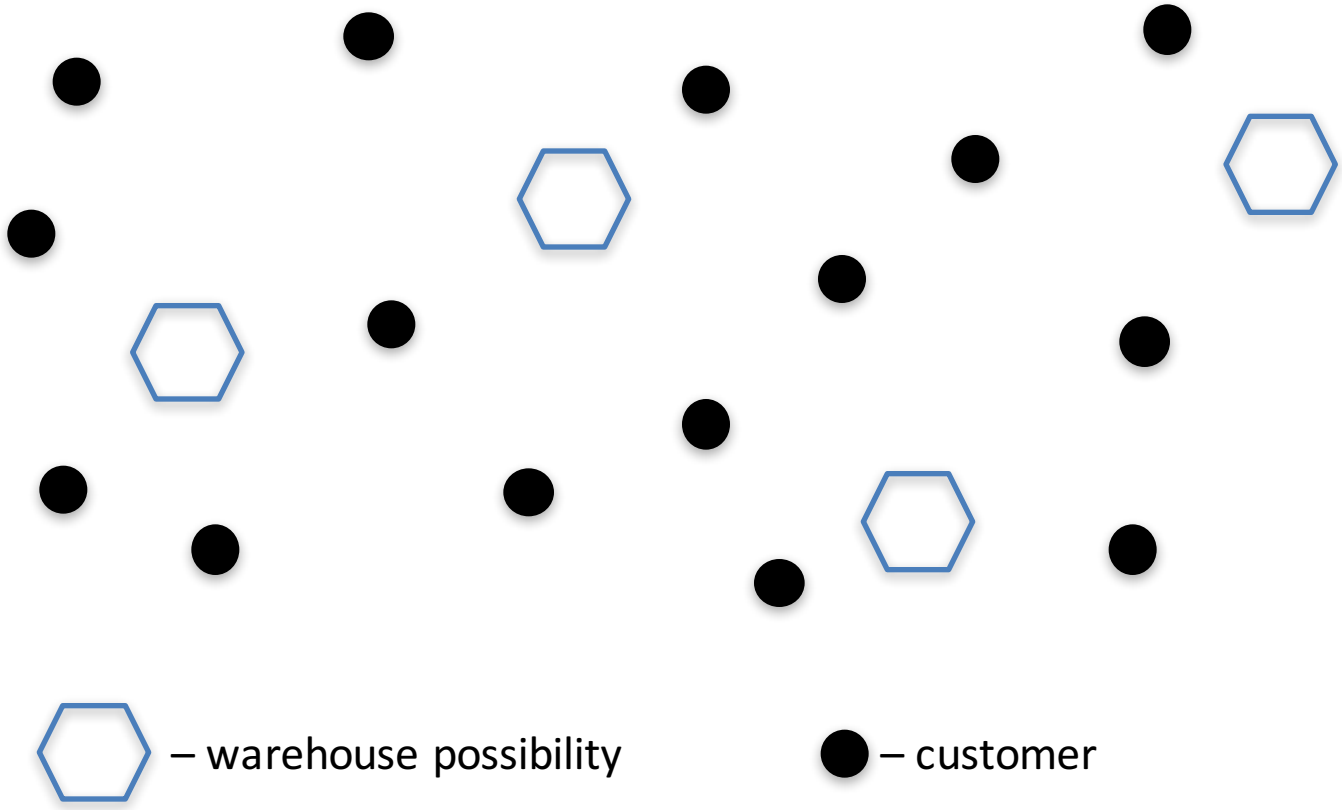
Escaping local optima is a critical issue in local search

Example: optimization problem

Task: where to place warehouses?

– Cost for opening warehouse at position w: f_w

– Cost for transportation from w to c: $t_{w,c}$

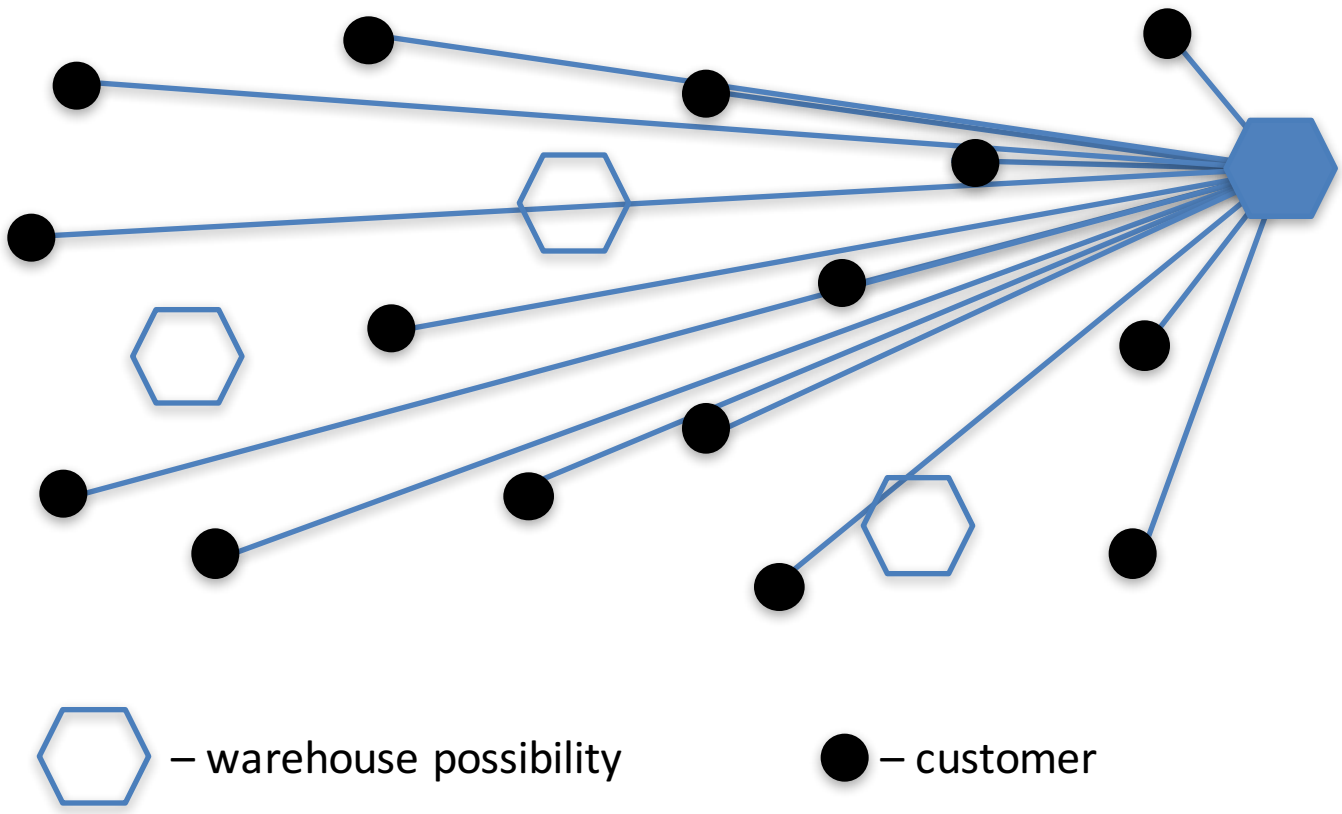


Warehouse location

Task: where to place warehouses?

– Cost for opening warehouse at position w: f_w

– Cost for transportation from w to c: $t_{w,c}$

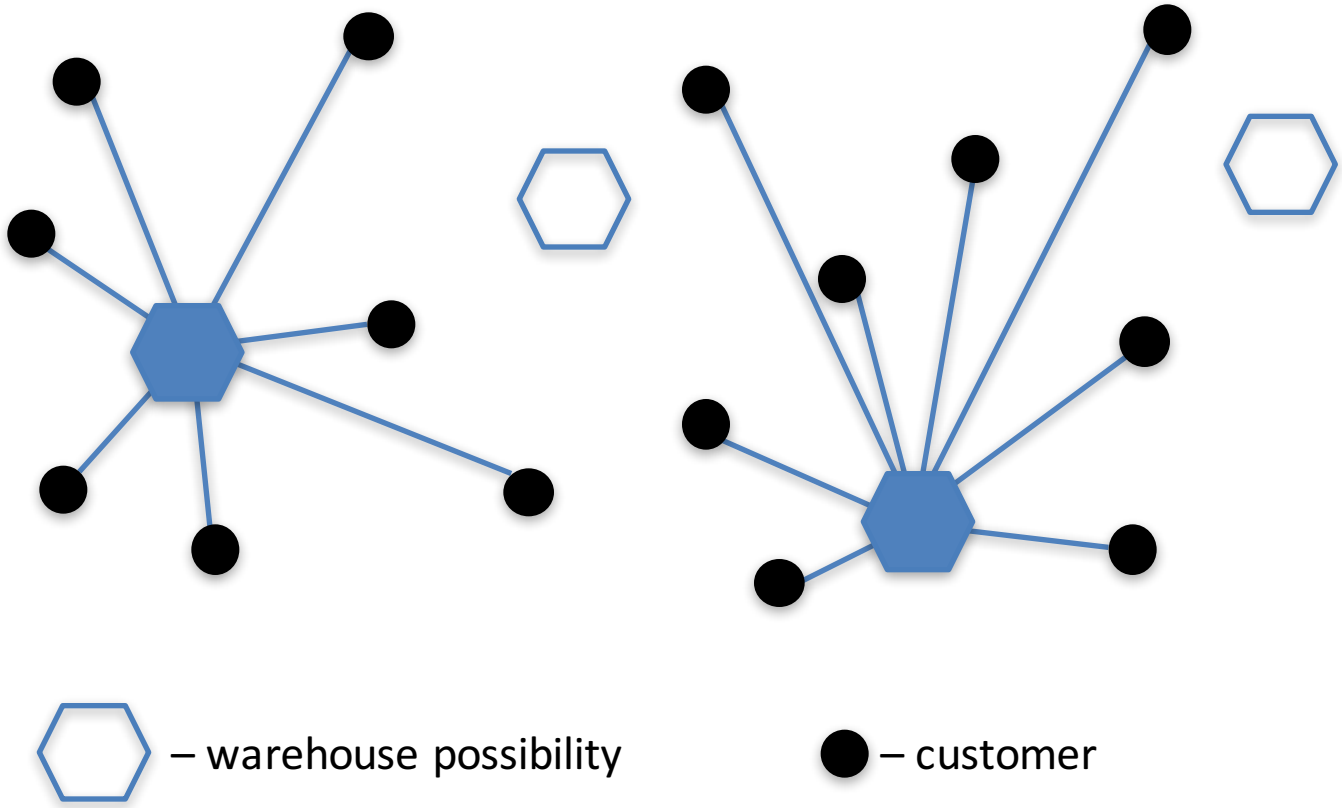


Warehouse location

Task: where to place warehouses?

– Cost for opening warehouse at position w: f_w

– Cost for transportation from w to c: $t_{w,c}$



Warehouse location

Task: where to place warehouses? (input: f_w , $t_{w,c}$)

Decision variables:

- $o_w \in \{0, 1\}$: whether warehouse w is open
- a[c] $\in \{1, \ldots, W\}$: the warehouse assigned to customer c

Objective:

$$\min \quad \sum_{w \in W} f_w o_w + \sum_{c \in C} t_{a[c],c}$$

Constraints:

customers can be assigned only to open warehouses

Warehouse location

Task: where to place warehouses? (input: f_w , $t_{w,c}$)

Key observation:

- once the warehouse locations have been chosen, the problem is easy
- it's enough to assign customers to warehouses minimizing the transportation costs

Objective:

$$\min \quad \sum_{w \in W} f_w o_w + \sum_{c \in C} \min_{w \in W : o_w = 1} t_{w,c}$$

Warehouse location

Task: where to place warehouses? (input: f_w , $t_{w,c}$)

Neighborhood:

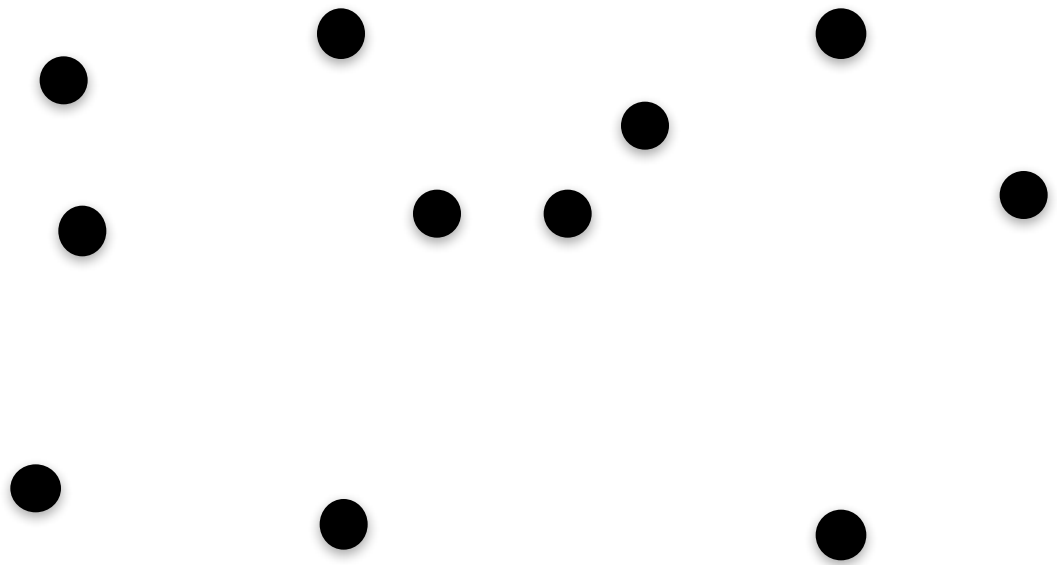
- open/close warehouses (flip the value of o_w)
- swap warehouses (close one and open the other)
- simultaneous swap of k warehouses

Travelling salesman problem (TSP)

Task: find the shortest path to visit all cities exactly once

– Hamiltonian cycle in a graph

– simplification: cities are points in 2D

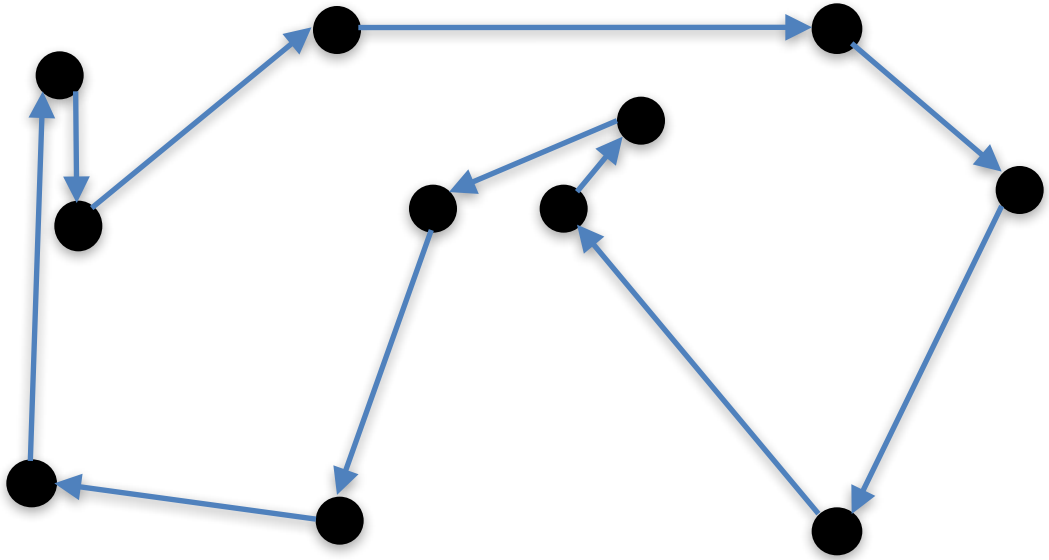


Travelling salesman problem (TSP)

Task: find the shortest path to visit all cities exactly once

– Hamiltonian cycle in a graph

– simplification: cities are points in 2D

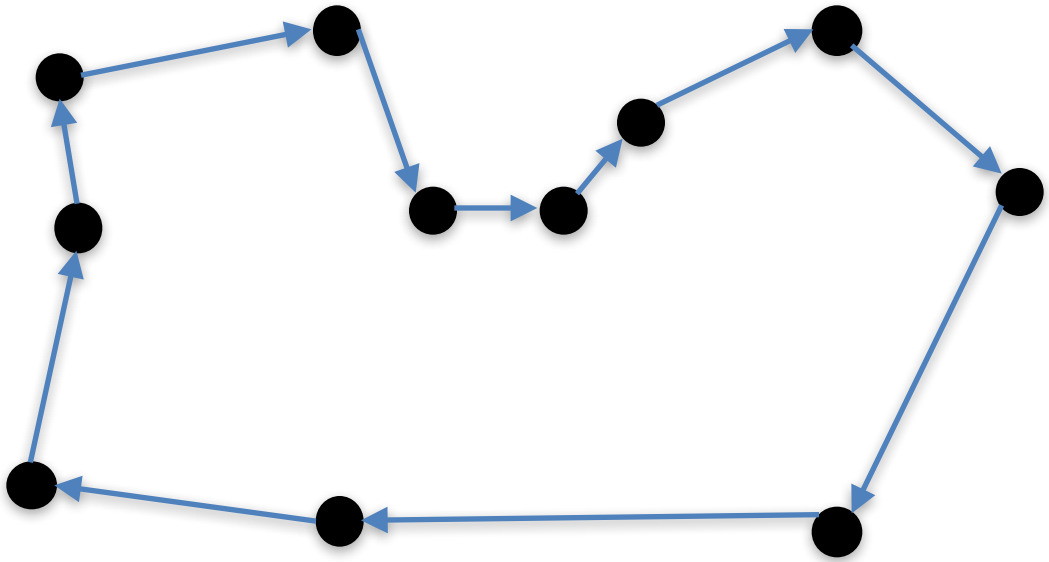


Travelling salesman problem (TSP)

Task: find the shortest path to visit all cities exactly once

– Hamiltonian cycle in a graph

– simplification: cities are points in 2D



Travelling salesman problem (TSP)

Task: find the shortest path to visit all cities exactly once

– Hamiltonian cycle in a graph

– simplification: cities are points in 2D

Decision variables:

where to go next after every city

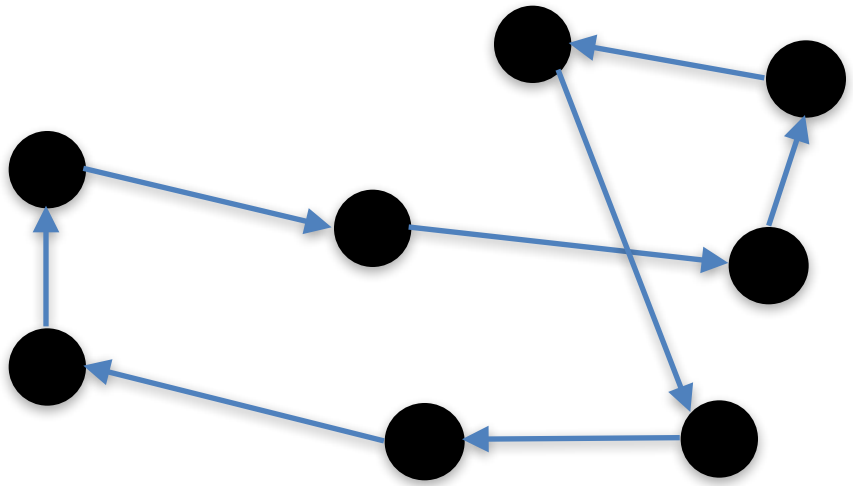
TSP if probably the most well-studied combinatorial problem!

Local search for TSP: 2-OPT

Task: find the shortest path to visit all cities exactly once

Local move:

select two edges and replace them by two other edges

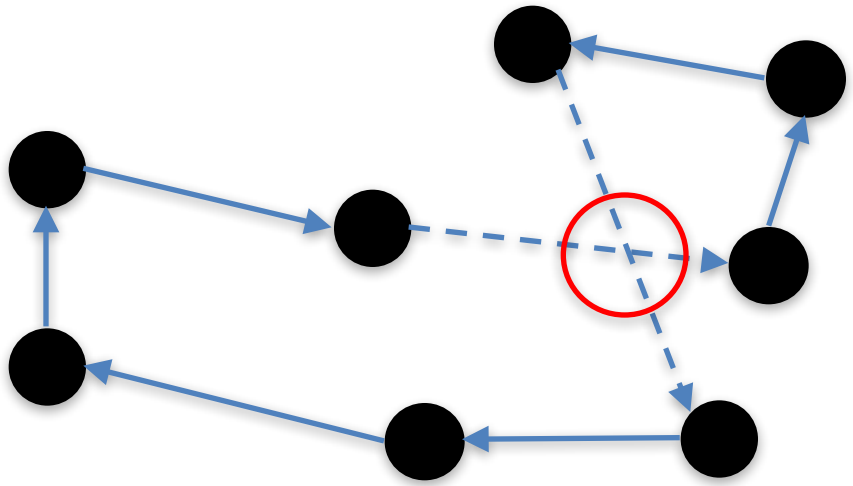


Local search for TSP: 2-OPT

Task: find the shortest path to visit all cities exactly once

Local move:

select two edges and replace them by two other edges



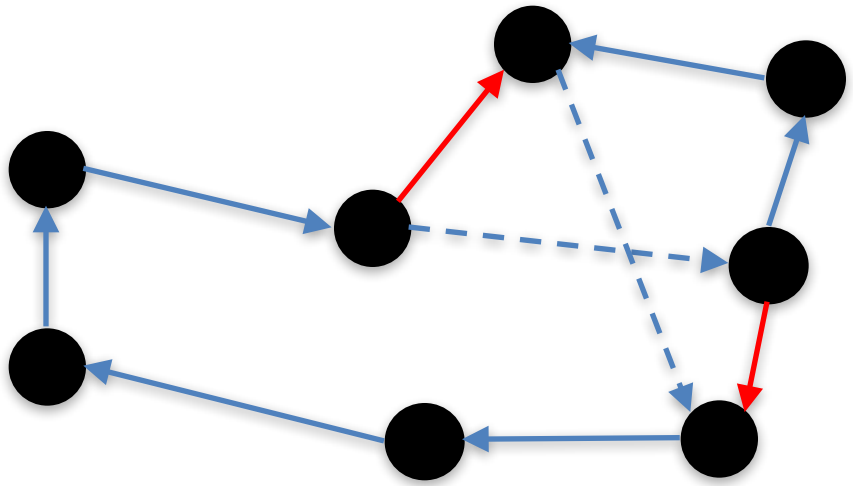
Crossings are bad!

Local search for TSP: 2-OPT

Task: find the shortest path to visit all cities exactly once

Local move:

select two edges and replace them by two other edges

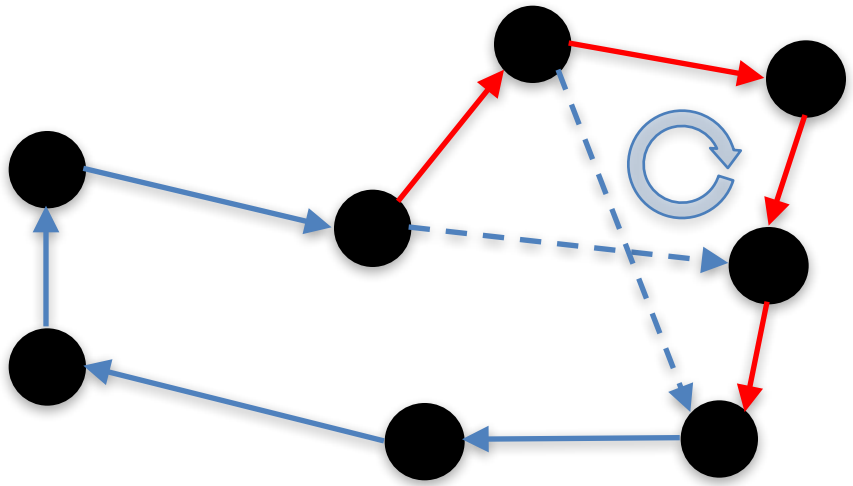


Local search for TSP: 2-OPT

Task: find the shortest path to visit all cities exactly once

Local move:

select two edges and replace them by two other edges

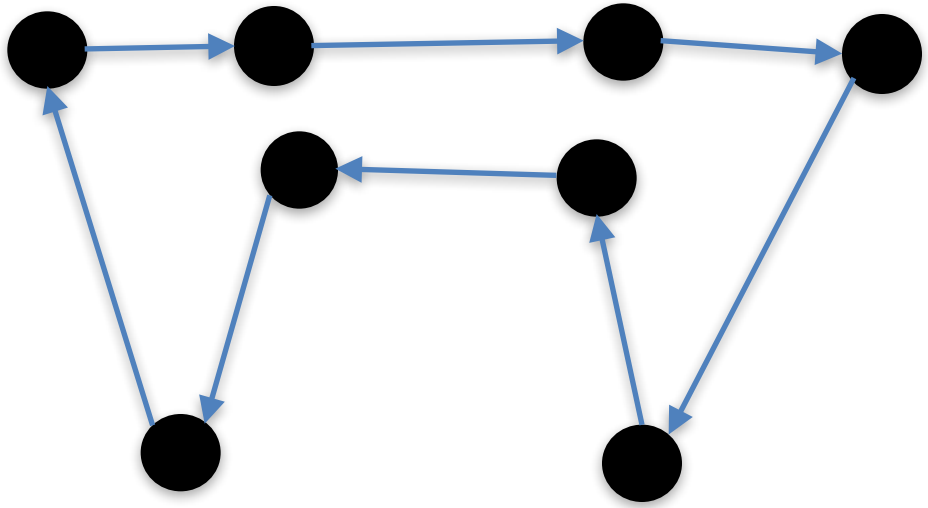


Local search for TSP: 3-OPT

Task: find the shortest path to visit all cities exactly once

Local move:

select three edges and replace them by three other edges

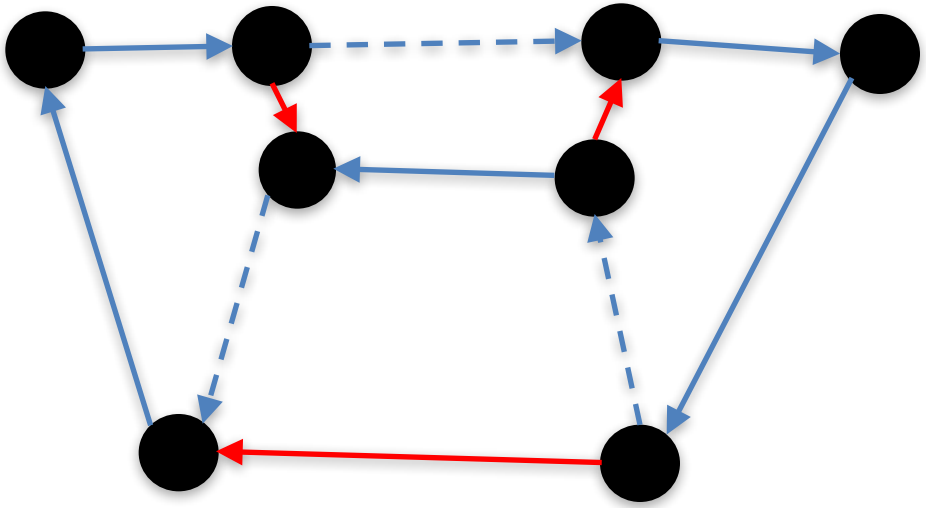


Local search for TSP: 3-OPT

Task: find the shortest path to visit all cities exactly once

Local move:

select three edges and replace them by three other edges

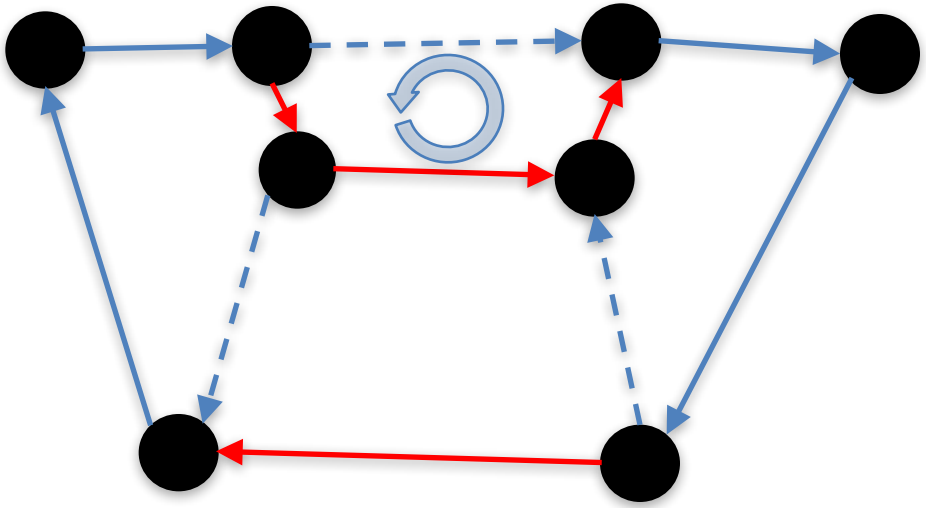


Local search for TSP: 3-OPT

Task: find the shortest path to visit all cities exactly once

Local move:

select three edges and replace them by three other edges



Local search for TSP

Task: find the shortest path to visit all cities exactly once

2-OPT:

– the neighborhood is a set of all tours that can be reached by swapping two edges

3-OPT:

– the neighborhood is a set of all tours that can be reached by swapping three edges

– much better than 2-OPT in quality but more expensive

4-OPT?

– marginally better but even more expensive

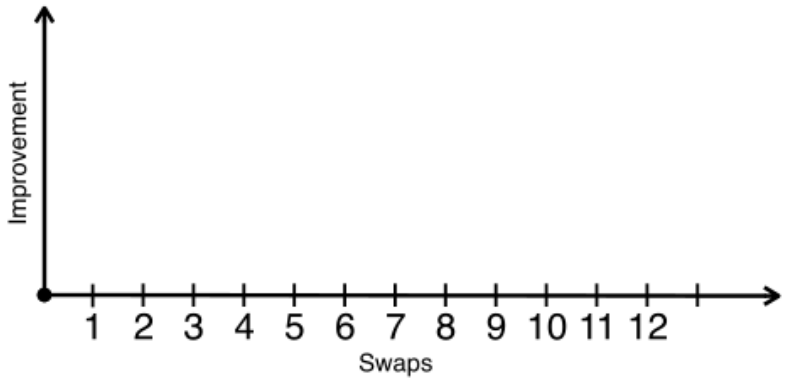
Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

K-OPT:

– replace the notion of one favorable swap by a sequence of favorable swaps

– do not search for the entire set of sequences but build them incrementally



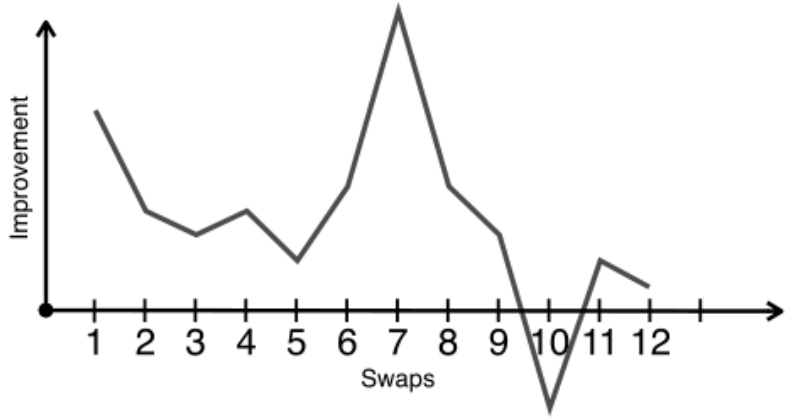
Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

K-OPT:

– replace the notion of one favorable swap by a sequence of favorable swaps

– do not search for the entire set of sequences but build them incrementally



Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

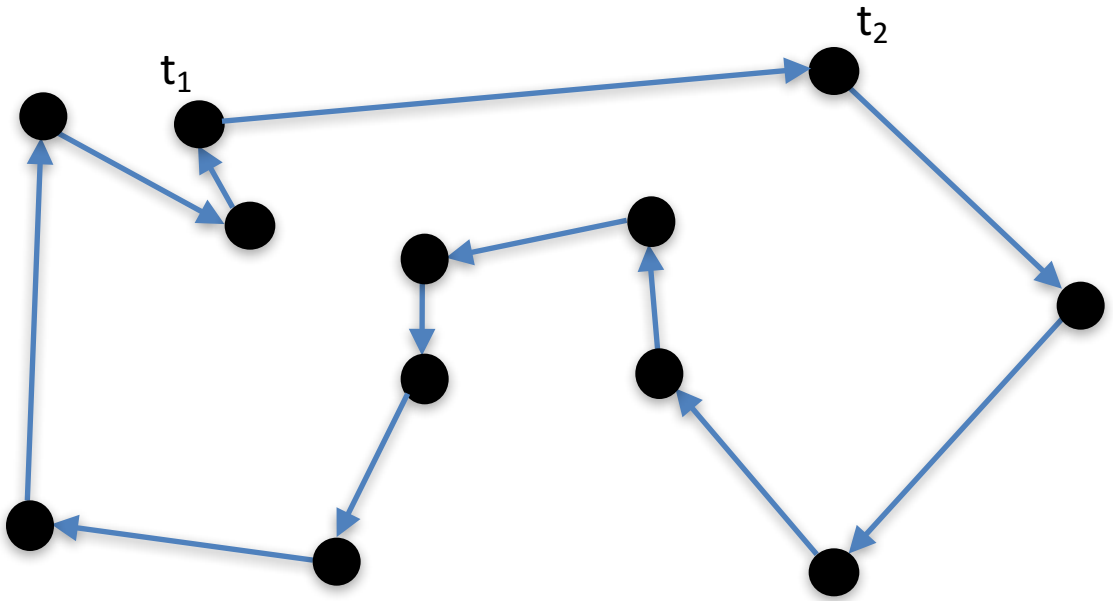
K-OPT:

- Choose a vertex t_1 and an edge (t_1, t_2)
- Choose a vertex t_3 with $d(t_2, t_3) < d(t_1, t_2)$
- If non exist, restart
- Consider solution by removing (t_3, t_4) and adding (t_1, t_4)
- Compute the cost but do not connect

Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

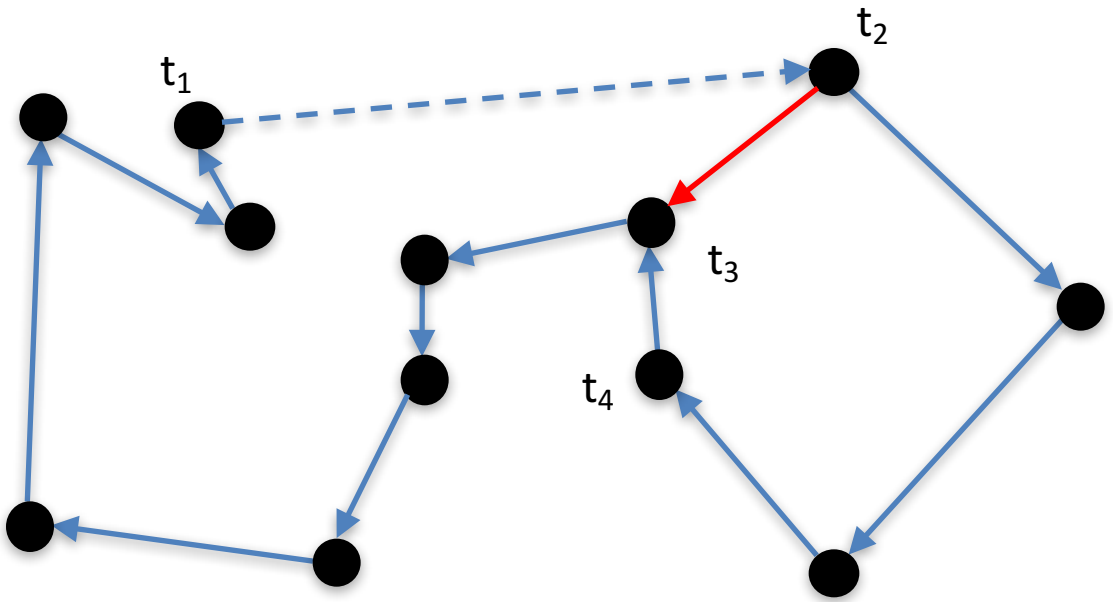
K-OPT:



Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

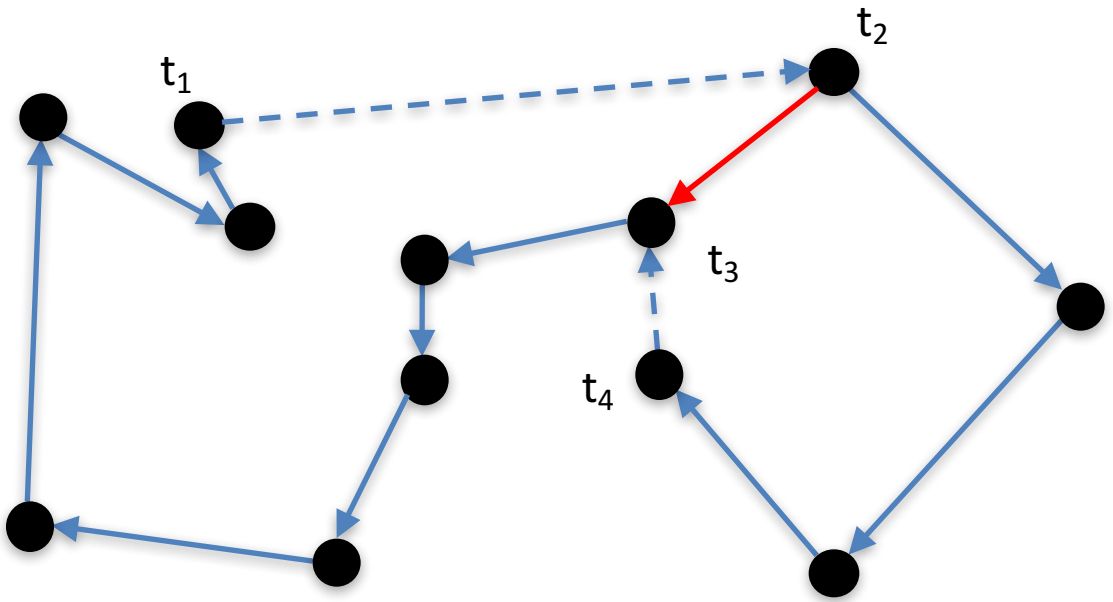
K-OPT:



Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

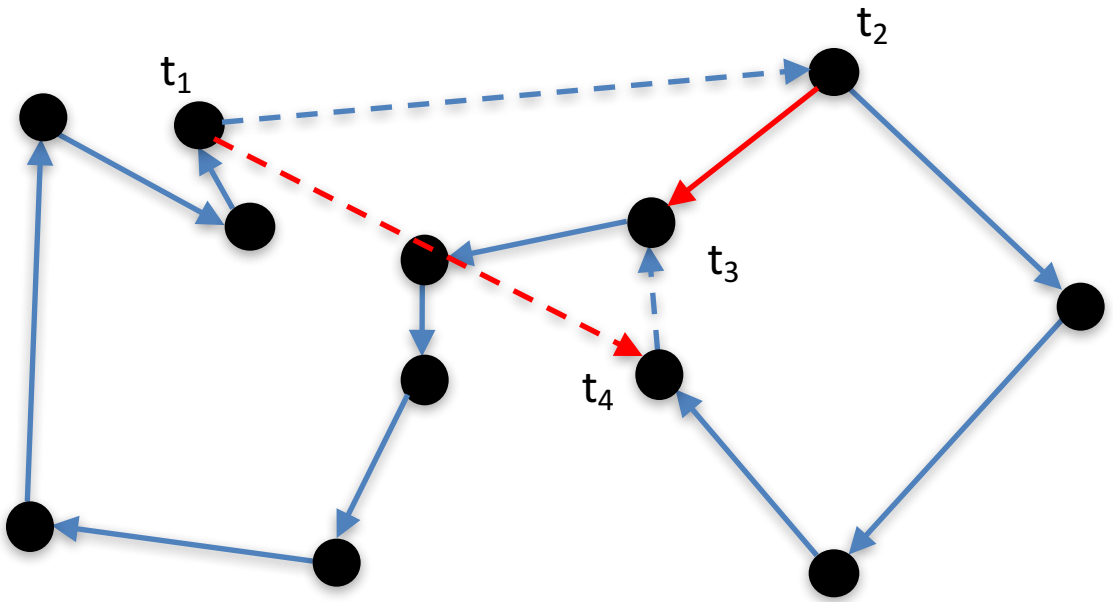
K-OPT:



Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

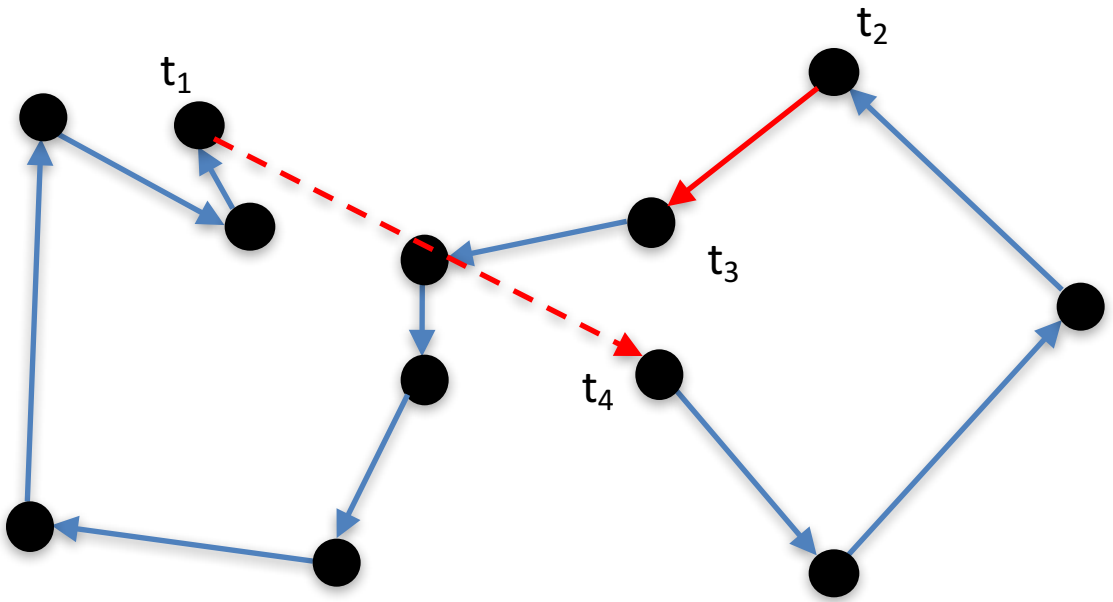
K-OPT:



Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

K-OPT:



Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

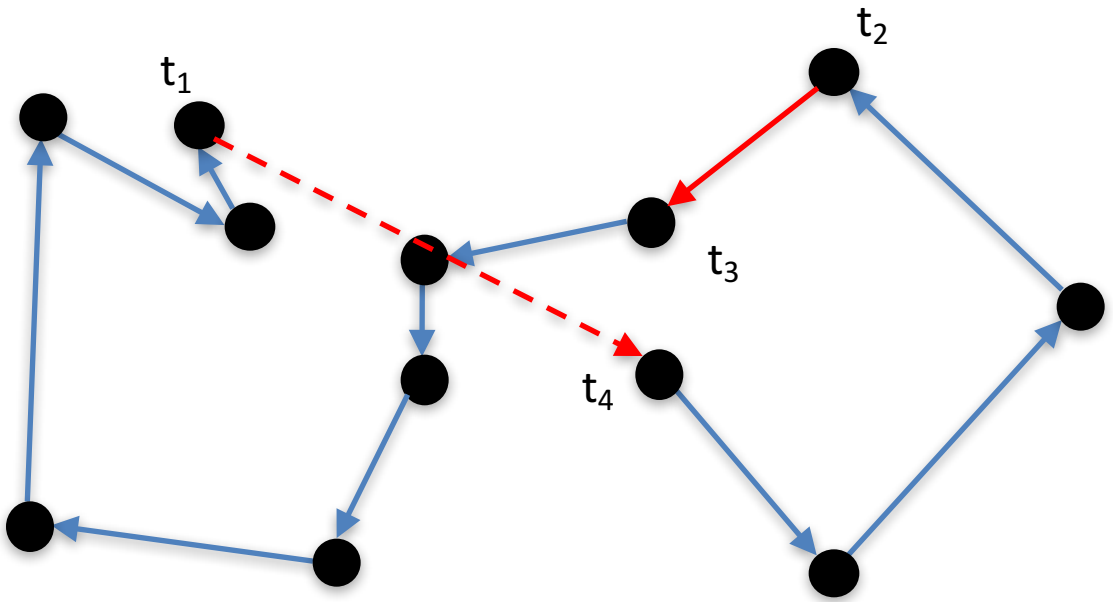
K-OPT:

- Choose a vertex t_1 and an edge (t_1, t_2)
- Choose a vertex t_3 with $d(t_2, t_3) < d(t_1, t_2)$
- If non exist, restart
- Consider solution by removing (t_3, t_4) and adding (t_1, t_4)
- Compute the cost but do not connect
- Restart with t_1 and edge (t_1, t_4)

Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

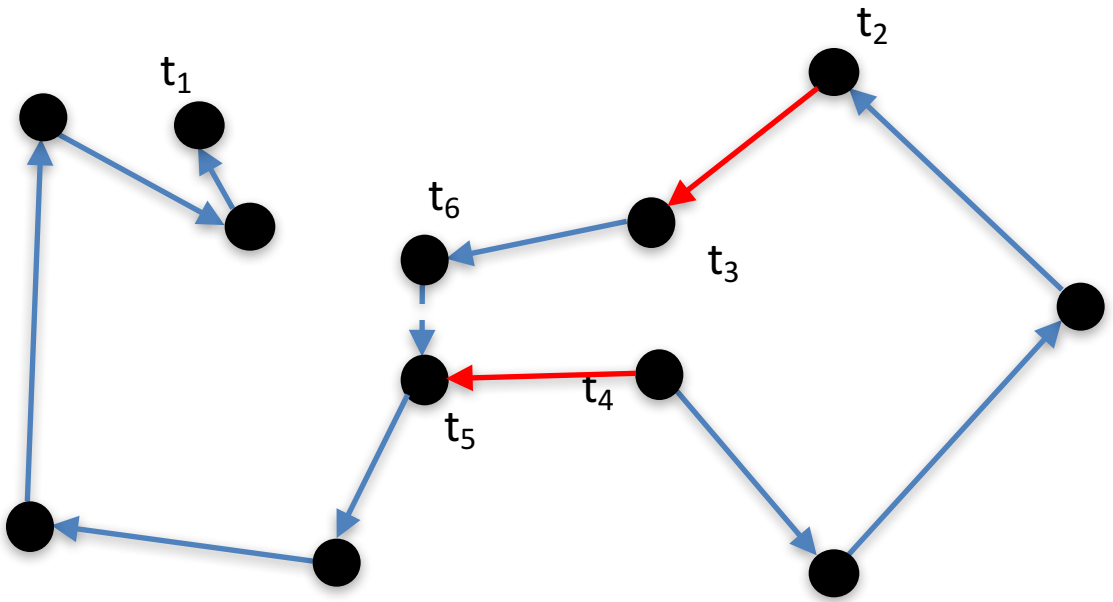
K-OPT:



Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

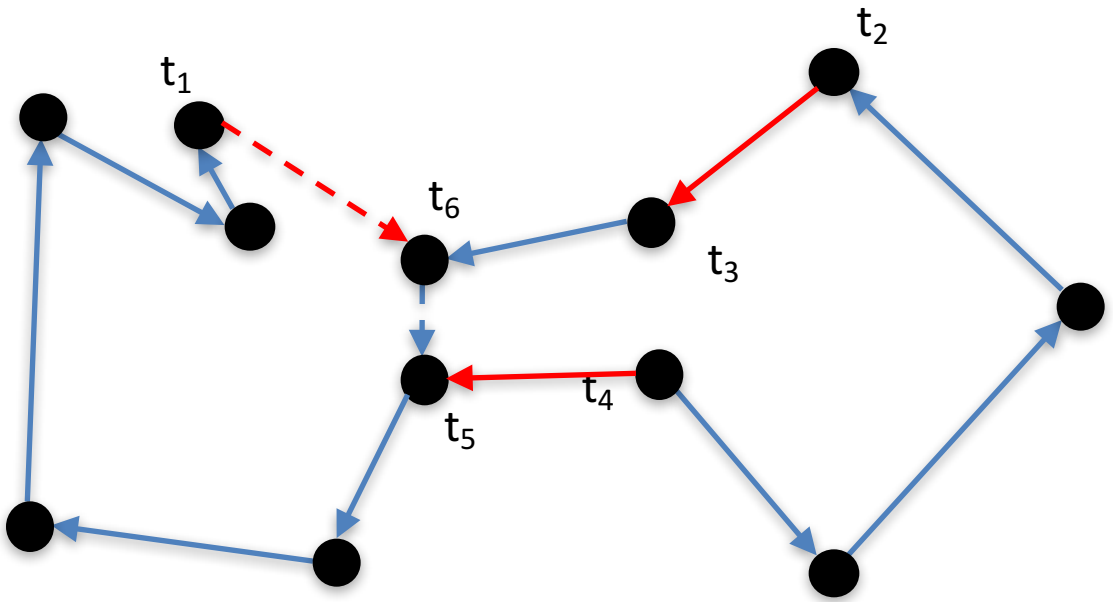
K-OPT:



Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

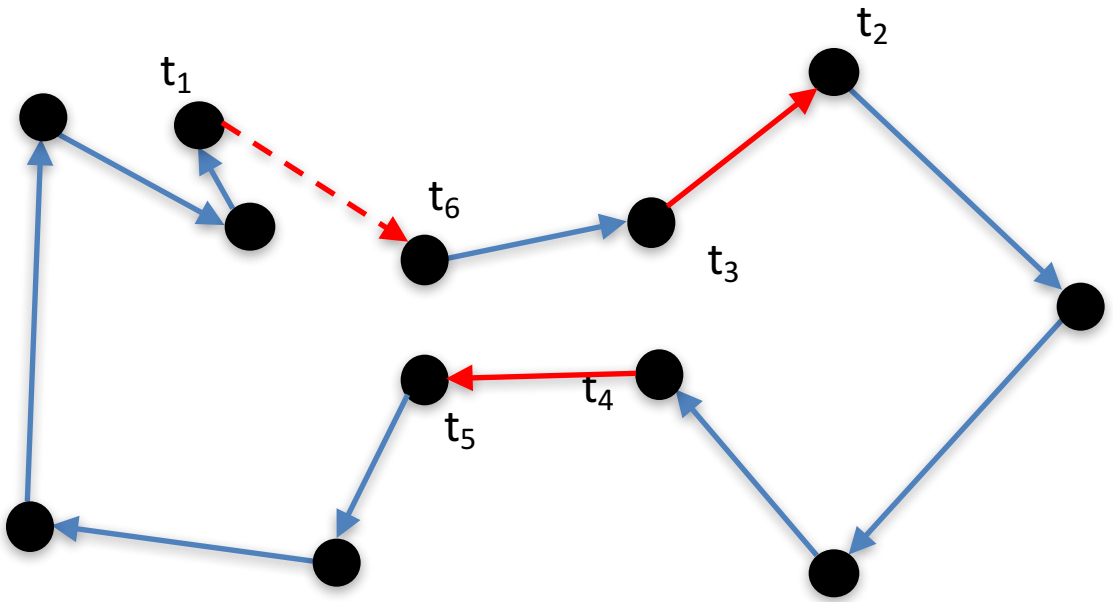
K-OPT:



Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

K-OPT:



Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

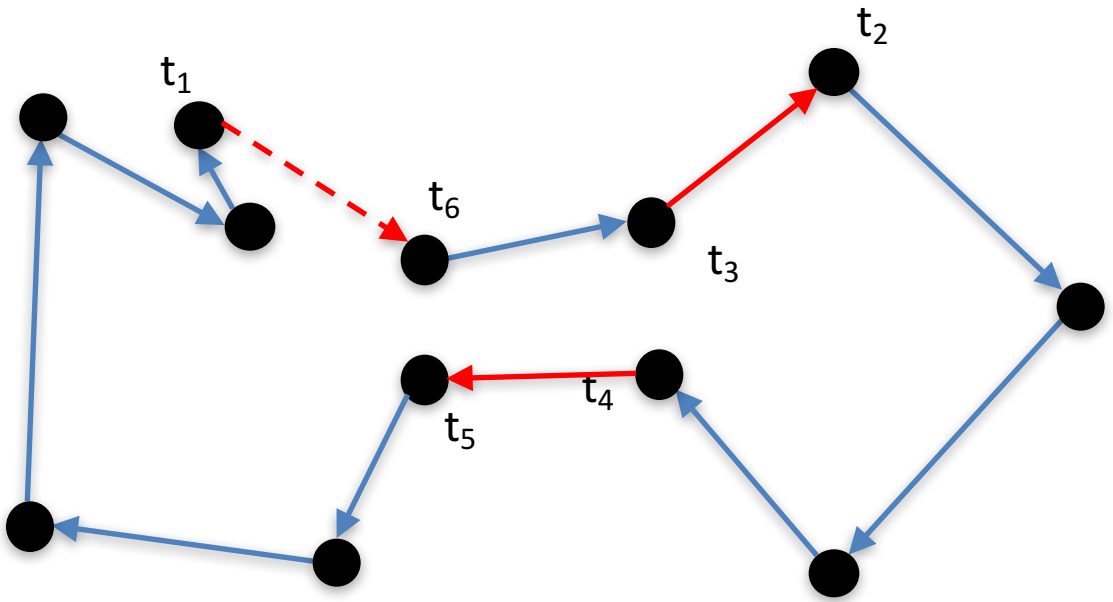
K-OPT:

- Choose a vertex t_1 and an edge (t_1, t_4)
- Choose a vertex t_5 with $d(t_4, t_5) < d(t_1, t_4)$
- If non exist, restart
- Consider solution by removing (t_6, t_5) and adding (t_1, t_6)
- Compute the cost but do not connect
- Restart with t_1 and edge (t_1, t_6)

Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

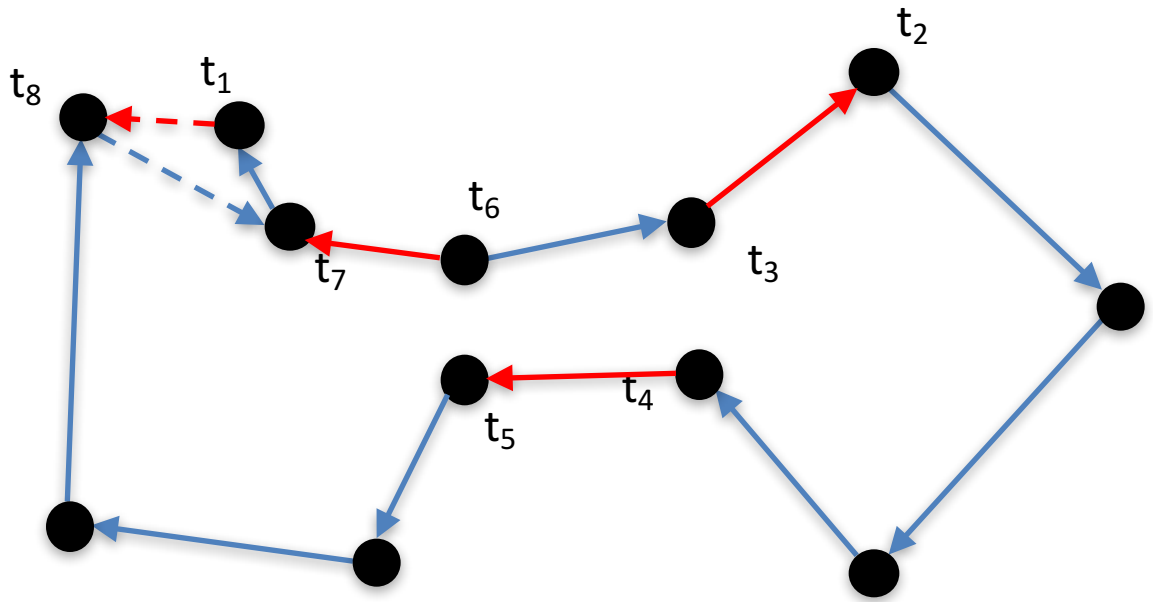
K-OPT:



Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

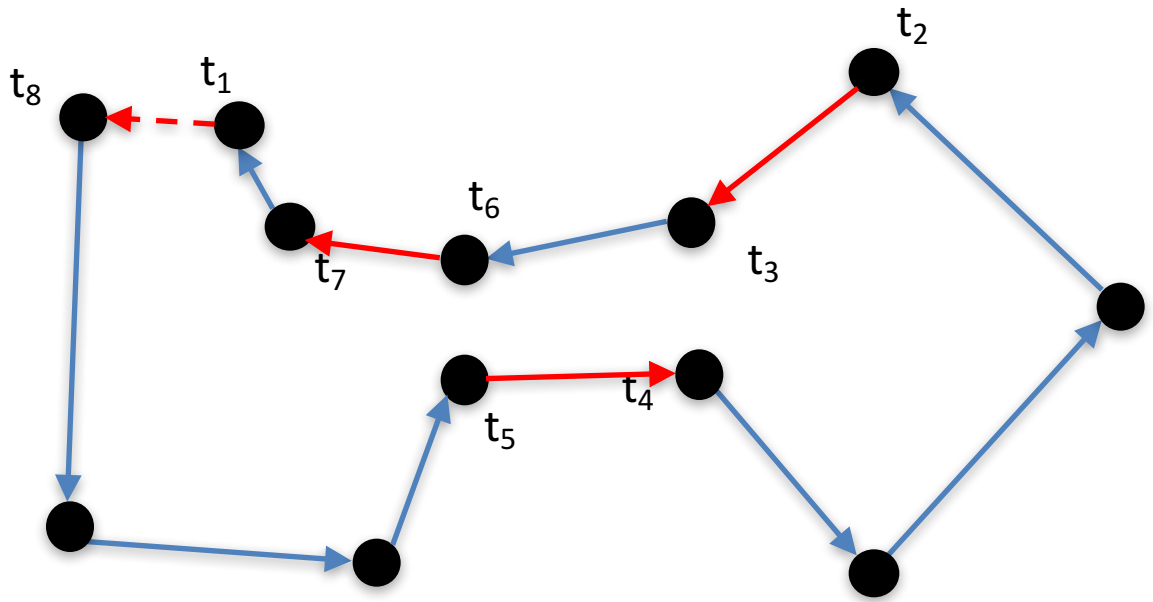
K-OPT:



Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

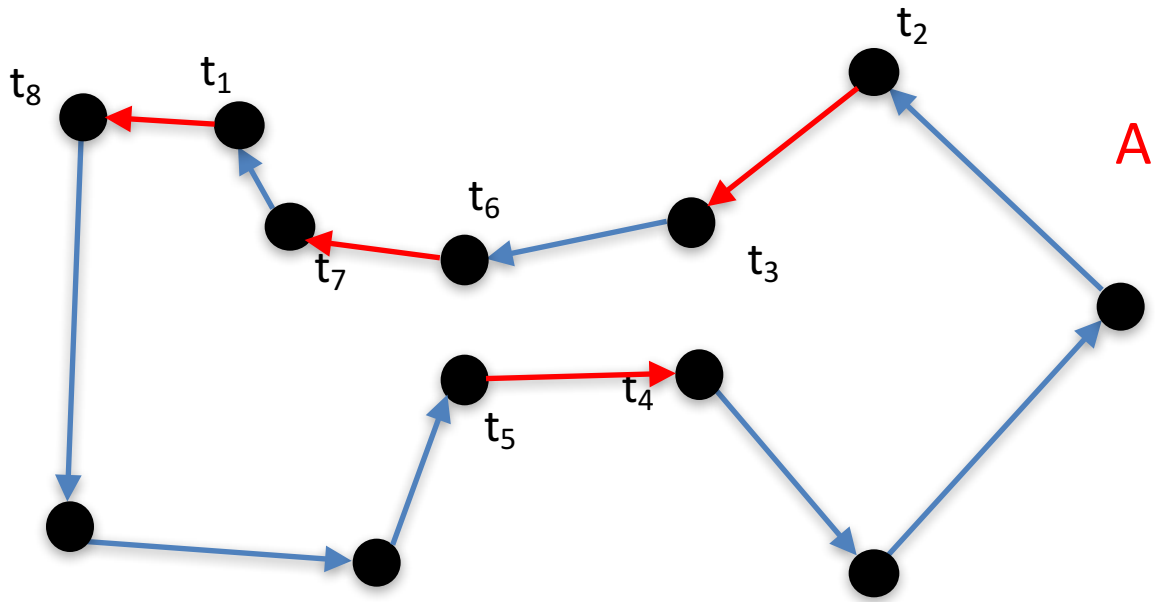
K-OPT:



Local search for TSP: K-OPT

Task: find the shortest path to visit all cities exactly once

K-OPT:



A good move!

Local search for TSP: hints

Task: find the shortest path to visit all cities exactly once

- Start with simple ideas
- Use common sense
  - Which moves are likely to be good?
  - Which moves are likely to be bad?
  - What connections do you need?
- Visualize the path
- Profile code before running for long
- Escape local minima

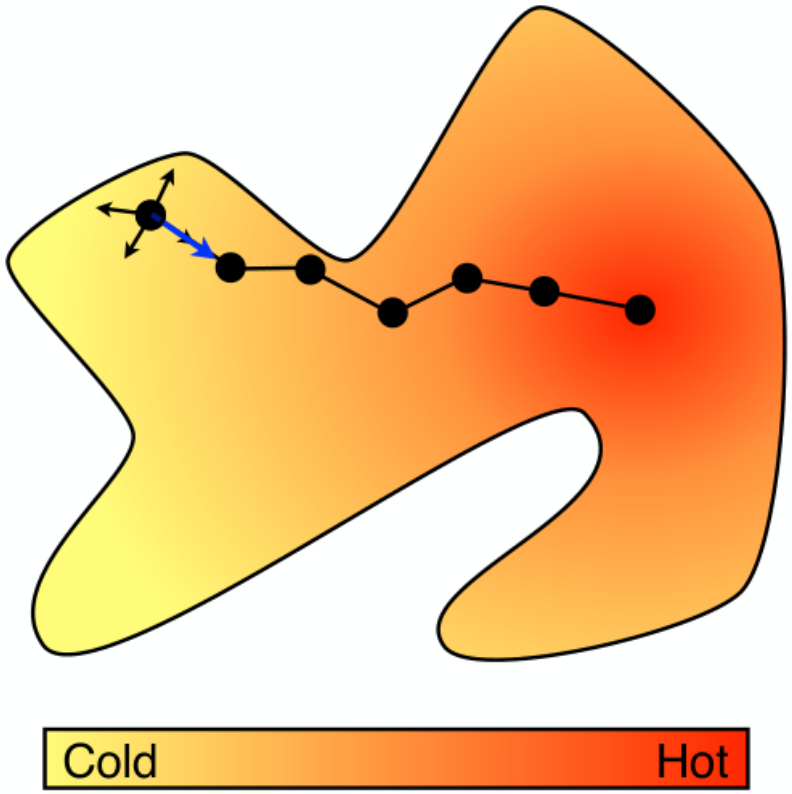
How to make local search work?

Local search has no guarantees.

How to make local search work well?

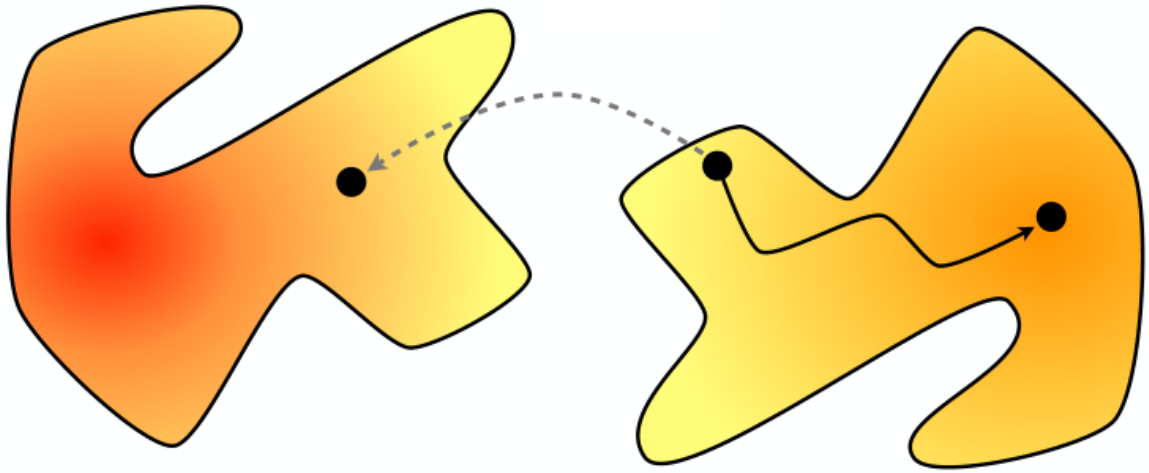
- Connectivity
- Escaping local minima

Local search



Connectivity

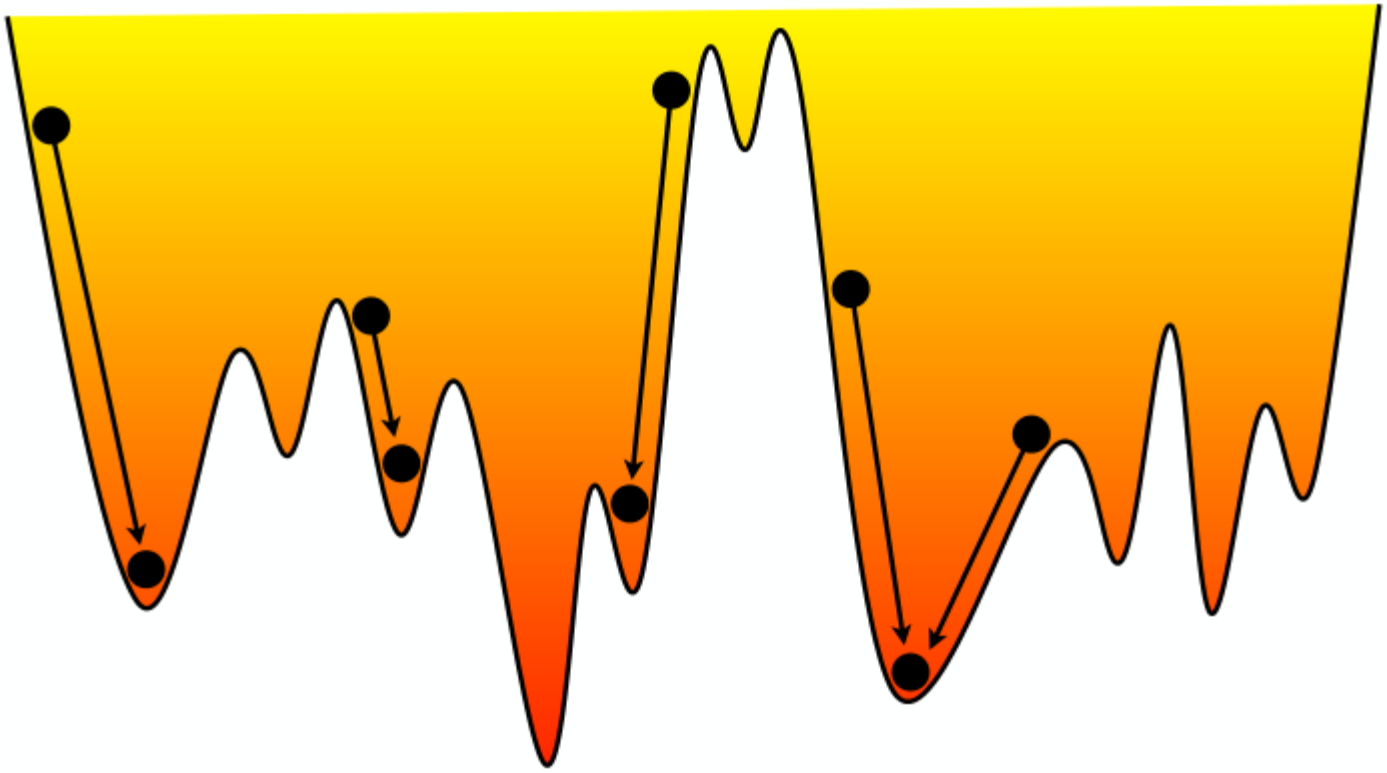
Important property of local neighborhoods



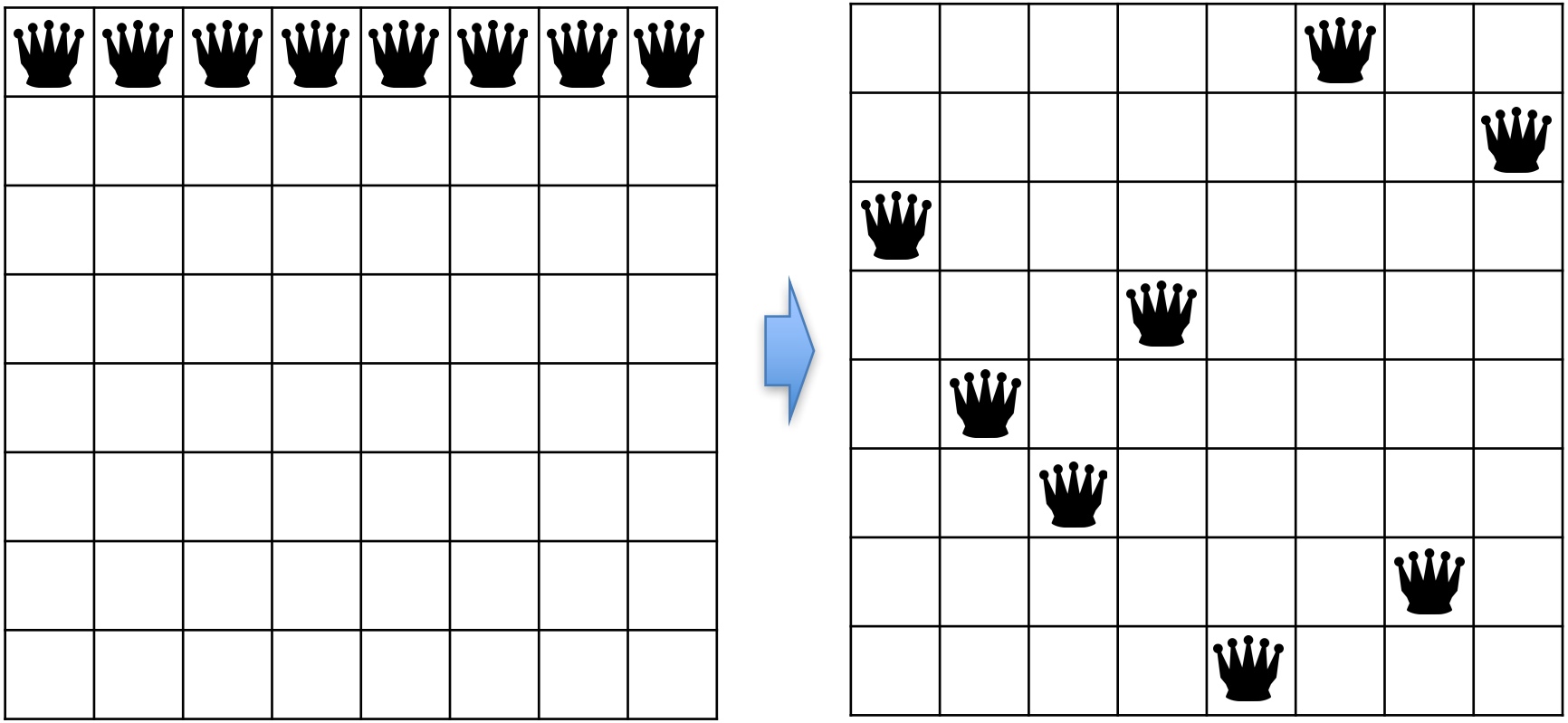
A neighborhood N is connected if, from every configuration S, some optimal solution O can be reached by a sequence of moves

Connectivity

Connectivity does not guarantee optimality



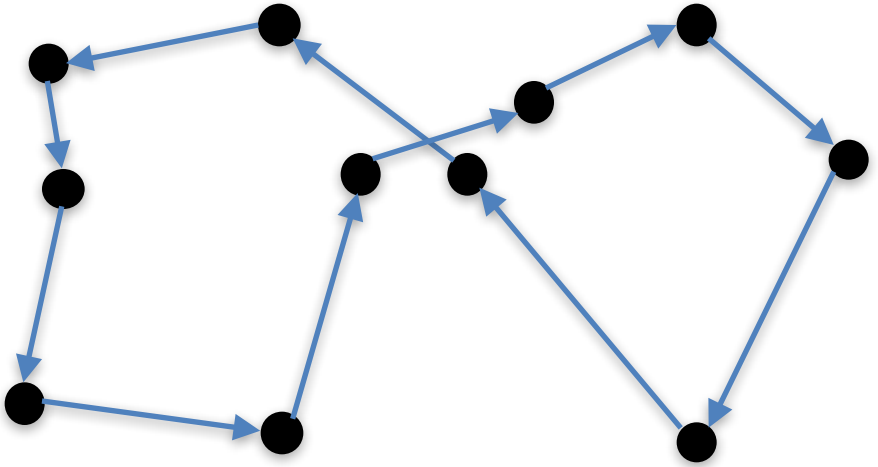
Connectivity of 8-queens



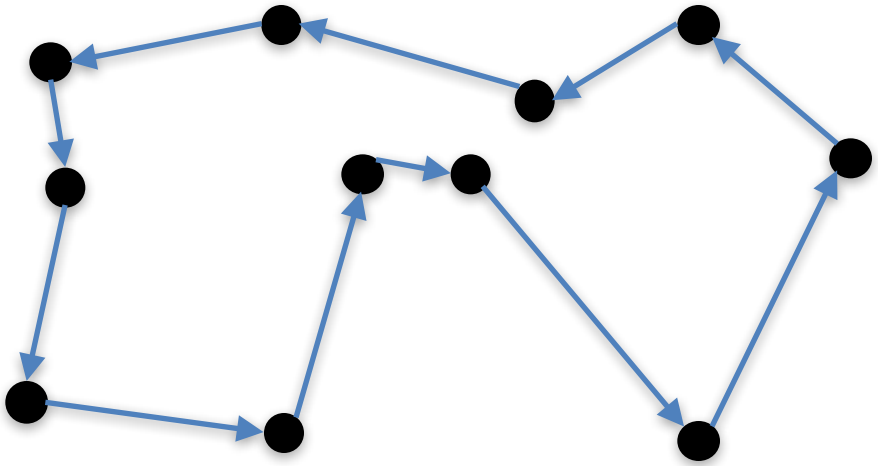
Any two configurations can be linked in at most 8 moves

Connectivity of TSP

Is 2-OPT connected?



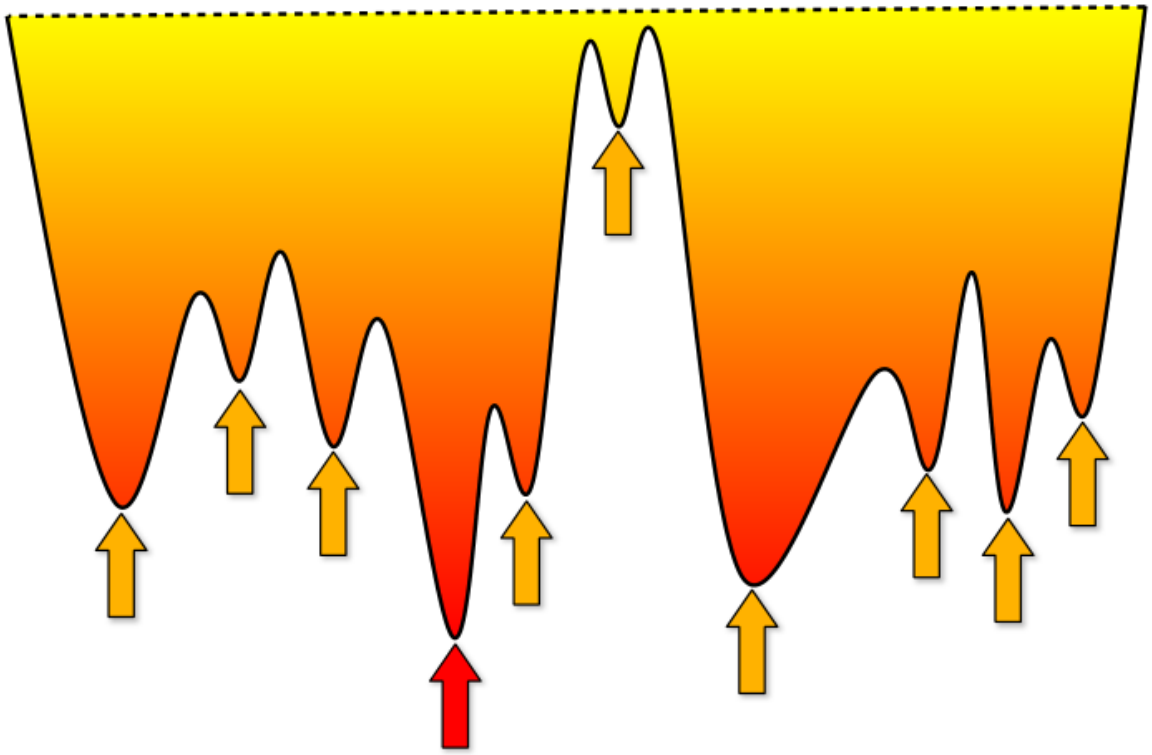
Simple algorithm:
loop other the points along the optimal path and fix edges of the initial path one by one



Local minima

Greedy local search can easily get stuck in local minima.

How to find a good one?



Local search

- States
  - configurations
- Moving from state s to one of its neighbors
  - N(s): neighborhood of s
- Some neighbors are legal; others are not
  - L(N(s), s): set of legal neighbors
- Select one of the legal neighbors
  - S(L(N(s), s), s); selection function
- Objective function
  - minimizing f(s)

Basic local search

```
1.  function LocalSearch(f, N, L, S) {
2.      s := generateInitialSolution();
3.      s* := s;
4.      for k := 1 to MaxTrials do
5.          if satisfiable(s) ∧ f(s) < f(s*) then
6.              s* := s;
7.          s := S(L(N(s), s), s);
8.      return s*;
9.  }
```

Example:

- Legal moves: local improvements
  - L(N, s) = {n in N | f(n) < f(s) }
- Selection function: greedy selection
  - S(L, s) = $\text{argmin}_{n \text{ in } L}$ f(n)

Heuristics and Metaheuristics

Heuristics

- choose the next neighbor
- use local information (state s and its neighborhood)
- drive the search towards a local minimum

Metaheuristics

- aim at escaping local minima
- drive the search towards a global minimum
- typically include some memory of learning

Properties of the neighbors

- Legal neighbors

  conditions on the value of the objective function

- Local improvement

  $L(N, s) = \{ n \text{ in } N \mid f(n) < f(s) \}$

- No degradation

  $L(N, s) = \{ n \text{ in } N \mid f(n) <= f(s) \}$

- Potential degradation

  $L(N, s) = N$

Selecting a neighbor

- How to select the neighbor?

  exploring the whole or part of the neighborhood

- Best neighbor

  select “the” best neighbor in the neighborhood

- First neighbor

  select the first “legal” neighbor

  avoid scanning the entire neighborhood

- Multi-stage selection

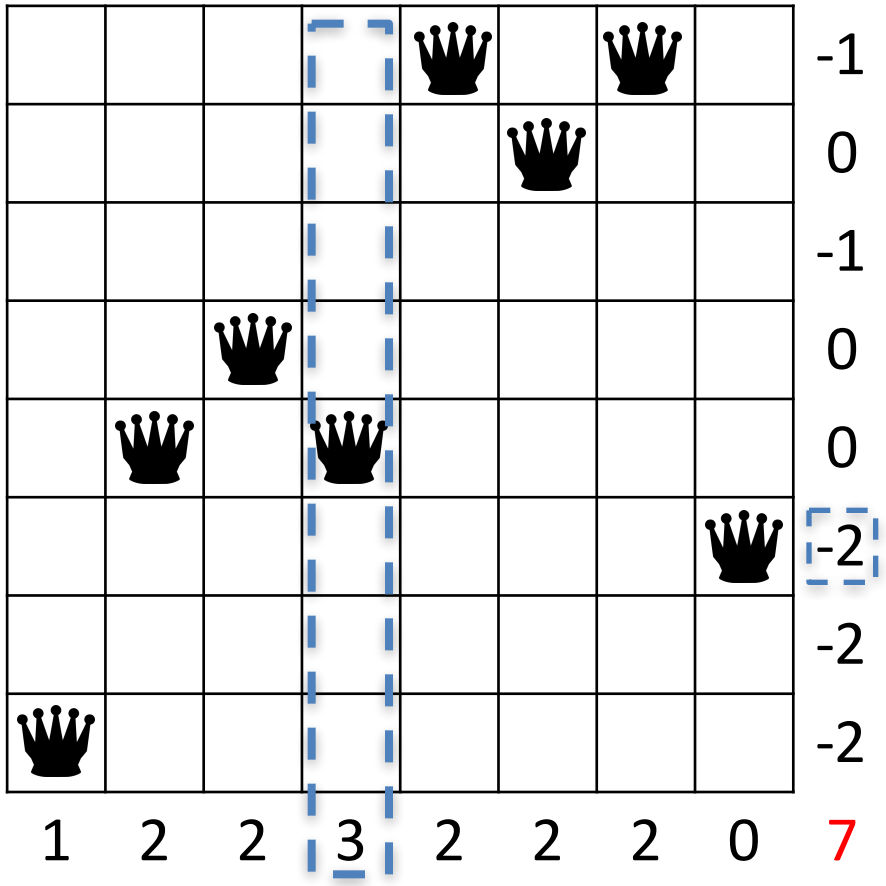
  first, select one “part” of neighborhood

  second, select from the remaining “part” of neighborhood

Multi-stage selection

1. Select the variable with the most violations

2. Select the position with the fewest resulting violations



O(n) runtime against O(n^2) of the full search

Escaping local minima

- Randomization
  - Random restarts

    generic, can be always tried
  - Metropolis scheme
  - Simulated annealing

- Tabu search

Metropolis heuristics

- Basic idea
  - accept a move if it improves the objective value
  - accept “bad moves” as well with some probability
  - the probability depends on how “bad” the move is
  - inspired by statistical physics

- How to choose the probability?
  - t is a scaling parameter (called temperature)
  - Δ is the difference f(n) – f(s)
  - a degrading move is accepted with probability

$$\exp\left(\frac{-\Delta}{t}\right)$$

Metropolis heuristics

- What happens for a large t ?
  - probability of accepting a degrading move is large
- What happens for a small t ?
  - probability of accepting a degrading move is small
- Finding the correct temperature is hard
- Let us gradually change the temperature

  simulated annealing

Simulated annealing

- Based on statistical physics
  - Heating and cooling schedules of crystals
- Key idea
  - Use Metropolis algorithm but adjust the temperature dynamically
  - Start with a high temperature (random moves)
  - Decrease the temperature
  - When the temperature is low becomes a local search

Simulated annealing

- Guaranteed to converge to a global optimum
  - connected neighborhood
  - slow cooling schedule

    slower than the exhaustive search

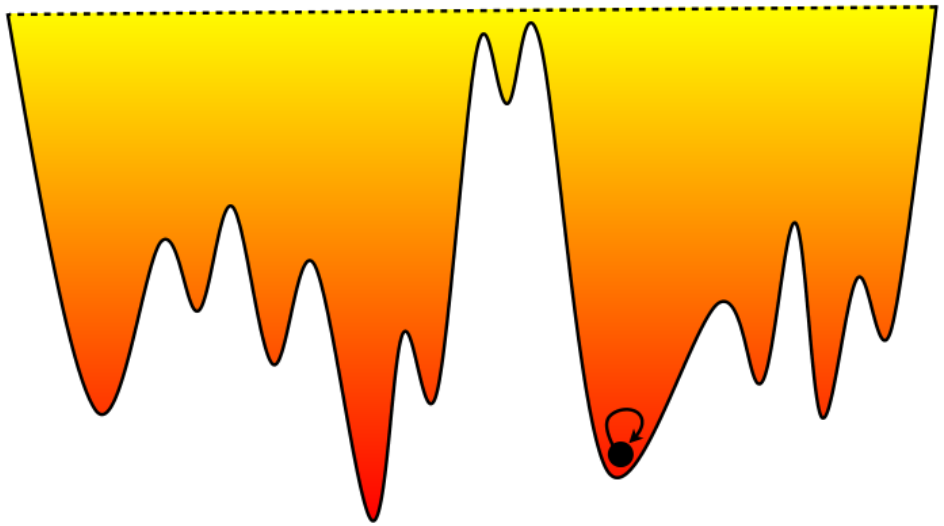
- In practice
  - can give excellent results
  - need to tune a temperature schedule
  - default choice: $t_{k+1} = \alpha\, t_k$

- Additional tools
  - restarts
  - reheats

Tabu search

- Key idea
  - maintain the sequence of nodes already visited

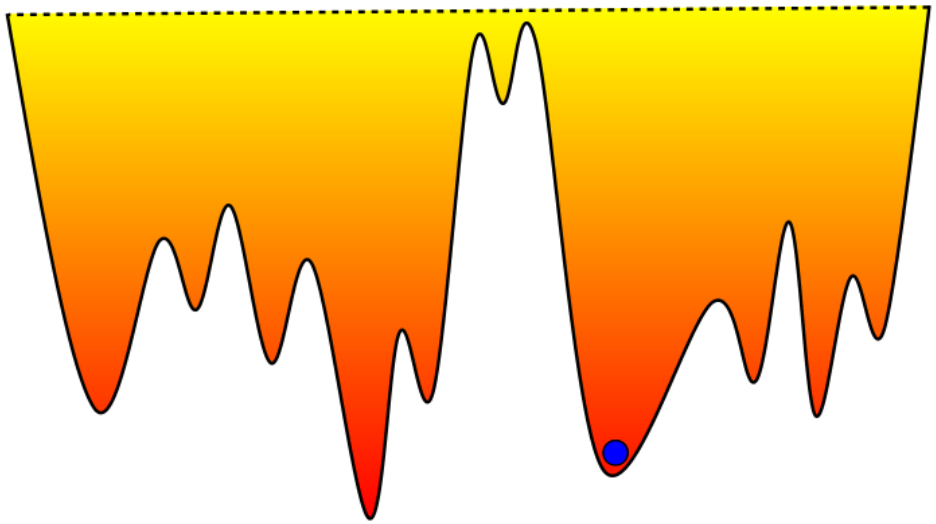
    tabu lists and tabu nodes



Tabu search

- Key idea
  - maintain the sequence of nodes already visited

    tabu lists and tabu nodes



Tabu nodes ●: nodes already visited

Tabu search

- Key idea
  - maintain the sequence of nodes already visited

    tabu lists and tabu nodes

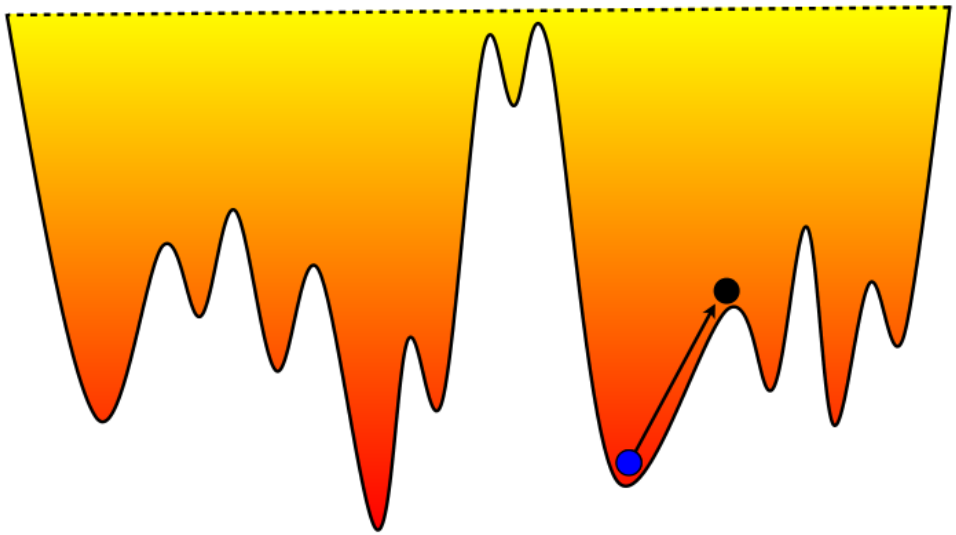
- Select the best configurations that is not tabu,

  i.e., has not been visited before

Tabu search

- Key idea
  - maintain the sequence of nodes already visited

    tabu lists and tabu nodes

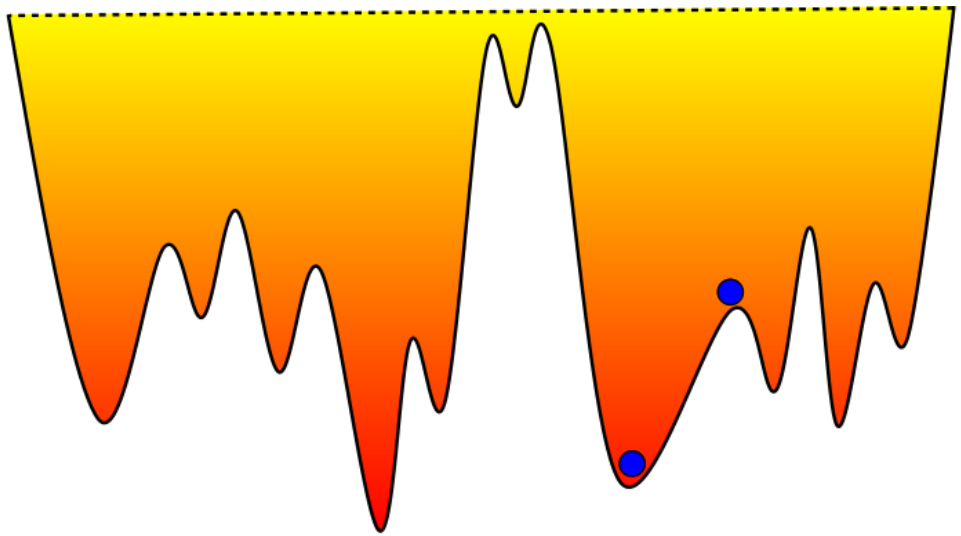


Tabu nodes ●: nodes already visited

Tabu search

- Key idea
  - maintain the sequence of nodes already visited

    tabu lists and tabu nodes

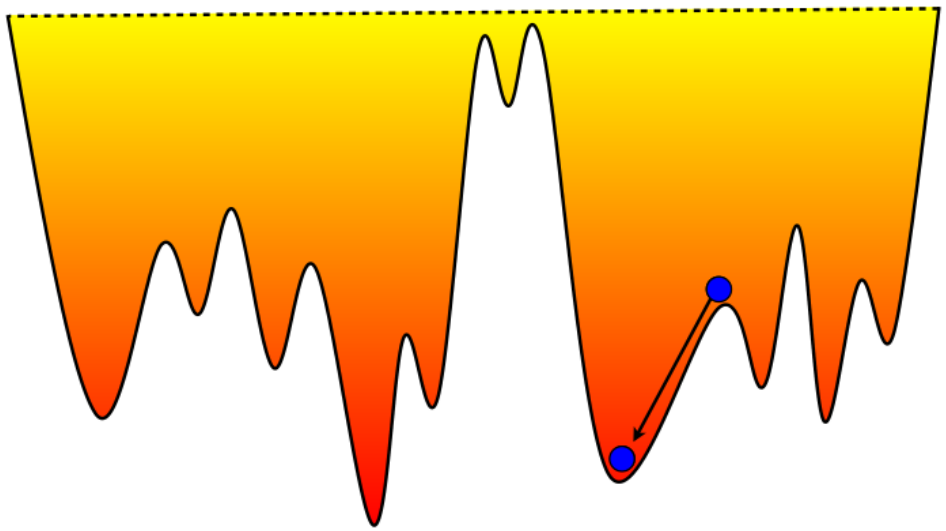


Tabu nodes ●: nodes already visited

Tabu search

- Key idea
  - maintain the sequence of nodes already visited

    tabu lists and tabu nodes

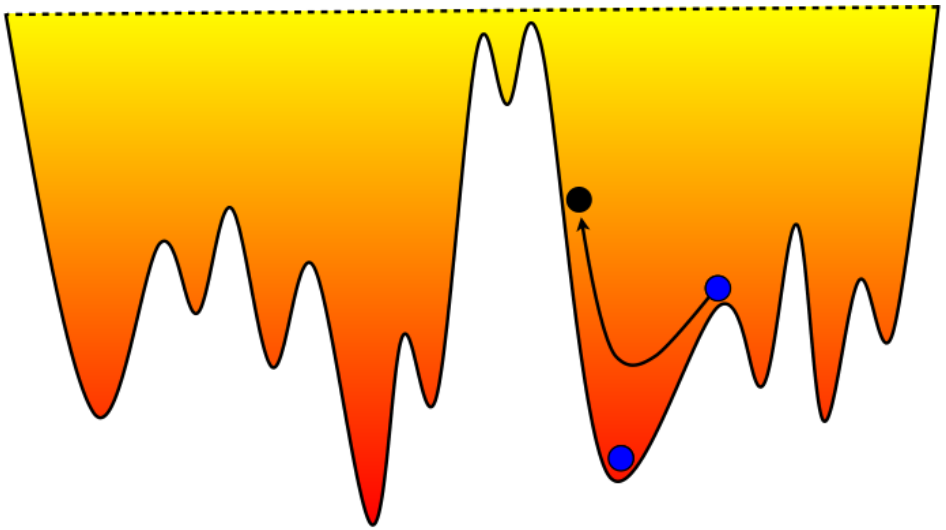


Tabu nodes ●: nodes already visited

Tabu search

- Key idea
  - maintain the sequence of nodes already visited

    tabu lists and tabu nodes

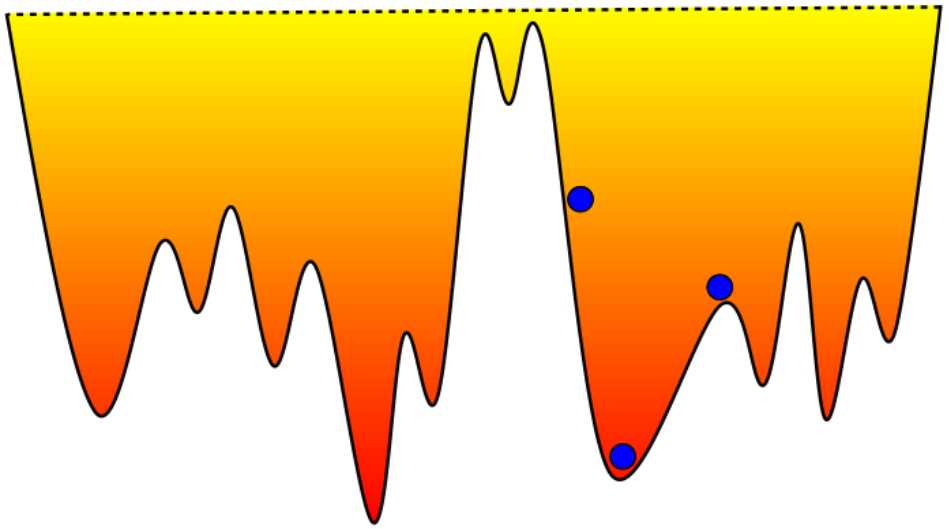


Tabu nodes ●: nodes already visited

Tabu search

- Key idea
  - maintain the sequence of nodes already visited

    tabu lists and tabu nodes

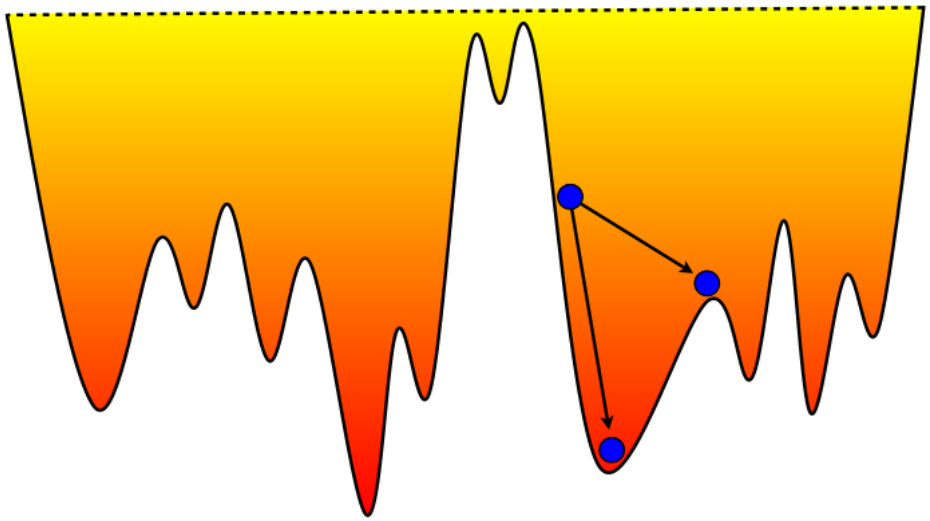


Tabu nodes ●: nodes already visited

Tabu search

- Key idea
  - maintain the sequence of nodes already visited

    tabu lists and tabu nodes

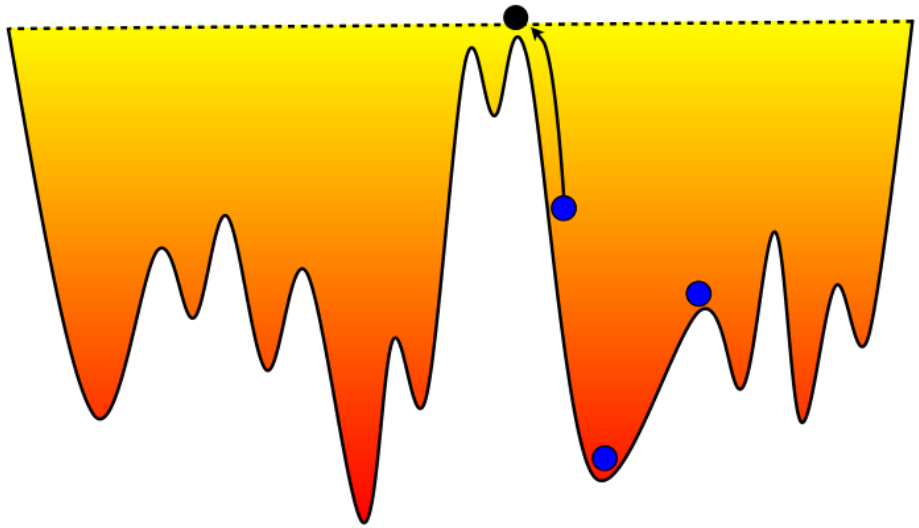


Tabu nodes ●: nodes already visited

Tabu search

- Key idea
  - maintain the sequence of nodes already visited

    tabu lists and tabu nodes



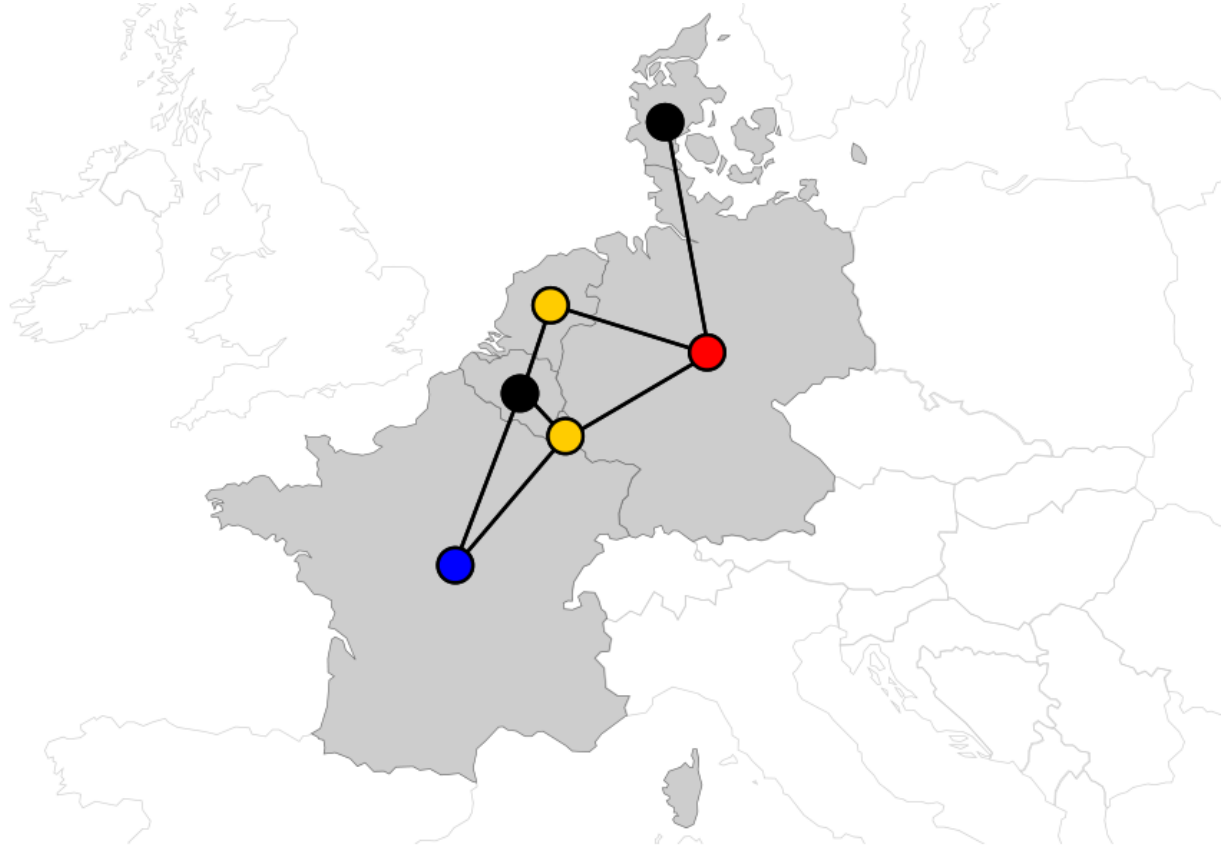
Tabu nodes ●: nodes already visited

Tabu search

- Key issue with tabu search
  - expensive to maintain all the visited nodes
- Short-term memory
  - only keep a small set of recently visited nodes (tabu list)
- Keep an abstraction of the nodes
  - many possibilities
  - store the transitions instead of the states
  - store the transitions and the objective values

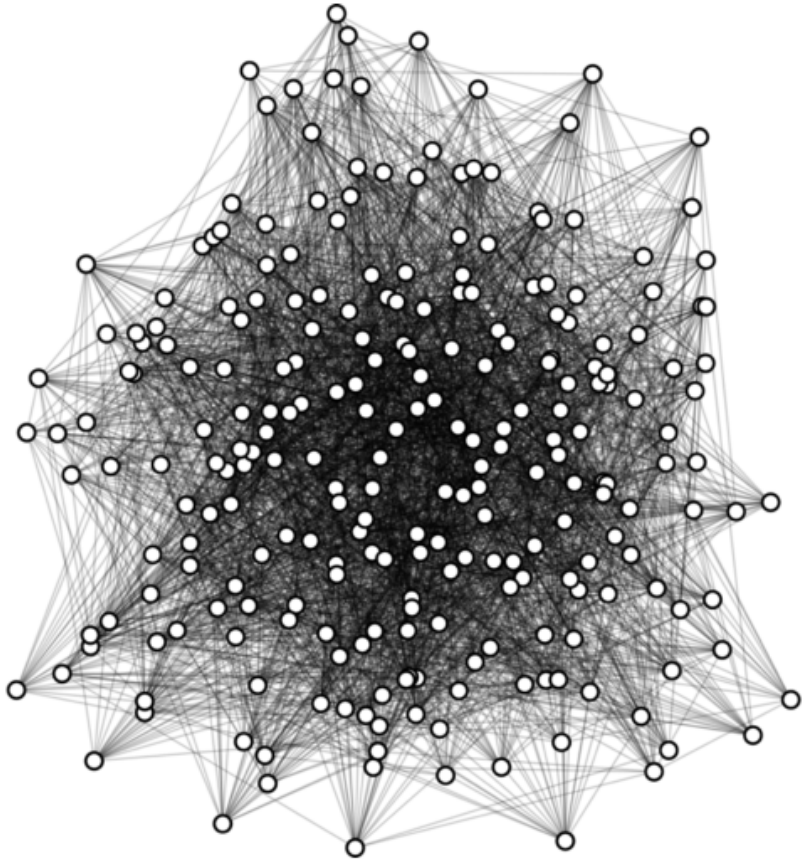
Optimization under constrains

Task: color the nodes of the graph such that the neighboring nodes are colored differently



Graph coloring

Task: color the nodes of the graph such that the neighboring nodes are colored differently



Graph coloring

Task: color the nodes of the graph such that the neighboring nodes are colored differently

Two aspects:

- optimization: reducing the number of colors
- feasibility: two adjacent vertices must be colored differently

How to combine them in local search?

- sequence of feasibility problems
- staying in the space of solutions
- considering feasible and infeasible configurations

Graph coloring

Task: color the nodes of the graph such that the neighboring nodes are colored differently

Sequence of feasibility problems:

- find an initial solution with k colors (e.g. greedy method)
- remove one color (randomly reassign the nodes)
- find a feasible solution with k-1 colors
- repeat

How to find a solution with k-1 colors?

- minimize the violations

Graph coloring

Task: color the nodes of the graph such that the neighboring nodes are colored differently

Staying in the feasible Space:

- simple neighborhood

  change the color of a vertex

- objective function

  minimizing the number of colors

- how to guide the search?

Staying in the feasible space

Task: color the nodes of the graph such that the neighboring nodes are colored differently

How to drive the search?

use a proxy as objective function

favor large classes

Objective: $\max \sum_{i=1}^{n} |C_i|^2$

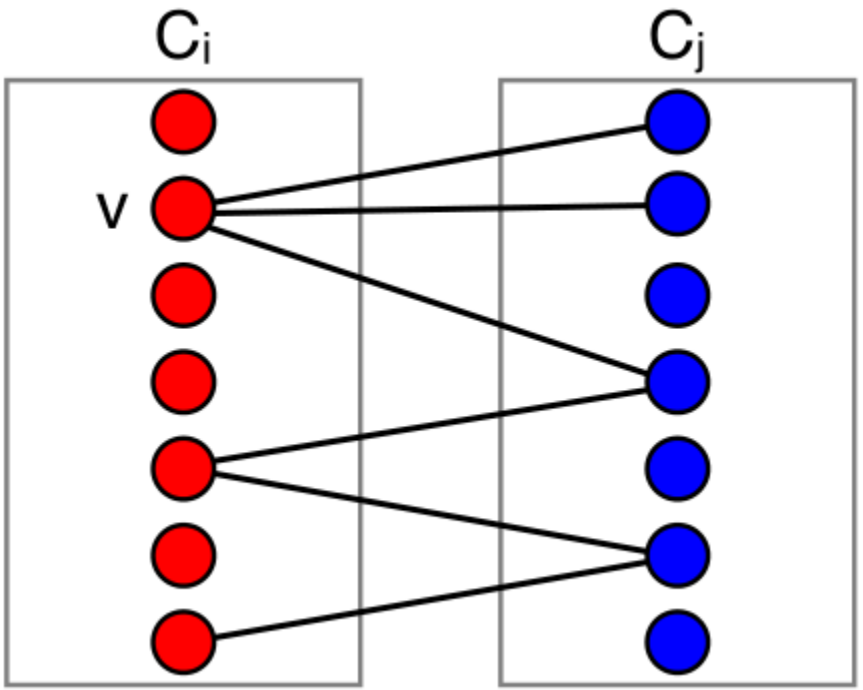
C_i is the set of vertices colored with i

Staying in the feasible space

Task: color the nodes of the graph such that the neighboring nodes are colored differently

Richer neighborhoods:

chains of variables

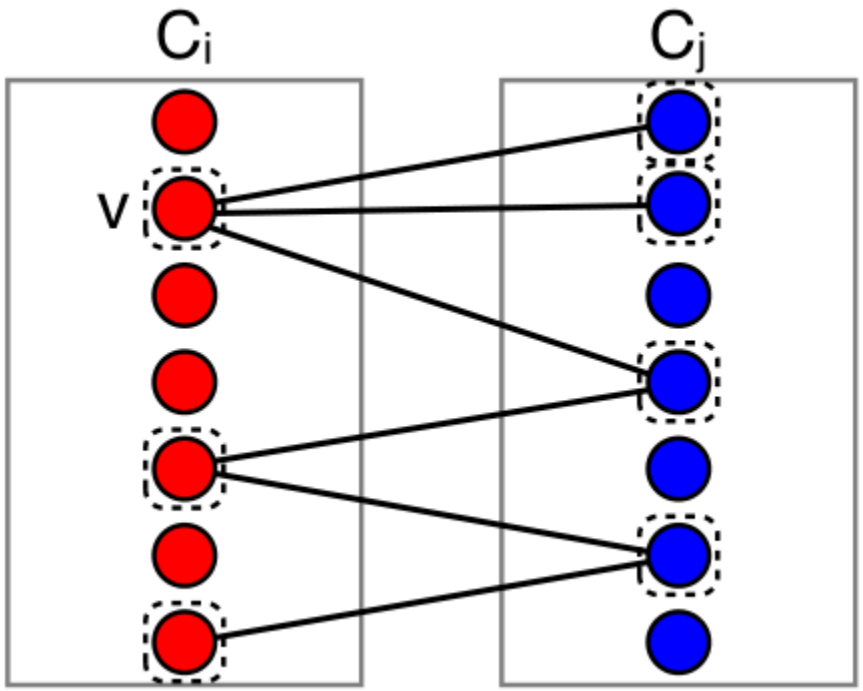


Staying in the feasible space

Task: color the nodes of the graph such that the neighboring nodes are colored differently

Richer neighborhoods:

chains of variables

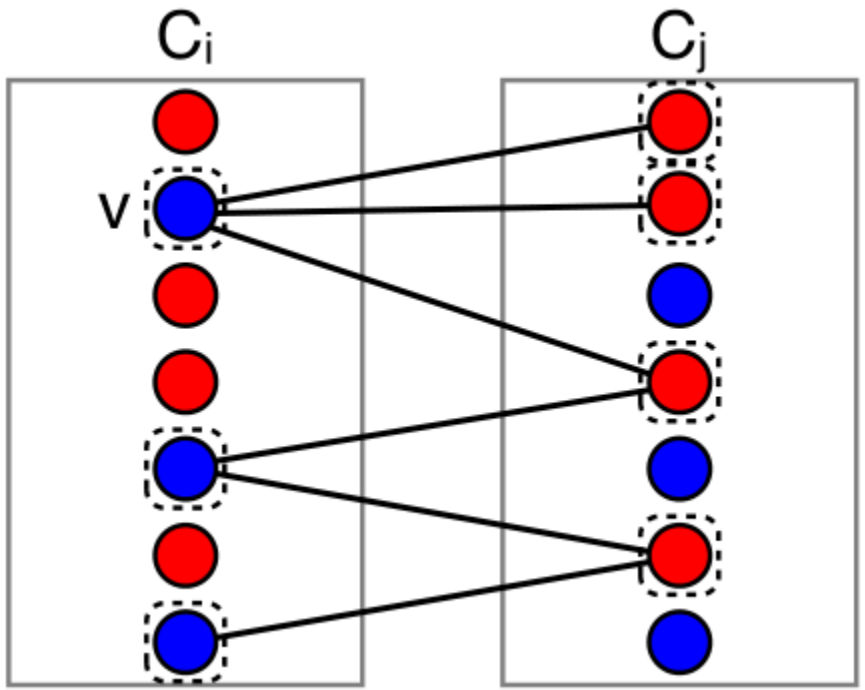


Staying in the feasible space

Task: color the nodes of the graph such that the neighboring nodes are colored differently

Richer neighborhoods:

chains of variables

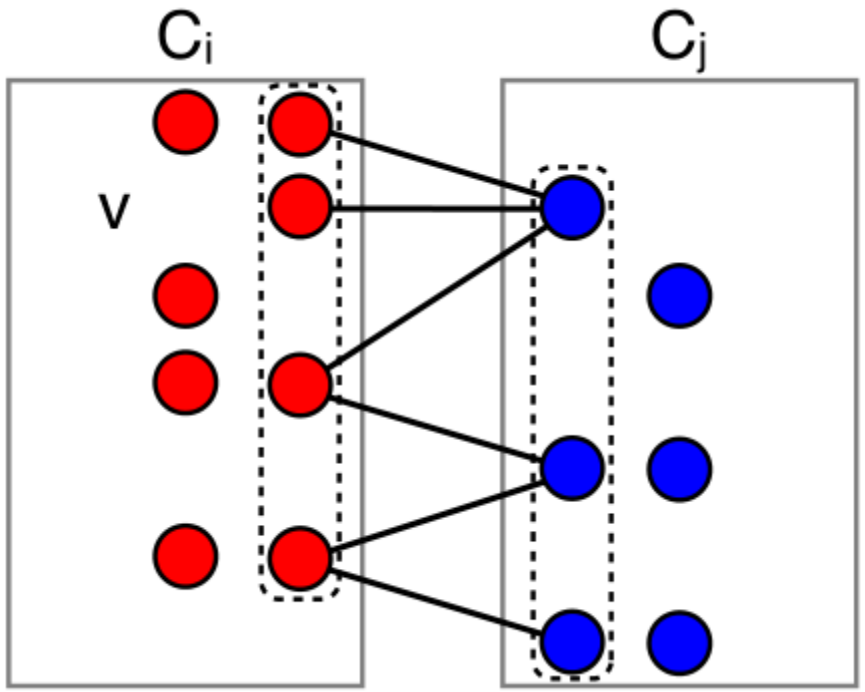


Staying in the feasible space

Task: color the nodes of the graph such that the neighboring nodes are colored differently

Richer neighborhoods:

chains of variables



Feasible and infeasible colorings

Task: color the nodes of the graph such that the neighboring nodes are colored differently

Explore both feasible and infeasible colorings

Search must focus on reducing the number of colors and on ensuring feasibility

Use an objective function than balances feasibility and optimality

$$\min w_f f + w_o O$$

Feasible and infeasible colorings

Task: color the nodes of the graph such that the neighboring nodes are colored differently

Use an objective function than balances feasibility and optimality

Decreasing the number of colors $f = \sum_{i=1}^{n} |C_i|^2$

Removing violations $O = \sum_{i=1}^{n} |B_i|$

B_i is a set of bad edges between vertices colored with *i*

Feasible and infeasible colorings

Task: color the nodes of the graph such that the neighboring nodes are colored differently

Use an objective function than balances feasibility and optimality

$$\min \sum_{i=1}^{n} 2|B_i||C_i| - \sum_{i=1}^{n} |C_i|^2$$

Why?

All local minima of this objective are legal colorings

Neighborhood: changing the color of one node

If there is a bad edge with color *i*

The left term: $2|B_i||C_i - 2(|B_i - 1)(C_i| - 1) = 2|B_i| + 2C_i| - 2 \geq 2|C_i|$

The right term: $|C_i|^2 ((|C_i| - 1)^2 + 1) = 2C_i - 2$

Examples of complex neighborhoods

Sports scheduling

- Practical applications
  - football, hockey, basketball, baseball
  - radio and TV: €€€
- The travelling tournament problem (TTP)
  - abstraction of major league tournament
  - proposed by Easton, Nemhauser, and Trick

Travelling Tournament Problem (TTP)

- Input
  - n teams
  - a matrix d of distances between teams
- Output: a double all-play-all schedule
  - each team plays each home and away
  - minimize travel distance

Travelling Tournament Problem (TTP)

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	4	3	6	@4	@1	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	@2	1	5	2	@6	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

$$d_{12} + d_{21} + d_{15} + d_{54} + d_{43} + d_{31} + d_{16} + d_{61}$$

$$d_{12} + d_{21} + d_{15} + d_{54} + d_{43} + d_{31} + d_{16} + d_{61}$$

$$+ \ldots +$$

$$d_{61} + d_{14} + d_{45} + d_{56} + d_{63} + d_{36} + d_{62} + d_{26}$$

Travelling Tournament Problem (TTP)

Moves:

- swap homes
- swap rounds
- swap teams
- partial swap rounds
- partial swap teams

TTP: swap homes

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	4	3	6	@4	@1	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	@2	1	5	2	@6	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	@4	3	6	4	@1	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	2	1	5	@2	@6	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

TTP: swap rounds

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	4	3	6	@4	@1	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	@2	1	5	2	@6	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	@5	3	4	@4	@3	5	2	@6
2	5	1	4	@6	@3	3	6	@4	@1	@5
3	@4	5	6	@1	2	@2	1	@6	@5	4
4	3	6	@2	@5	@1	1	5	2	@6	@3
5	@2	@3	1	4	6	@6	@4	@1	3	2
6	@1	@4	@3	2	@5	5	@2	3	4	1

TTP: swap teams

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	4	3	6	@4	@1	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	@2	1	5	2	@6	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	@3	6	4	1	@6	@4	@1	3	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	2	1	5	@2	@6	@3
5	@2	1	@3	@6	4	3	6	@4	@1	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

TTP: swap teams

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	4	3	6	@4	@1	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	@2	1	5	2	@6	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	@3	6	4	1	@6	@4	@1	3	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	2	1	5	@2	@6	@3
5	@2	1	@3	@6	4	3	6	@4	@1	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

TTP: swap teams

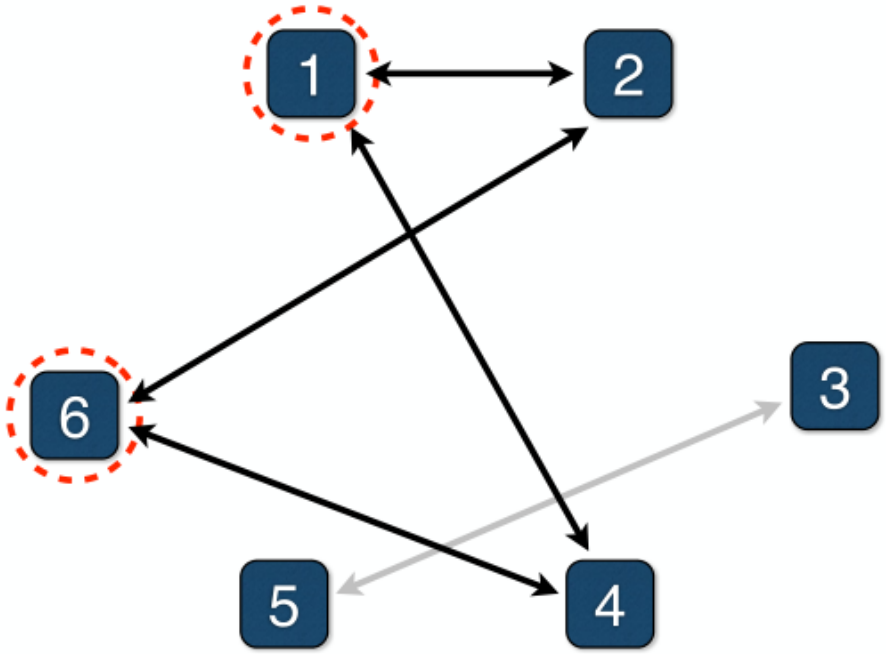
T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	@3	6	4	1	@6	@4	@1	3	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	2	1	5	@2	@6	@3
5	@2	1	@3	@6	4	3	6	@4	@1	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@5	4	3	@2	@4	@3	2	5	@6
2	5	@3	6	4	1	@6	@4	@1	3	@5
3	@4	2	5	@1	6	@5	1	@6	@2	4
4	3	6	@1	@2	@5	1	2	5	@6	@3
5	@2	1	@3	@6	4	3	6	@4	@1	2
6	@1	@4	@2	5	@3	2	@5	3	4	1

TTP: swap partial rounds

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	2	3	@5	@4	@3	5	4	@6
2	5	1	@1	@5	4	3	6	@4	@6	@3
3	@4	5	4	@1	6	@2	1	@6	@5	2
4	3	6	@3	@6	@2	1	5	2	@1	@5
5	@2	@3	6	2	1	@6	@4	@1	3	4
6	@1	@4	@5	4	@3	5	@2	3	2	1

TTP: swap partial rounds



TTP: swap partial rounds

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	2	3	@5	@4	@3	5	4	@6
2	5	1	@1	@5	4	3	6	@4	@6	@3
3	@4	5	4	@1	6	@2	1	@6	@5	2
4	3	6	@3	@6	@2	1	5	2	@1	@5
5	@2	@3	6	2	1	@6	@4	@1	3	4
6	@1	@4	@5	4	@3	5	@2	3	2	1

T-R	1	2	3	4	5	6	7	8	9	10
1	6	4	2	3	@5	@4	@3	5	@2	@6
2	5	@6	@1	@5	4	3	6	@4	1	@3
3	@4	5	4	@1	6	@2	1	@6	@5	2
4	3	@1	@3	@6	@2	1	5	2	6	@5
5	@2	@3	6	2	1	@6	@4	@1	3	4
6	@1	2	@5	4	@3	5	@2	3	@4	1

TTP: swap partial teams

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	4	3	6	@4	@1	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	@2	1	5	2	@6	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	4	3	6	@4	@6	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	@2	1	5	2	@1	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

TTP: swap partial teams

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	4	3	6	@4	@1	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	@2	1	5	2	@6	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	4	3	6	@4	@6	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	@2	1	5	2	@1	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

TTP: swap partial teams

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	4	3	6	@4	@6	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	@2	1	5	2	@1	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@1	@5	4	3	6	@4	@6	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@3	@6	@2	1	5	2	@1	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

TTP: swap partial teams

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	4	3	6	@4	@6	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	@2	1	5	2	@1	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@1	@5	4	3	6	@4	@6	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@3	@6	@2	1	5	2	@1	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

TTP: swap partial teams

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	4	3	6	@4	@6	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	@2	1	5	2	@1	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@1	@5	4	3	6	@4	@6	@3
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@3	@6	@2	1	5	2	@1	@5
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

TTP: swap partial teams

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@3	@6	4	3	6	@4	@6	@5
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@1	@5	@2	1	5	2	@1	@3
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@1	@5	4	3	6	@4	@6	@3
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@3	@6	@2	1	5	2	@1	@5
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

TTP: swap partial teams

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	4	3	@5	@4	@3	5	2	@6
2	5	1	@1	@5	4	3	6	@4	@6	@3
3	@4	5	2	@1	6	@2	1	@6	@5	4
4	3	6	@3	@6	@2	1	5	2	@1	@5
5	@2	@3	6	4	1	@6	@4	@1	3	2
6	@1	@4	@5	2	@3	5	@2	3	4	1

T-R	1	2	3	4	5	6	7	8	9	10
1	6	@2	2	3	@5	@4	@3	5	4	@6
2	5	1	@1	@5	4	3	6	@4	@6	@3
3	@4	5	4	@1	6	@2	1	@6	@5	2
4	3	6	@3	@6	@2	1	5	2	@1	@5
5	@2	@3	6	2	1	@6	@4	@1	3	4
6	@1	@4	@5	4	@3	5	@2	3	2	1

Image segmentation

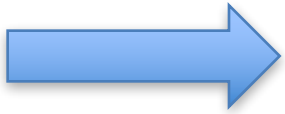
Image

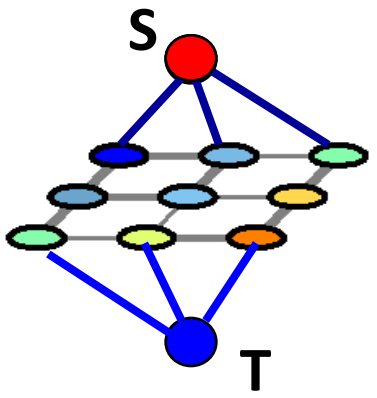
Segmentation

Following slides based on the tutorial by P. Kohli

Image segmentation

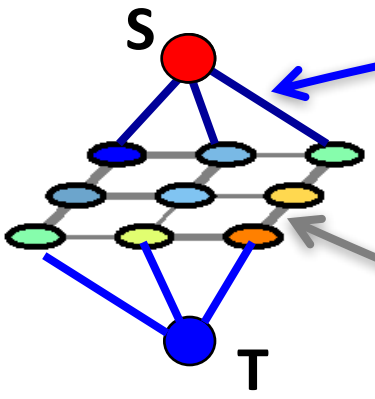
Image





Construct a graph

Image segmentation



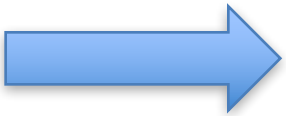
$$E(x) = \sum_i c_i x_i + \sum_{i,j} c_{ij} x_i (1 - x_j)$$

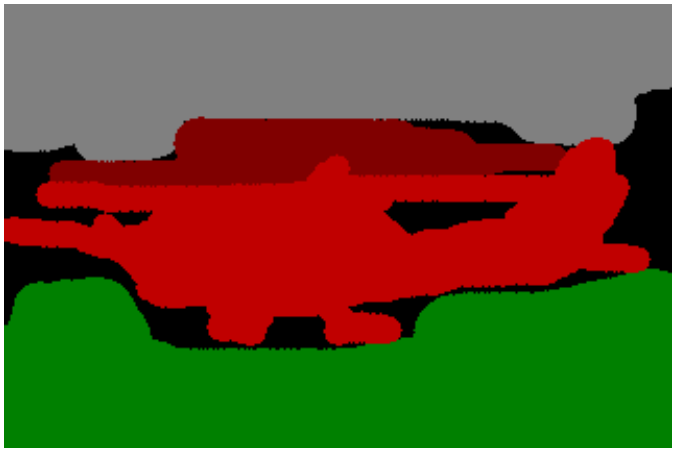
$$E : \{0,1\}^n \rightarrow \mathbb{R}$$

Set the edge weights

Image segmentation

What if we have multiple objects?





Image

Segmentation

$$E(x) = \sum_{i \in \mathcal{V}} U_i(x_i) + \sum_{(i,j) \in \mathcal{E}} c_{ij}[x_i \neq x_j]$$

$$x_i \in \{1, \ldots, P\}$$

Minimization is NP-hard if
P > 2 and the graph is cyclic

Image segmentation

Task: assign label to each pixel of an image

$$E(x) = \sum_{i \in \mathcal{V}} U_i(x_i) + \sum_{(i,j) \in \mathcal{E}} c_{ij}[x_i \neq x_j]$$

$$x_i \in \{1, \ldots, P\}$$

Local search:

maintain labeling x

make a move such that the objective decreases

Image segmentation

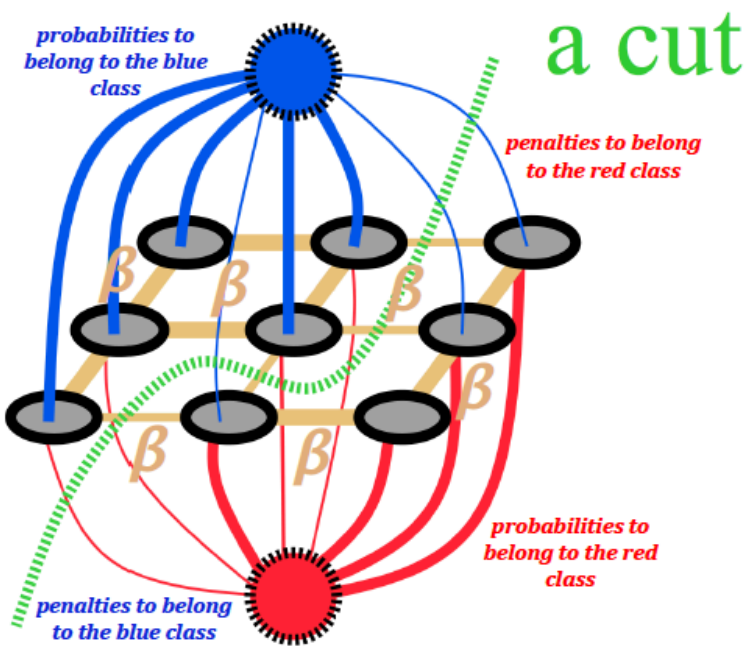
Task: assign a label to each pixel of the input image

Simple moves:

- change value of a variable x_i
- change values of several variables
  exponential in the number of variables
- global moves: α-expansion

MRF-based segmentation

- **Two-class segmentation/classification problem**
  - map an image onto a graph
  - minimize a submodular energy of the form:



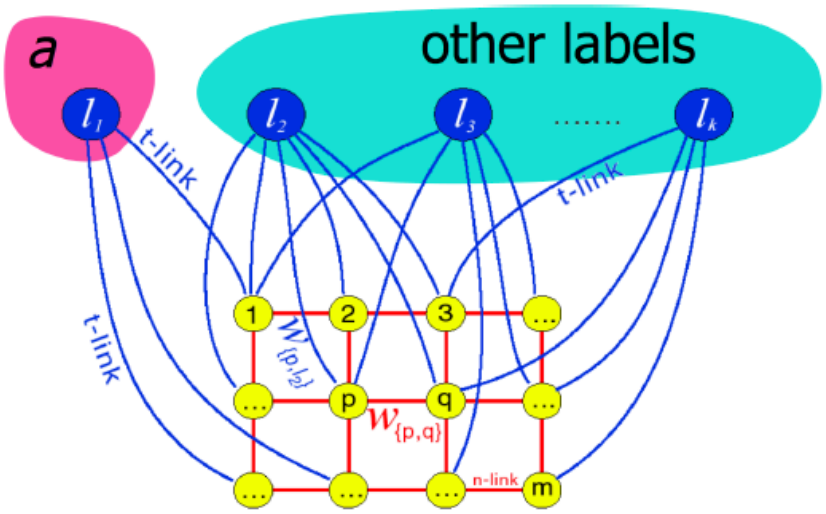
$$E(L) = \sum_{\text{pixels } i} V_i(L_i) + \sum_{i \sim j} W_{i,j}(L_i, L_j)$$

- L_i^t = label of pixel i
- individual potential $V_i(L_i)$ = penalty for a pixel i to have a label L_i
- $W_{i,j}(L_i, L_j)$ = interaction term between neighboring pixels i and j
- **Potts model:** $W_{i,j}(L_i, L_j) = \beta\,(1 - \delta(L_i, L_j))$, $\delta(a, b) = 1$ for $a = b$ and $\delta(a, b) = 0$ otherwise

- Globally-optimal solution is computed with a **graph cut** [Boykov04]

α-expansion

- **Basic idea:** break computation into a **sequence of binary cuts**

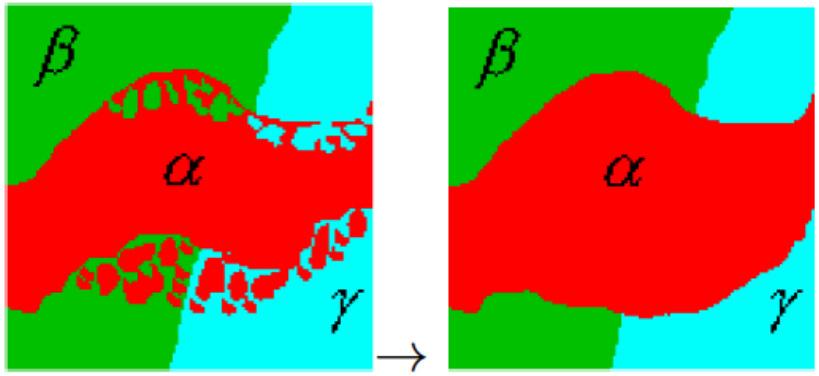


1. Start with any initial solution
2. For each label *alpha* in any (e.g. random) order:
   - Compute optimal *alpha*-expansion move (binary cut)
   - Decline the move if there is no energy decrease
   - Stop when no expansion move would decrease energy

- Guaranteed approximation quality [Veksler01]
  - within a factor of 2 from the global minima (Potts model)

α-expansion

- **Basic idea:** break computation into a **sequence of binary cuts**



1. Start with any initial solution
2. For each label *alpha* in any (e.g. random) order:
   - Compute optimal *alpha*-expansion move (binary cut)
   - Decline the move if there is no energy decrease
   - Stop when no expansion move would decrease energy

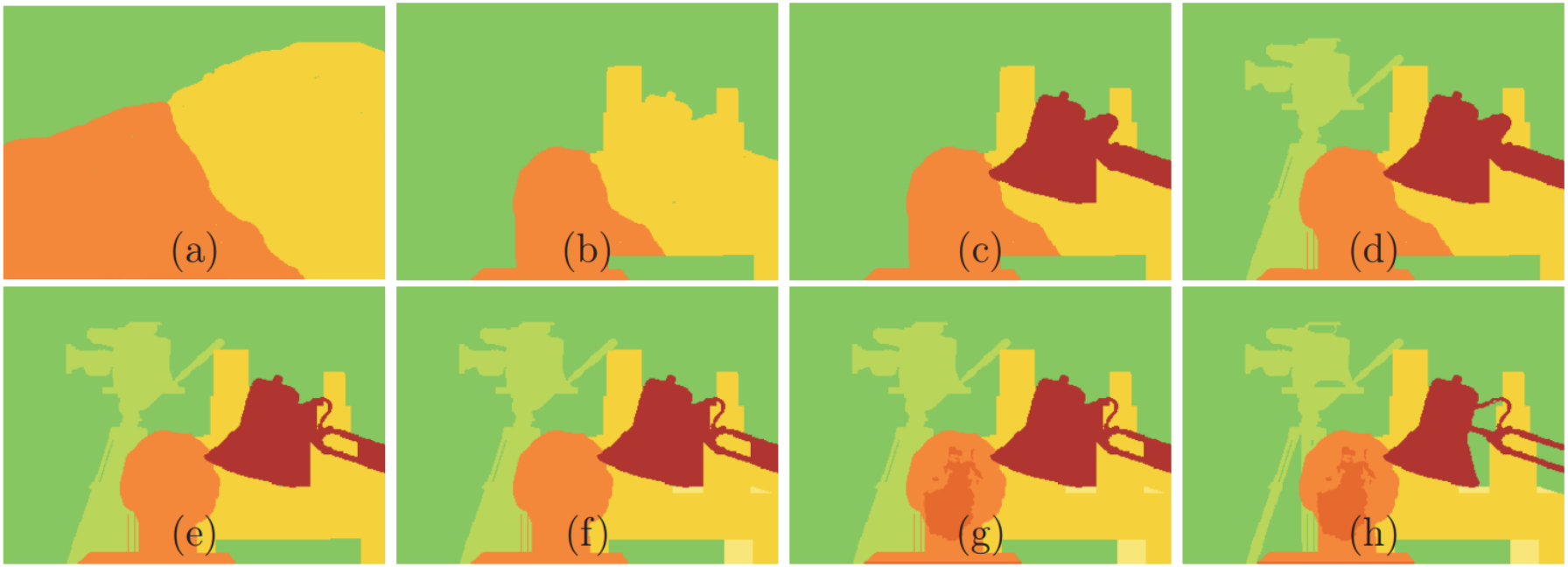
- Guaranteed approximation quality [Veksler01]
  - within a factor of 2 from the global minima (Potts model)

α-expansion

Task: assign label to each pixel of an image

Expansion move:

allow all the variables to take label α



α-expansion

Task: assign label to each pixel of an image

Expansion move: allow all the variables to take label α

Convert function E(x) into F(y) where y are binary:
$y_i = 1$ when $x_i = \alpha$

$$E(x) = \sum_{i \in \mathcal{V}} U_i(x_i) + \sum_{(i,j) \in \mathcal{E}} c_{ij}[x_i \neq x_j]$$

$$F(y) = \sum_{i \in \mathcal{V}} (y_i U_i(\alpha) + (1 - y_i) U_i(x_i)) + \sum_{(i,j) \in \mathcal{E}} \Big((1 - y_i)(1 - y_j) c_{ij}[x_i \neq x_j] + y_i(1 - y_j) c_{ij}[\alpha \neq x_j] + (1 - y_i) y_j c_{ij}[x_i \neq \alpha] \Big)$$

α-expansion

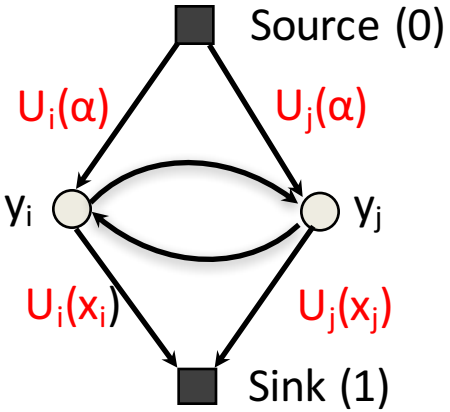
Task: assign label to each pixel of an image

Expansion move: allow all the variables to take label α

$$F(y) = \sum_{i \in \mathcal{V}} (y_i U_i(\alpha) + (1 - y_i) U_i(x_i)) + \sum_{(i,j) \in \mathcal{E}} \Big((1 - y_i)(1 - y_j) c_{ij} [x_i \neq x_j]$$
$$+ y_i (1 \quad y_j) c_{ij} [\alpha \neq x_j]$$
$$+ (1 - y_i) y_j c_{ij} [x_i \neq \alpha] \Big)$$

Build a graph to minimize F(y) with s-t min-cut

Unary potentials as in the binary case



α-expansion

Task: assign label to each pixel of an image

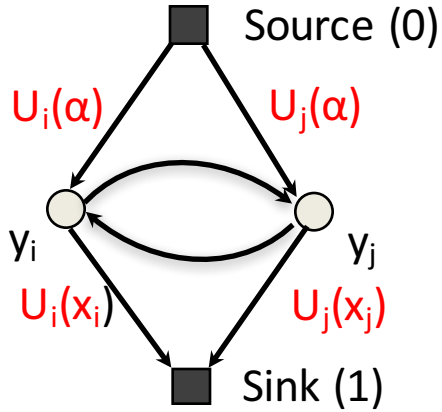
Expansion move: allow all the variables to take label α

$$F(y) = \sum_{i \in \mathcal{V}} (y_i U_i(\alpha) + (1 - y_i) U_i(x_i)) + \sum_{(i,j) \in \mathcal{E}} \Big((1 - y_i)(1 - y_j) c_{ij} [x_i \neq x_j] + y_i (1 - y_j) c_{ij} [\alpha \neq x_j] + (1 - y_i) y_j c_{ij} [x_i \neq \alpha] \Big)$$

Build a graph to minimize F(y) with s-t min-cut

Arc j -> i: $c_{ij}[\alpha \neq x_j]$

Arc i -> j: $c_{ij}[x_i \neq \alpha]$



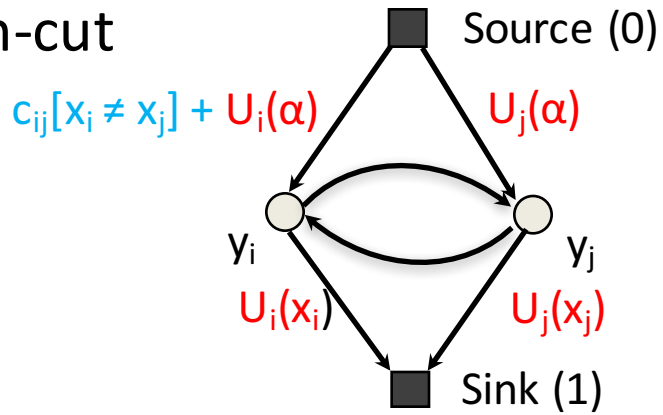
α-expansion

Task: assign label to each pixel of an image

Expansion move: allow all the variables to take label α

$$F(y) = \sum_{i \in \mathcal{V}} (y_i U_i(\alpha) + (1 - y_i) U_i(x_i)) + \sum_{(i,j) \in \mathcal{E}} \Big((1 - y_i)(1 - y_j) c_{ij} [x_i \neq x_j] + y_i (1 - y_j) c_{ij} [\alpha \neq x_j] + (1 - y_i) y_j c_{ij} [x_i \neq \alpha] \Big)$$

Build a graph to minimize F(y) with s-t min-cut



Arc j -> i: $c_{ij}[\alpha \neq x_j]$

Arc i -> j: $c_{ij}[x_i \neq \alpha] - c_{ij}[x_i \neq x_j]$
positive if $x_i \neq \alpha$ and $c_{ij} \geq 0$