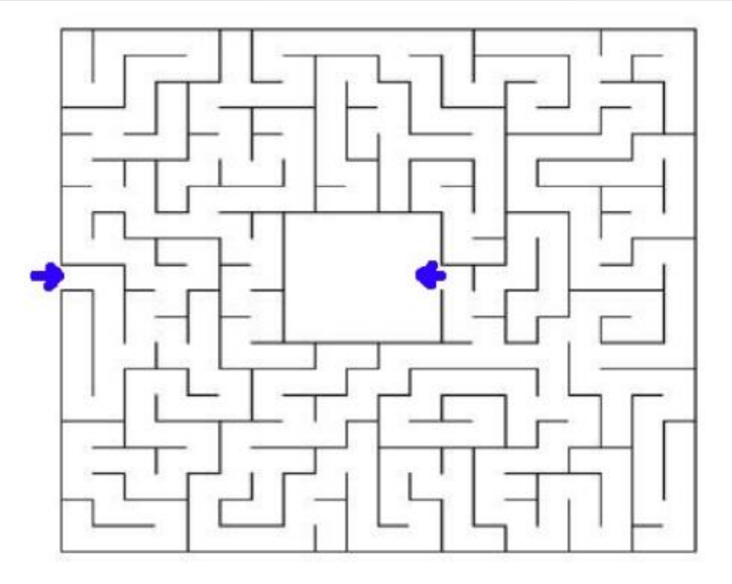
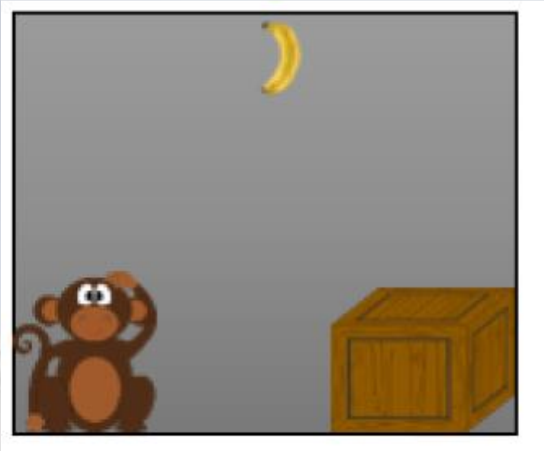
Methods of Artificial Intelligence

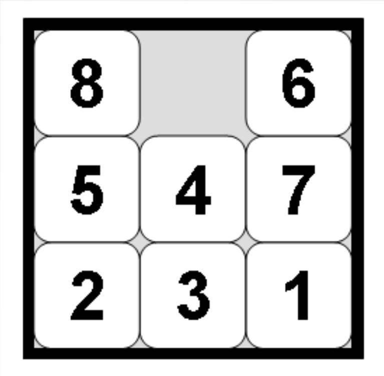
Local Search

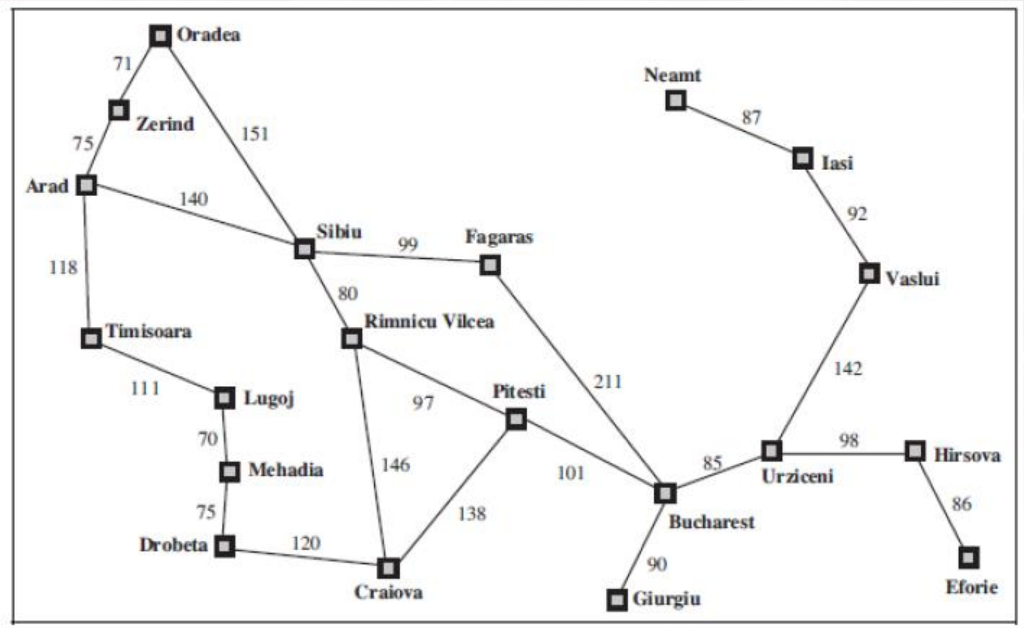
From Classical Search to Local Search

Search Problems

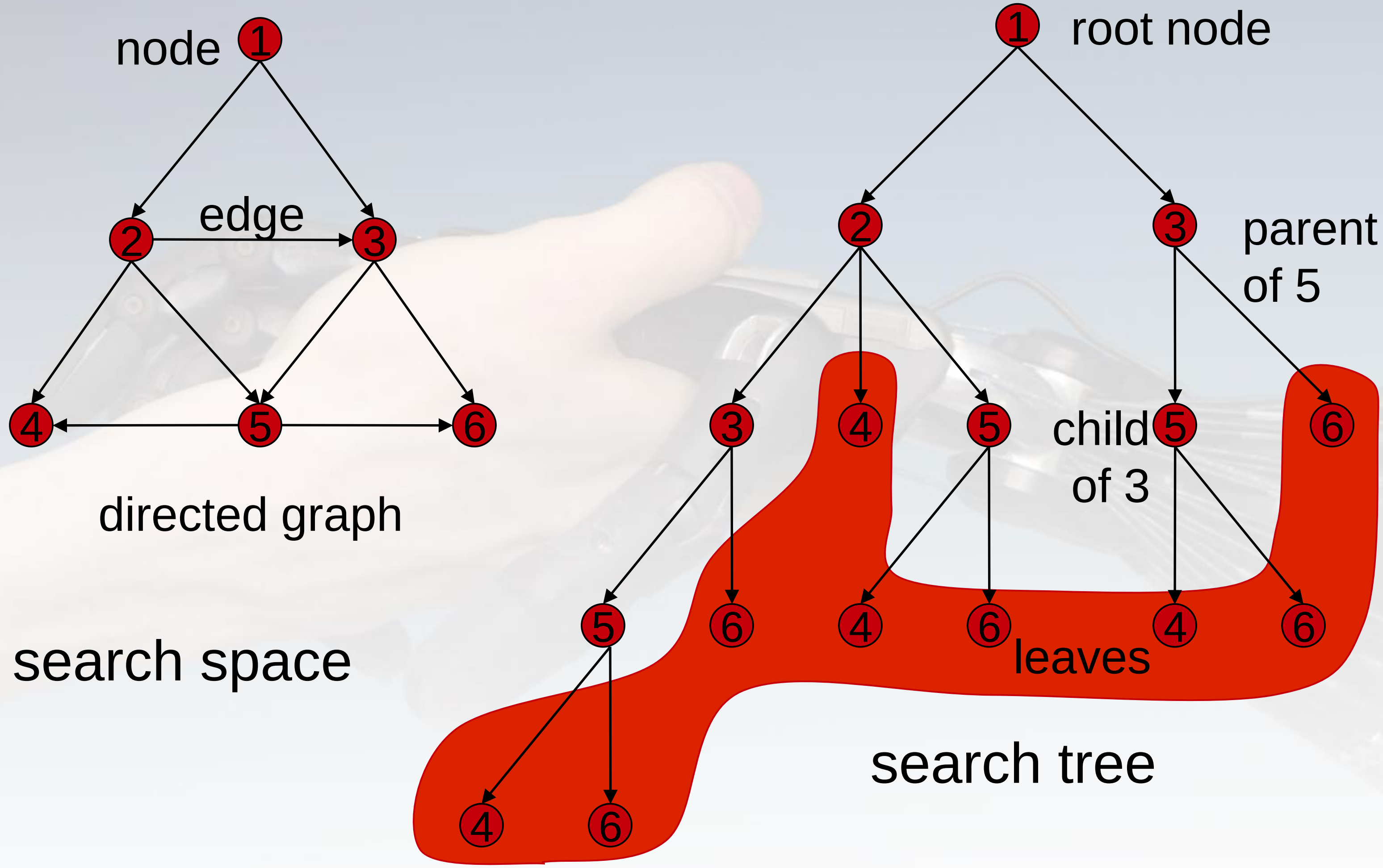








Search Space and Search Tree



Classical Search Paradigms

- Uninformed Search
  - No information about search space topology (blind search)
- Informed Search
  - We can estimate distance to goal states

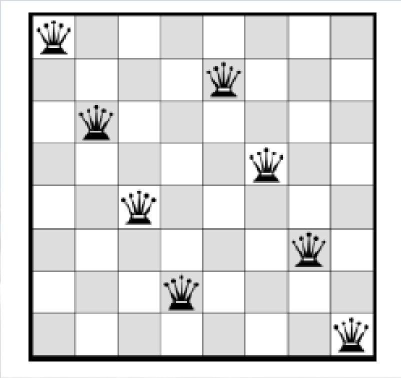
Classical Search Algorithms

- Uninformed (blind)
  - Random search
  - Systematic search
    - Depth-first search (DFS)
    - Breadth-first search (BFS)
    - Variations
      - Uniform-cost search
      - Depth-limited search
  - Features
    - no information about the path cost from the current state to the goal state

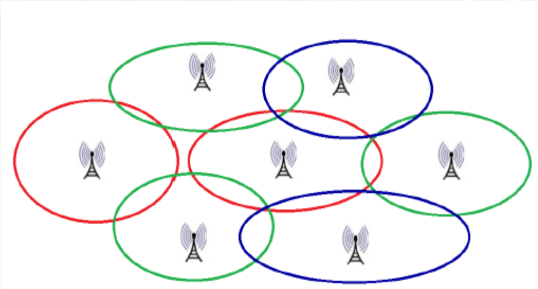
- Informed (heuristic)
  - Systematic search
    - Greedy best-first search
    - A* search
  - Features
    - heuristic information about the path cost from current state to goal state (background knowledge)

From Classical Search to Local Search

- In classical search, we look for an (optimal) sequence of actions that leads from an initial state to a goal state



- However, in many problems, there is
  - no given initial state (CSPs)
  - no obvious transition model
  - no simple goal test (optimization problems)



- In particular, path from initial to goal state is often irrelevant

- Local Search is often better suited to address these problems

Local Search Main Ideas

- States often correspond to variable assignments
- We define neighborhood that determines what other assignments can be reached from a given assignment
- Local Search works roughly as follows:
  - start from some initial assignment (e.g. random choice)
  - select a promising neighbor of the current assignment
  - continue selecting neighbors until we cannot improve anymore

Local Search Applications

- (Discrete) Optimization Problems
  - Production planning (maximize profit)
  - Scheduling (minimize conflicts)
  - Machine Learning (minimize classification error)
  - …
- Constraint Satisfaction Problems

Algorithms Overview

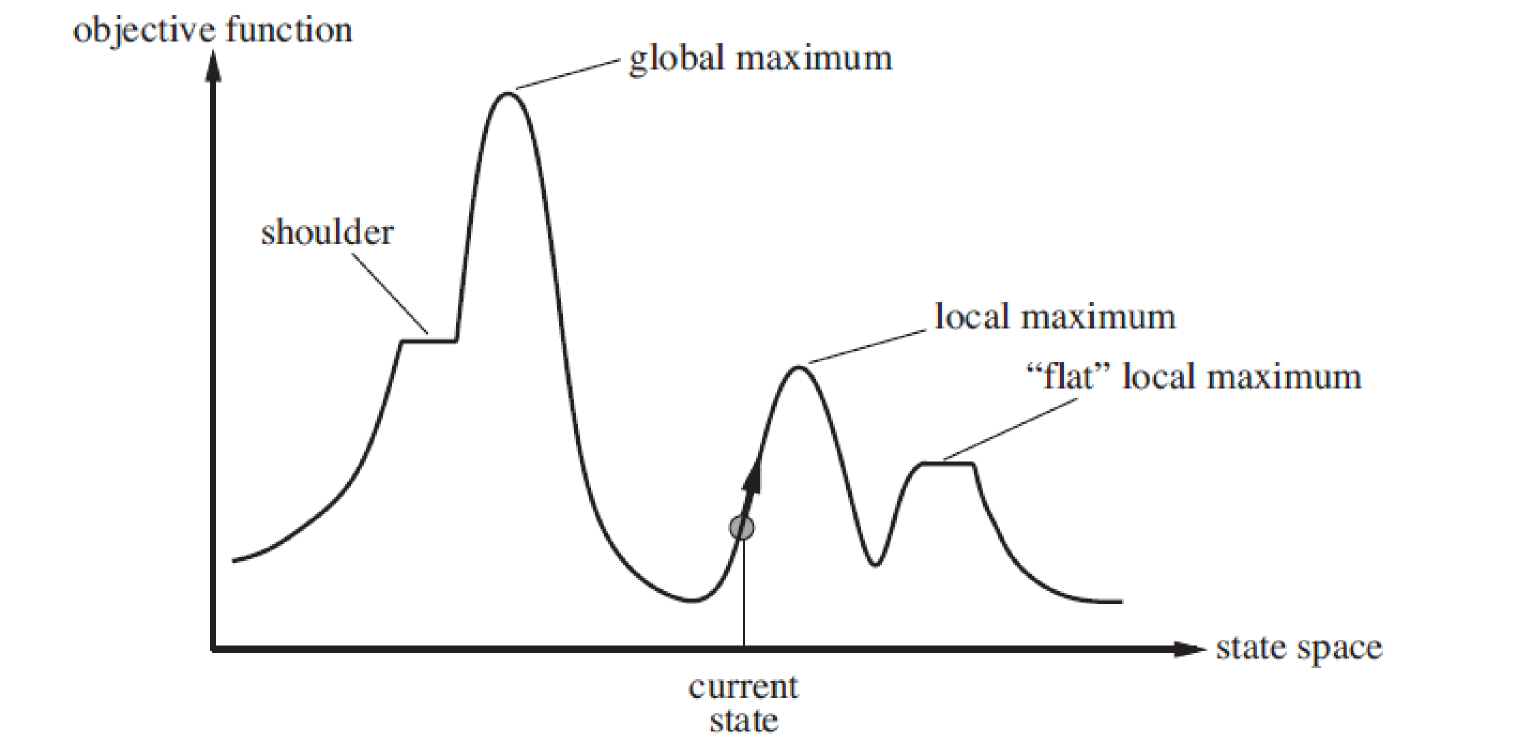
We will discuss the following local search algorithms:

- Hill-Climbing
- Simulated Annealing
- Local Beam Search
- Genetic Algorithms

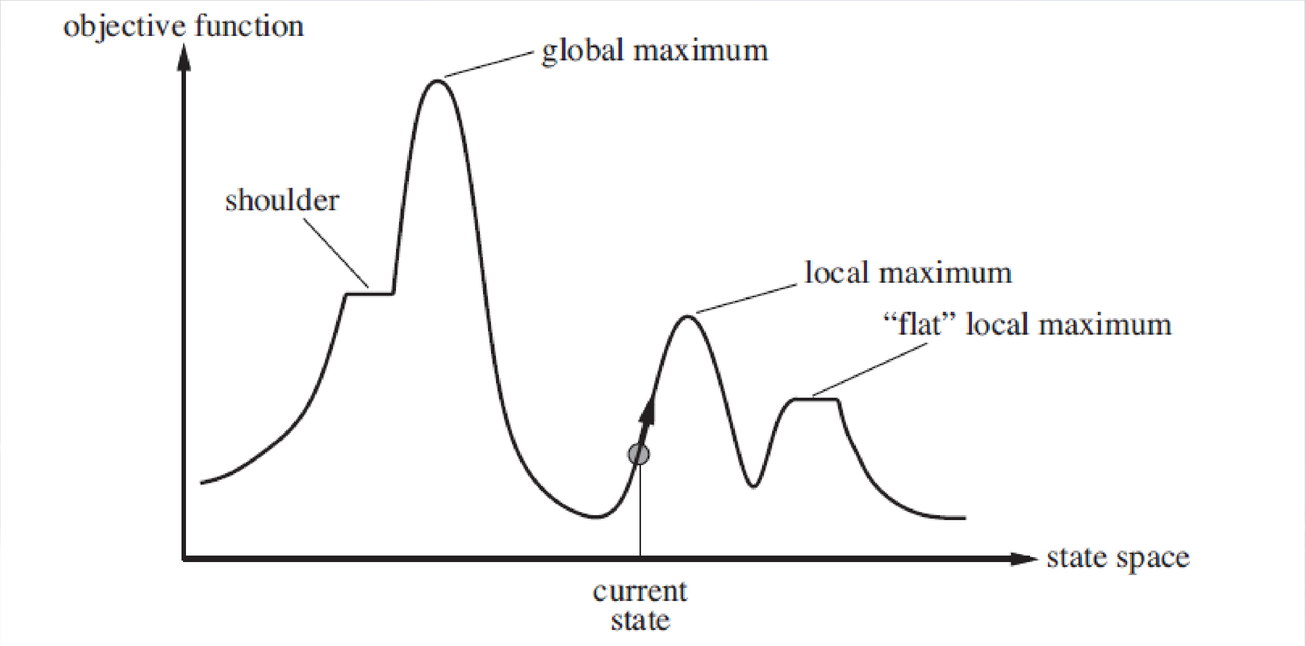
State-space Landscape and Hill-Climbing

State-space Landscape

- Suppose, we want to maximize real function f(x) (objective function)
- we can do so by starting from random point and following ascent direction (gradient ascent algorithm)

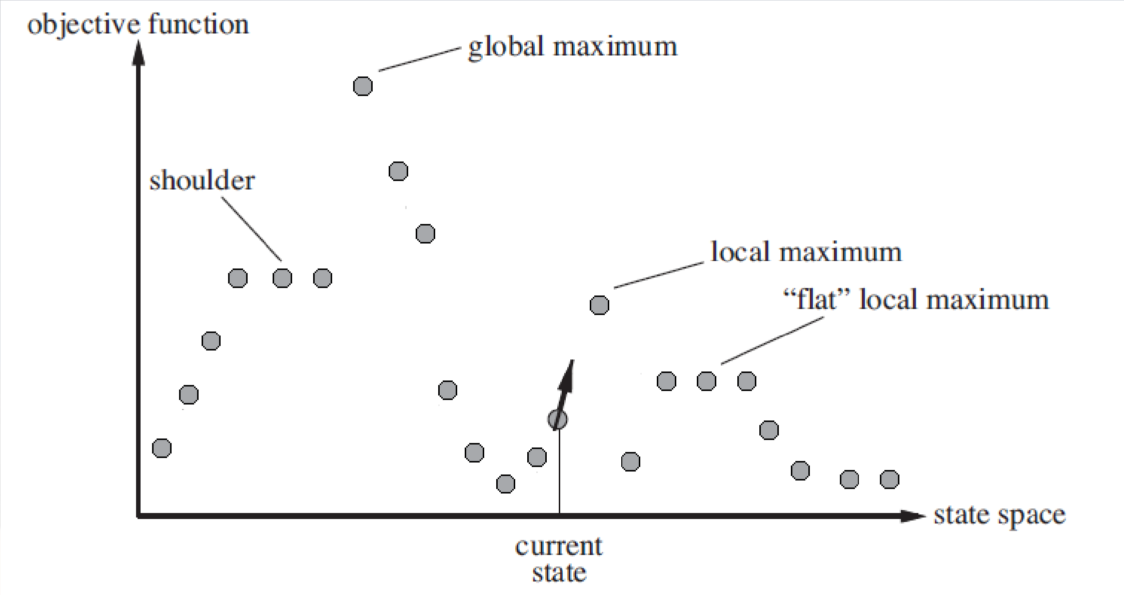


Challenges



- if we stop if no ascent is possible anymore, we may end up in a non-global local maximum or plateaux
- If we try to reach end of plateaux, we may search forever

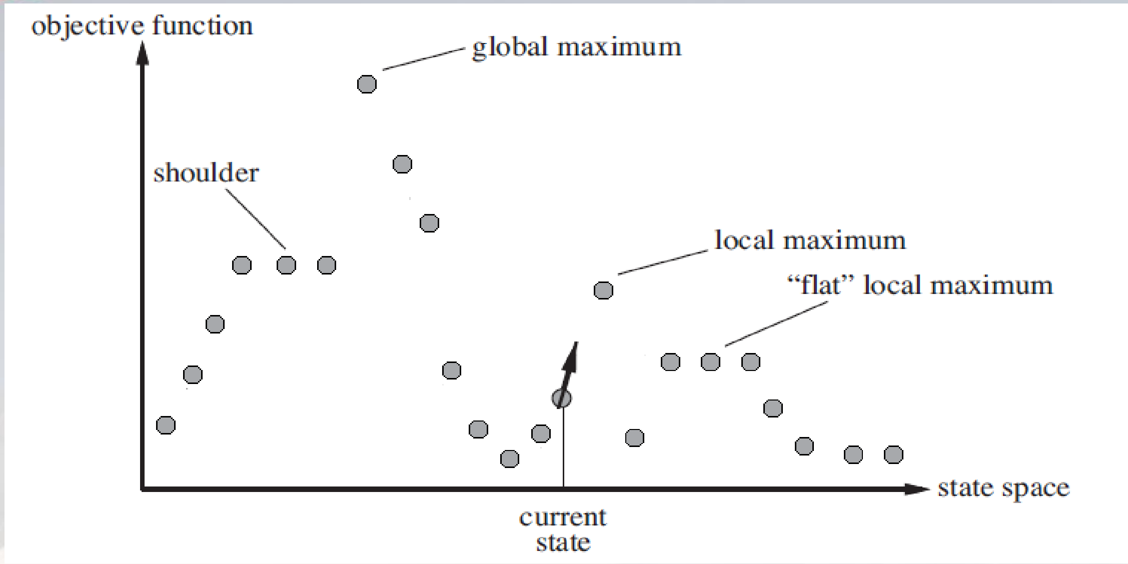
Discrete State Spaces



- We will often consider discrete state spaces here

  (we cannot move continuously through state space)
- Often our state space is even finite, but exponentially large
- However, some problems of continuous spaces remain

Navigating in Discrete State Spaces



- In (unconstrained) continuous state spaces, we can move in an arbitrary direction for an arbitrary length
- In discrete spaces, every state is associated with a neighborhood that contains other states
- We move through state space, by selecting a neighbor

Hill-Climbing

current ← select random initial state
do
  neighbor ← neighbor of *current* with highest value
  if value(*neighbor*) ≤ value(*current*)
    return *current*
  current←*neighbor*
until termination condition is met

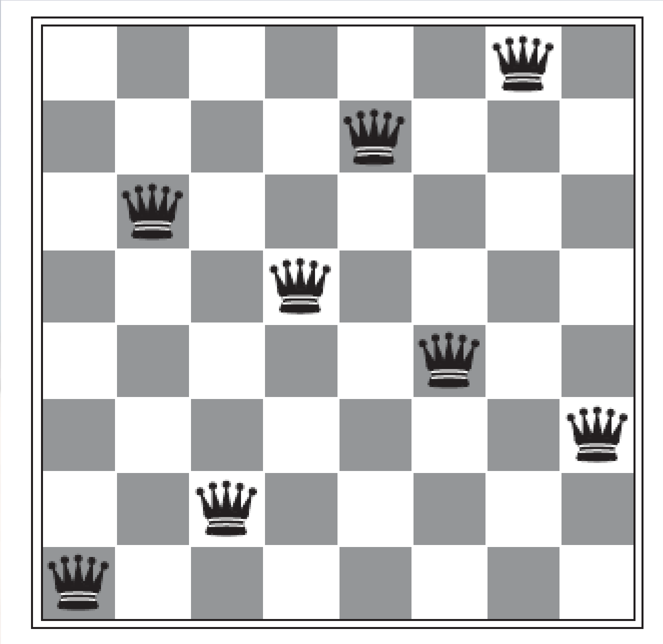
- Hill-Climbing can be regarded as a discrete state-space version of gradient ascent
- In each step, we select neighbor with highest value

Hill-Climbing Termination Conditions

current ← select random initial state
do
  neighbor ← neighbor of *current* with highest value
  if value(*neighbor*) ≤ value(*current*)
    return *current*
  current←*neighbor*
until termination condition is met

- algorithm may end up in non-global local maximum or plateaux
- termination condition can bound the maximum number of search steps or search time

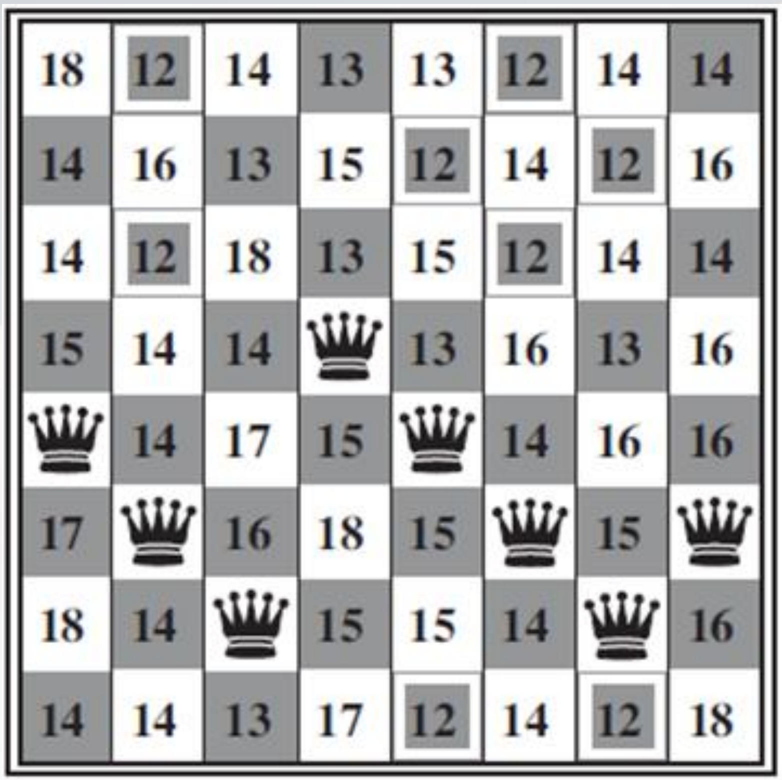
Example: 8-Queens Problem



- State-Space: board configurations with one queen per column (tuple (8,3,7,4,2,5,1,6) corresponds to board configuration above)
- Neighborhood: configurations that can be obtained by moving a single queen to another field in the same column (hence, every state has 8 * 7 = 56 neighbors)

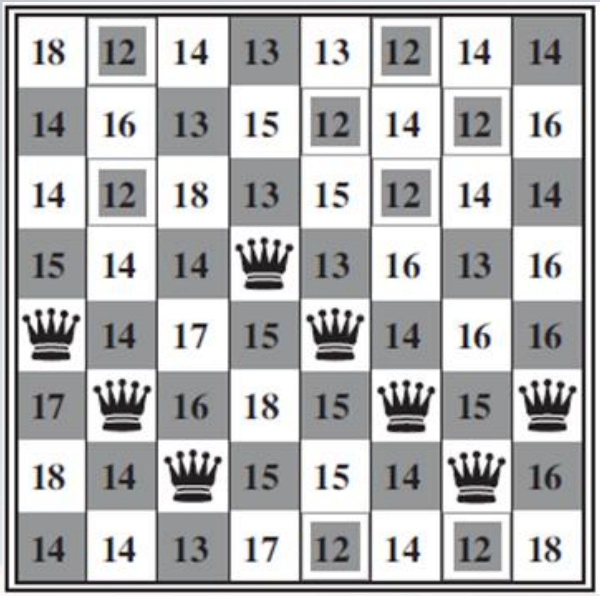
Objective Function

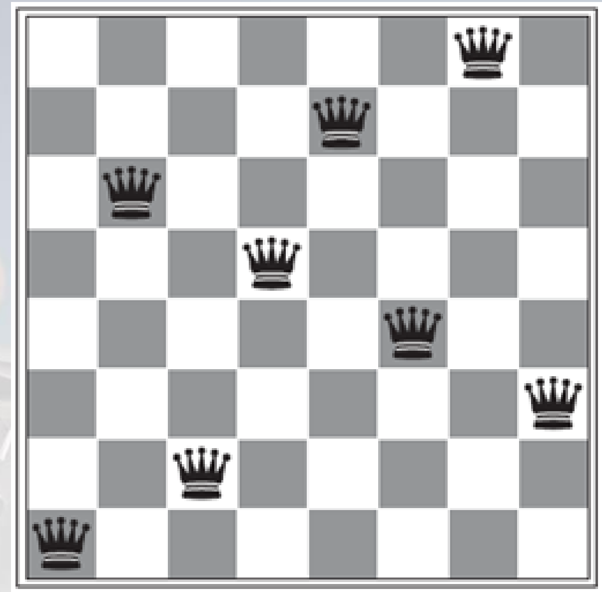
- Objective Function: number of pairs of queens that can attack each other directly or indirectly *(maximize negative objective function)*



- Queen in column 1 can directly attack queen in column 2
- Queen in column 1 can indirectly attack queen in column 3
- State has value -17
- Numbers show values of neighbors

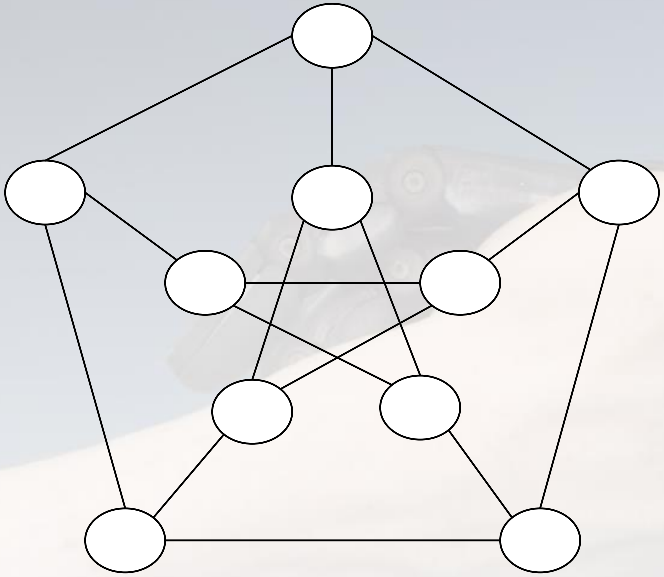
Local Optima





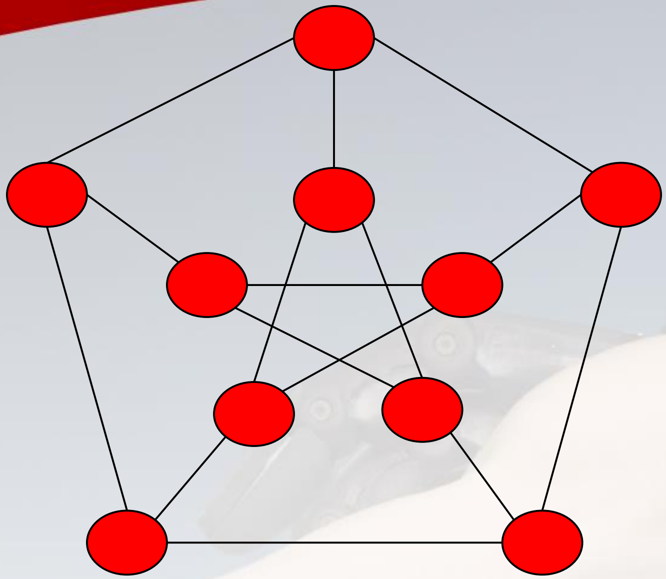
- Hill Climbing goes from left state with value -17 to right state with value -1 *(queen 4 attacks queen 7)* in only 5 steps
- However, right state is only locally optimal

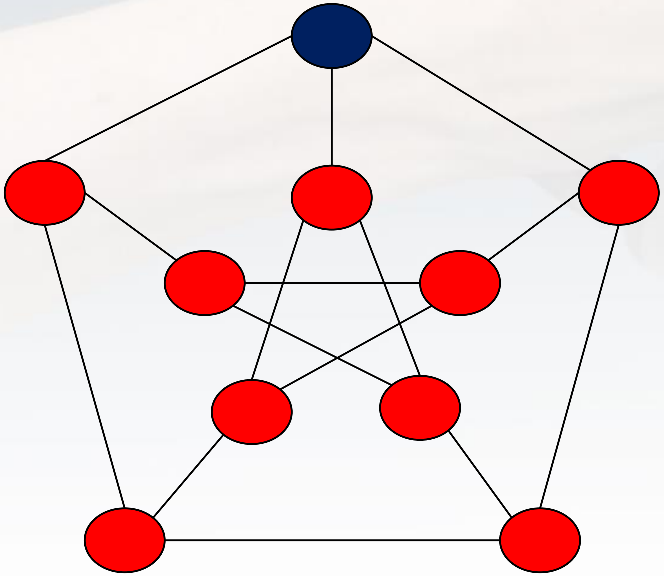
Excercise: Graph Coloring



- Problem: assign a color from {1,2,3} to each node such that no two adjacent nodes have the same color
- Define
  - State space
  - Neighborhood
  - Objective function
- Perform Hill-climbing search from an initial state where all nodes have color 1

Solution: Local Search Problem

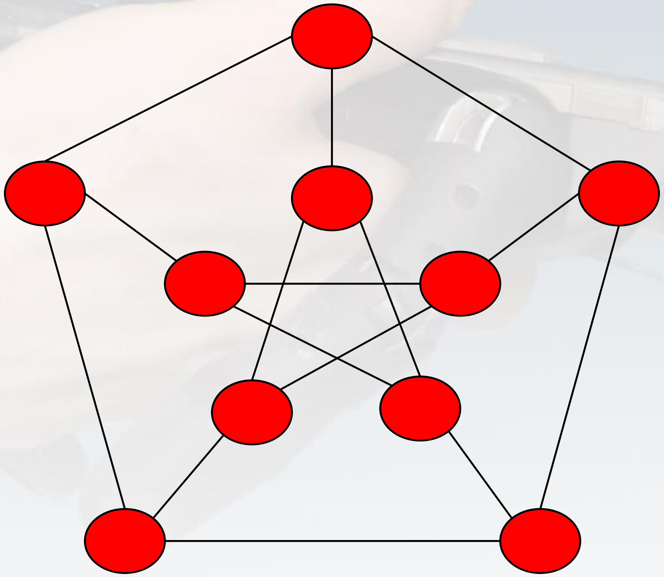




- State Space: each assignment of colors from {1,2,3} to nodes. Formally, we can use 10-tuples like (1,1,1,1,1,1,1,1,1,1)
- Neighborhood: change color of one node (each state has 10 * 2 = 20 neighbors)
- Objective function: number of pairs of adjacent nodes with equal colors (maximize negative objective function)

Solution: Hill-Climbing Search

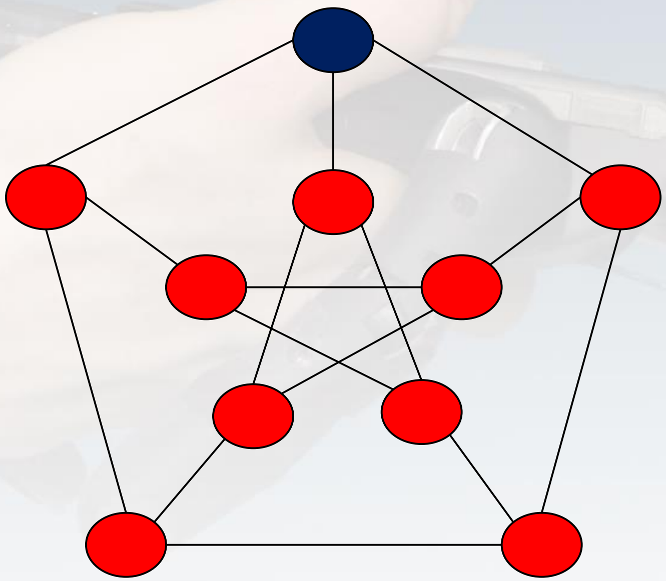
- Objective function can be increased by at most 3



State (1,1,1,1,1,1,1,1,1,1)
Objective function: -15

Solution: Hill-Climbing Search

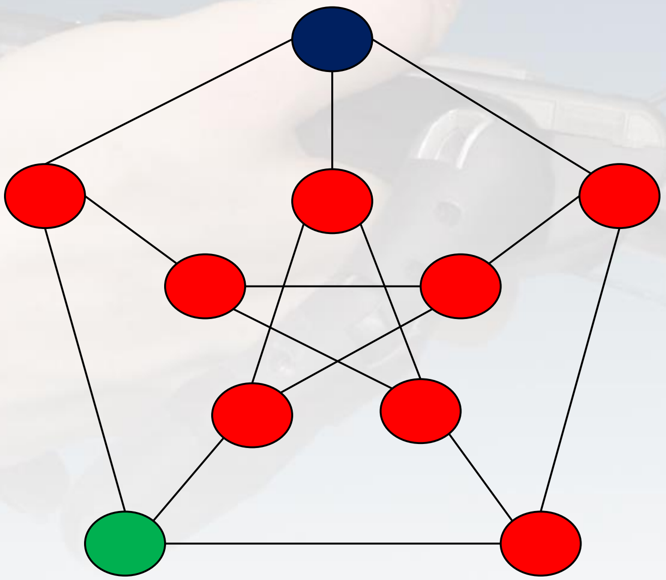
- Objective function can be increased by at most 3



State (2,1,1,1,1,1,1,1,1,1)
Objective function: -12

Solution: Hill-Climbing Search

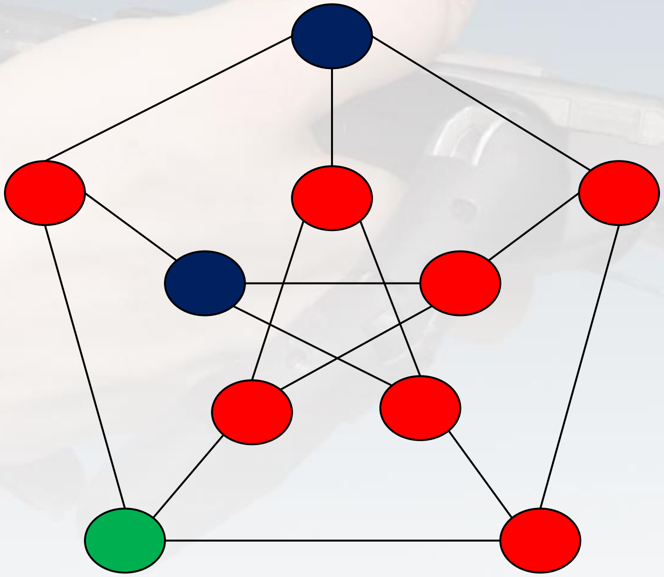
- Objective function can be increased by at most 3



State (2,1,1,1,1,1,1,1,3,1)
Objective function: -9

Solution: Hill-Climbing Search

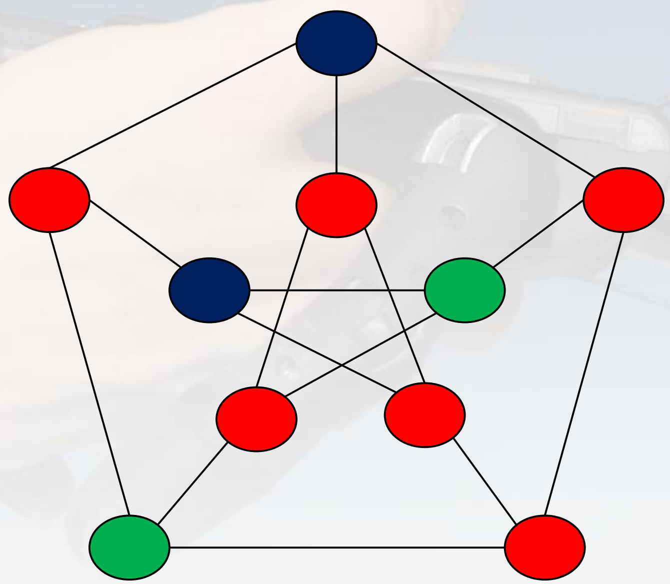
- Objective function can be increased by at most 2



State (2,1,1,1,2,1,1,1,3,1)
Objective function: -6

Solution: Hill-Climbing Search

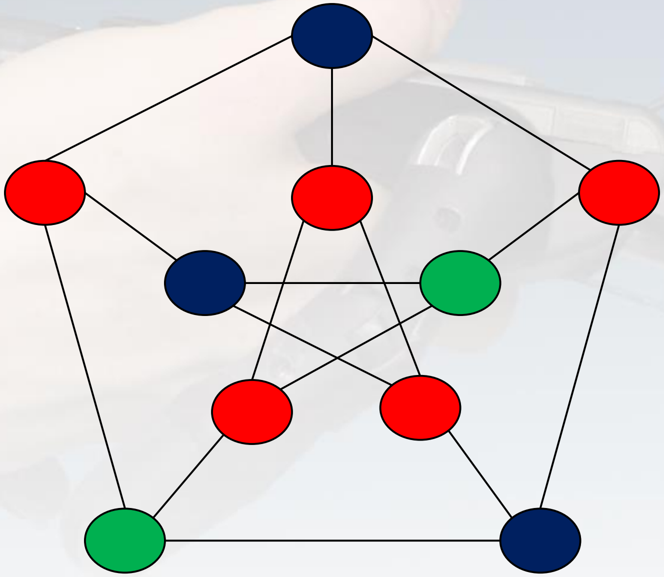
- Objective function can be increased by at most 2



State (2,1,1,1,2,3,1,1,3,1)
Objective function: -4

Solution: Hill-Climbing Search

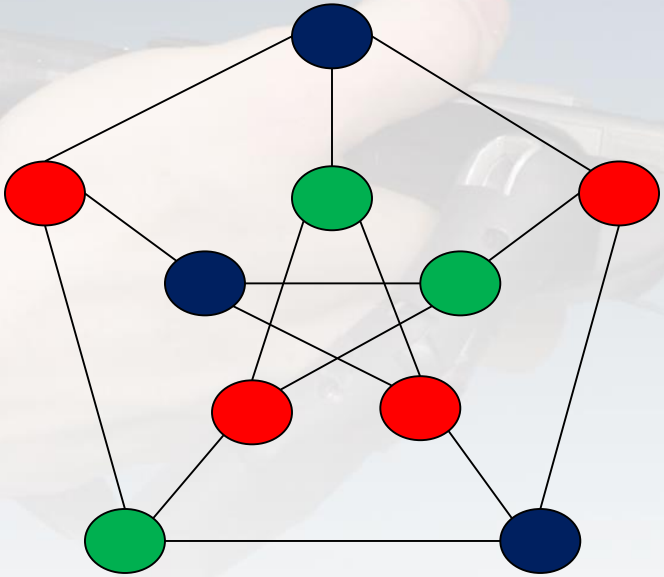
- Objective function can be increased by at most 2



State (2,1,1,1,2,3,1,1,3,2)
Objective function: -2

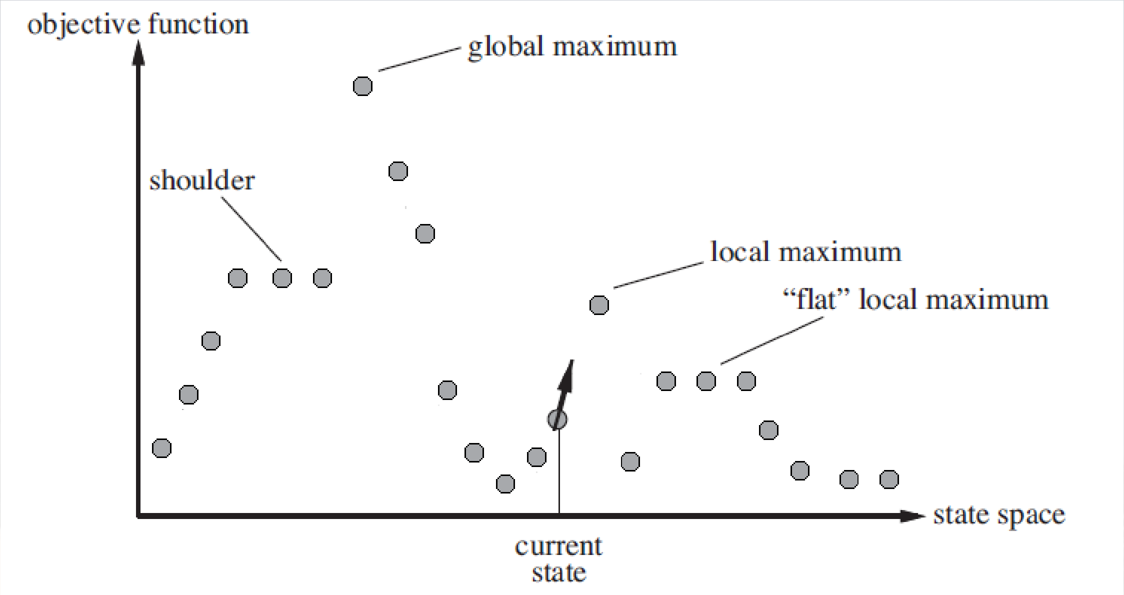
Solution: Hill-Climbing Search

- Objective function cannot be increased anymore



State (2,1,3,1,2,3,1,1,3,2)
Objective function: 0 (global maximum)

Variants of Hill-Climbing



- Hill-Climbing in its basic form can perform poorly in many problems because there is a high risk of ending up in non-global local maxima
- There exist several variants that can alleviate the problem

Stochastic Hill-Climbing

```
current ← select random initial state
do
   neighbor ← random neighbor of current with higher value
   if neighbor = null
       return current
   current←neighbor
until termination condition is met
```

- Instead of selecting neighbor with maximum value, we select random neighbor that improves value
- Probability of selection can increase with value
- Compromise between random search and Hill-Climbing

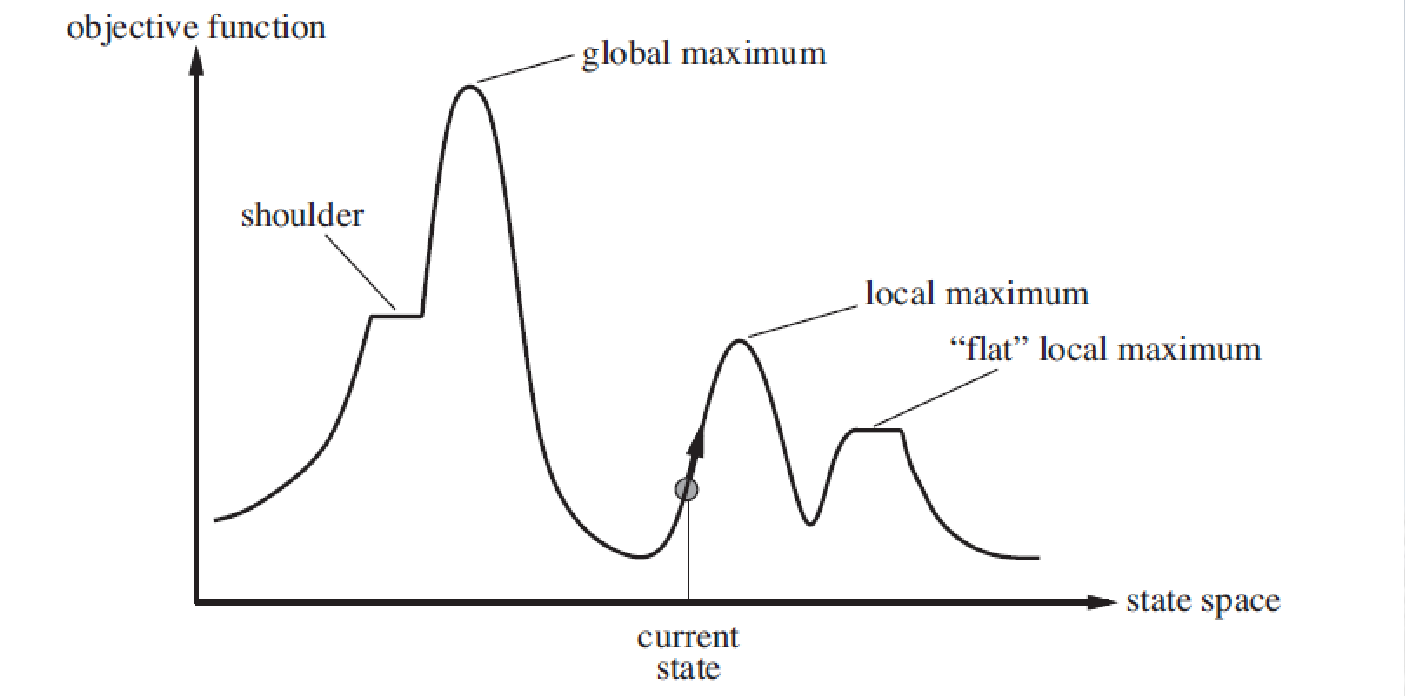
Other Variants

- First-choice Hill-climbing: in neighbor selection step, pick first neighbor that improves objective function
  (also useful if neighborhood is too large to enumerate all neighbors)

- Random-restart (or parallel) Hill-Climbing: perform *n* Hill-climbing searchs starting from randomly generated initial states
  (as *n* goes to infinity, probability of finding global optimum goes to 1)

Simulated Annealing

Downhill Moves



- Hill-climbing never makes downhill moves
- However, as figure illustrates, downhill moves are sometimes necessary to find global maximum

Simulated Annealing Intuition

- Simulated Annealing is a randomized algorithm that allows downhill moves
- In the beginning, the probability for downhill moves is high (exploration)
- As search progresses, the probability decreases (exploitation)
- This process is modeled by means of a temperature variable that decreases (thus simulated annealing)

Simulated Annealing

```
current ← select random initial state
for t = 1 to ∞
   T ← schedule(t)
   if T = 0
      return current
   next ← randomly selected neighbor of current
   ΔE ←value(next) – value(current)
   if ΔE > 0
      current ←next
   else
      current ←next only with probability exp(ΔE/T)
```

Cooling Schedule

- schedule(t) controls how fast temperature reaches 0
- A slow cooling schedule increases probability of finding a high-quality solution but increases runtime
- Some simple scheduling schemes:
  - Stepwise Linear: start with arbitrary T and decrease T by a constant c in each step
  - Delayed Stepwise Linear: start with arbitrary T and decrease T by a constant c every k-th step
- Starting with T=∞ may also make sense to perform random exploration in the beginning

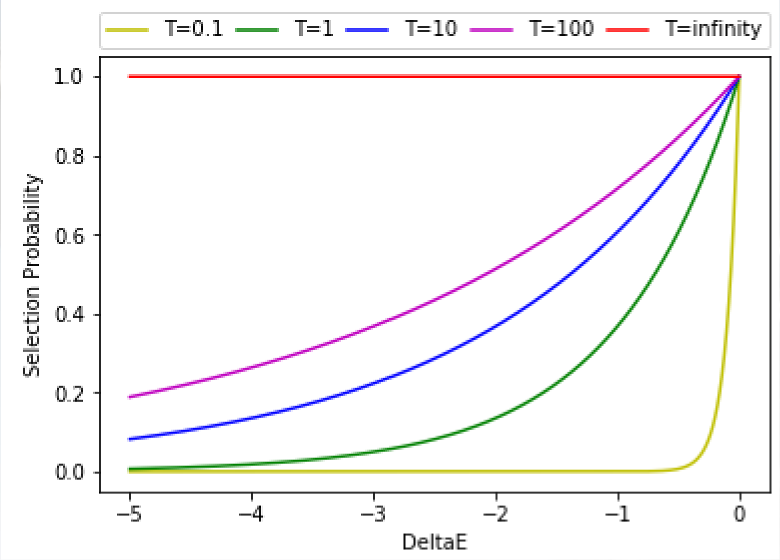
Neighbor Selection

- In each step, Simulated Annealing picks random neighbor
  - If neighbor improves objective, neighbor replaces current state
  - Otherwise, it replaces current state only with probability exp(ΔE/T),

    where ΔE = value(*next*) – value(*current*)

Neighbor Selection: ΔE

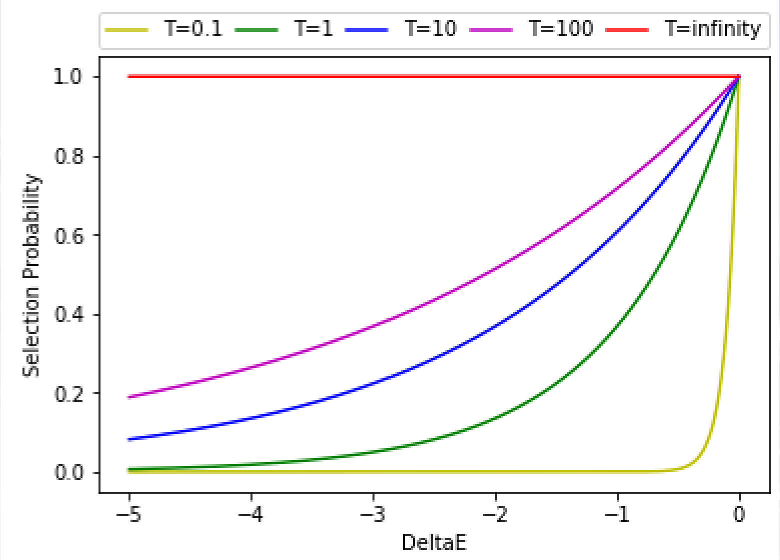
- Note that ΔE is always non-positive in else-branch
- Therefore, $0 \leq \exp(\Delta E/T) \leq \exp(0) = 1$
- Probability decreases exponentially with respect to difference



- For instance, for T=1, we have
  - $\Delta E = 0$: $\exp(0/1) = \exp(0) = 1$
  - $\Delta E = -1$: $\exp(-1/1) = 1/e = 0.37$
  - $\Delta E = -2$: $\exp(-2/1) = 1/e^2 = 0.14$

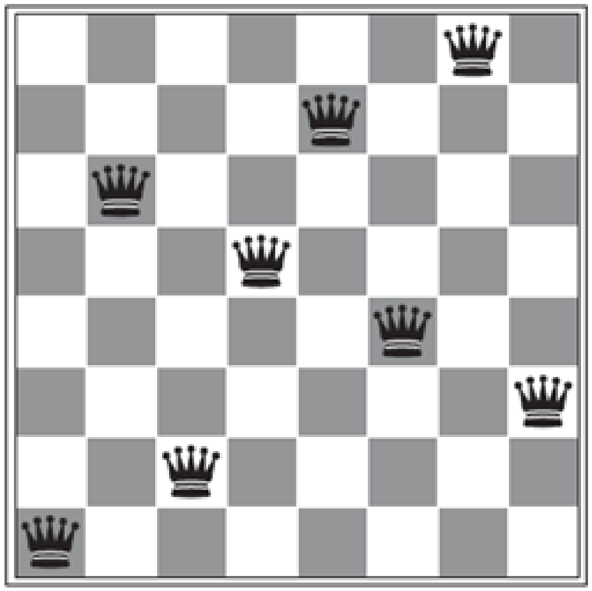
Neighbor Selection: Temperature

- As temperatue T decreases, probability further decreases
- For instance, assume ΔE = -2
  - T= ∞ : exp(-2/∞) = exp(0) = 1
  - T=100: exp(-2/100) = 0.98
  - T=10: exp(-2/10) = 0.81
  - T=1: exp(-2/1) = 0.13
  - T=0.1: exp(-2/0.1) = 0.002



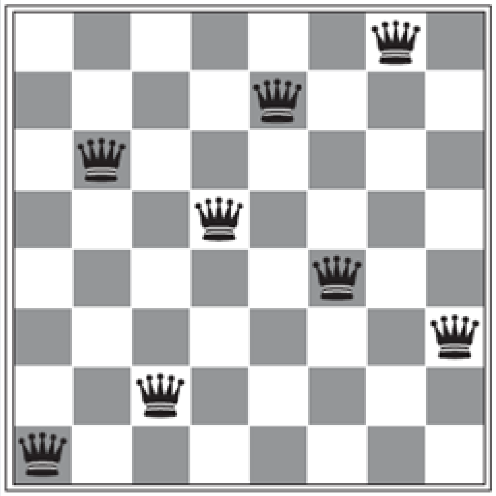
- Therefore, Simulated Annealing is similar to random exploration for high temperatures and similar to First-choice Hill-Climbing later

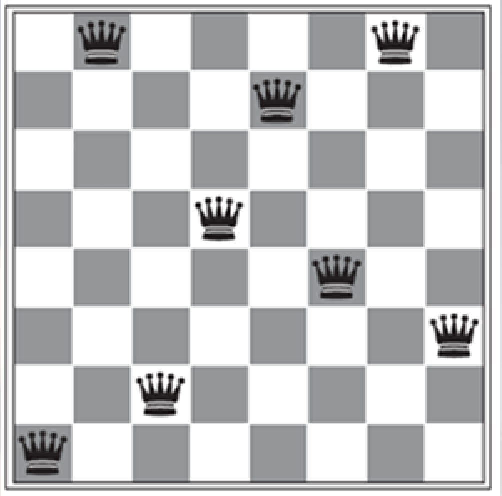
Exercise: Local Optima Revisited



- As we observed before, given state is locally optimal with value -1
- Compute ΔE for the neighbors obtained by
  - moving queen 2 to row 1 (State (8,1,7,4,2,5,1,6))
  - moving queen 4 to row 8 (State (8,3,7,8,2,5,1,6))
- Compute the probability of performing these moves for T=10 and T=1

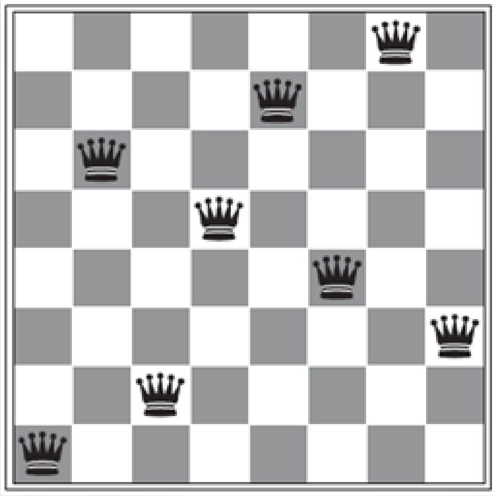
Solution: Local Optima Revisited

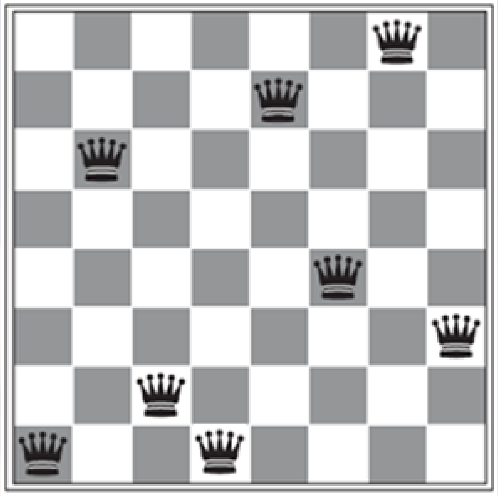




- When moving queen 2 to row 1, we add two new attacks
- We have ΔE = -3 – (-1) = -2
- The probabilities of performing these move are
  - Exp(-2/10) = 0.81 for T=10
  - Exp(-2/1) = 0.13 for T=1

Solution: Local Optima Revisited



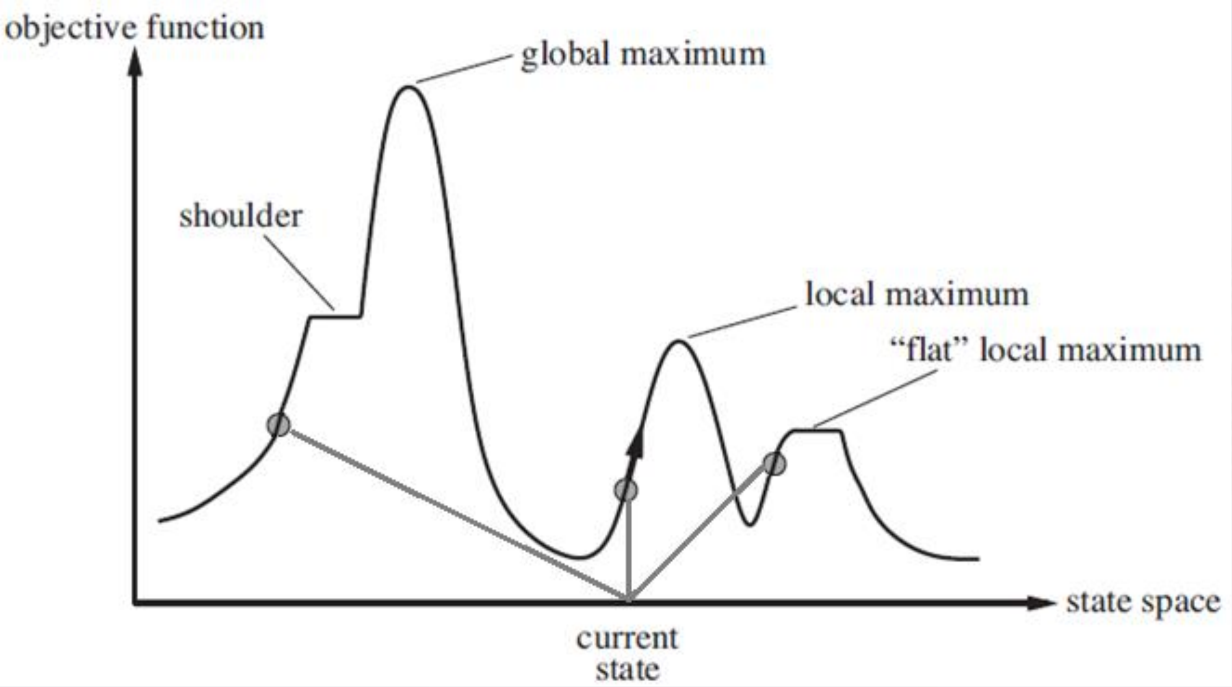


- When moving queen 4 to row 8, the value is -2
- We have $\Delta E = -2 - (-1) = -1$
- The probabilities of performing this move are
  - $\text{Exp}(-1/10) = 0.9$ for $T=10$
  - $\text{Exp}(-1/1) = 1/e = 0.37$ for $T=1$

Local Beam Search

Local Beam Search Intuition

- Instead of following a single line through the state space, Local Beam Search follows multiple lines (a beam)



- In each step, we select the k best states from the set of all neighbors of all k current states

Local Beam Search

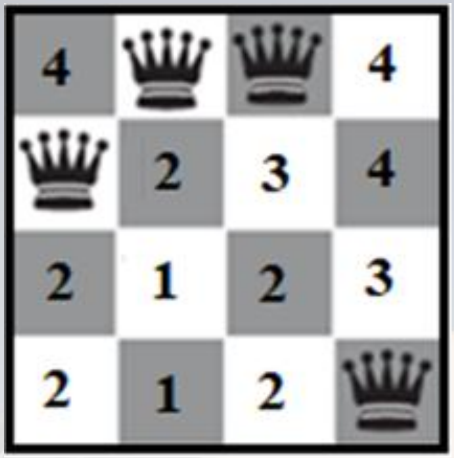
k_current ← select k random states
do
  neighbors ← all neighbors of all current states
  k_best_neighbors ← best k states from *neighbors*
  if no neighbor improves current value
    return *k_current*
  k_current← *k_best_neighbors*
until termination condition is met

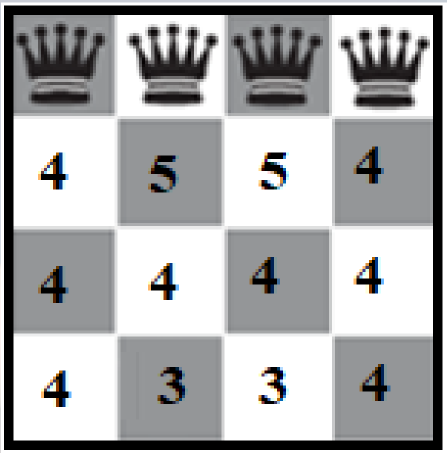
- Termination conditions can be chosen as before

Local Beam Search vs Parallel Hill-Climbing

- Local Beam Search is not equivalent to performing k Hill-Climbing searchs in parallel!
- In each step, Local Beam Search selects the k best neighbors of all current states
- For instance, if all best neighbors are neighbors of the first current state, we will only use neighbors of this state in the following step

Exercise: 4-Queens Problem

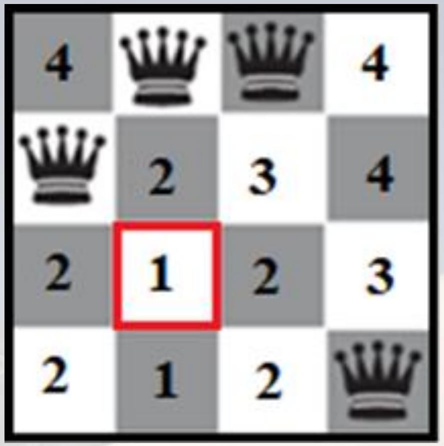




Consider the two states above with the given number of attacks for their neighbors.

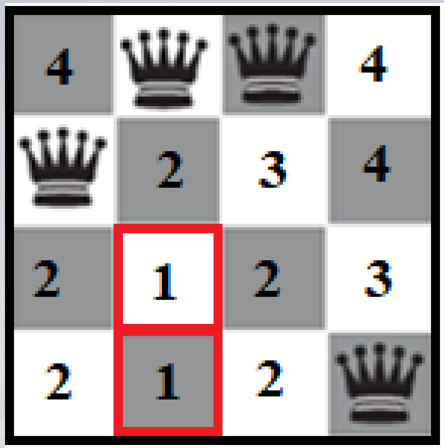
- Give two possible next states when performing parallel Hill-climbing (2 runs in parallel).
- Give two possible next states when performing Local Beam Search (k=2).

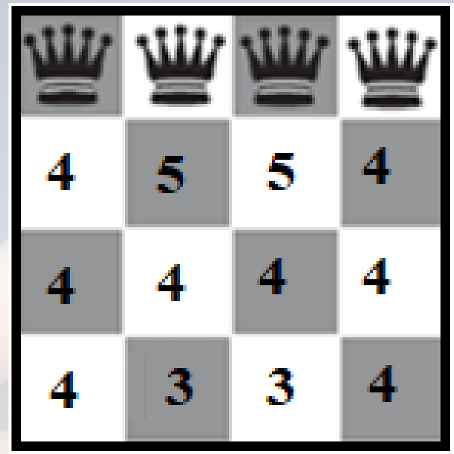
Solution: Hill-Climbing



- Each Hill-climbing search will select a best neighbor for each of the states
- We will continue with one state of value -1 and one state with value -3

Solution: Local Beam Search





- Local Beam Search will select the best two neighbors from the set of all neighbors of both states
- We will continue with two states of value -1

Stochastic Beam Search

- Local Beam Search can concentrate on a small region of the search space too early
- Stochastic Beam Search alleviates this problem by using randomized selection similar to Stochastic Hill-Climbing

```
k_current ← select k random states
do
  neighbors ← all neighbors of all current states
  k_best_neighbors ← k 'random' states from neighbors
  if no neighbor improves current value
     return k_current
  k_current← k_best_neighbors
until termination condition is met
```

Random Selection

```
k_current ← select k random states
do
   neighbors ← all neighbors of all current states
   k_best_neighbors ← k 'random' states from neighbors
   if no neighbor improves current value
       return best state from k_current
   k_current← k_best_neighbors
until termination condition is met
```

- Again, random selection is usually not completely random
- probability of selecting a state should increase with its value
- In this way, we balance diversity (exploration) and exploitation

Genetic Algorithms

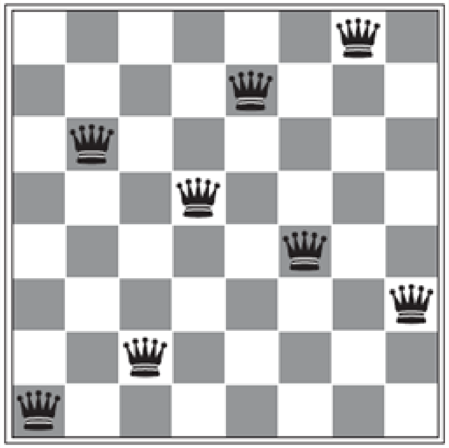
Genetic Algorithms Motivation

- Local Beam Search bears some resemblance to natural selection:
  - Successors (offspring) of states (organisms) populate next generation
  - Likelihood of survival depends on value (fitness)

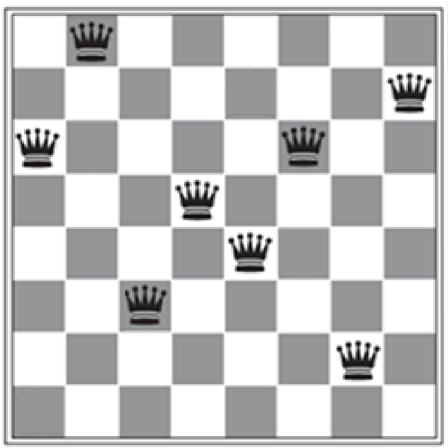
- Genetic Algorithms extend this analogy
  - In each step, two selected individuals reproduce
  - Selection probability increases with fitness (value)
  - There is a small probability of mutation
  - Offspring usually replaces other individuals (that die)

Population (States)

- Genetic Algorithms work on a population of individuals
- Each individual is a state represented as a chromose (e.g. string) over a set of genes (e.g. set of characters)
- For 8-Queens problem, we could use string consisting of 8 integers from {1,…8}, where i-th integer is row position of i-th queen



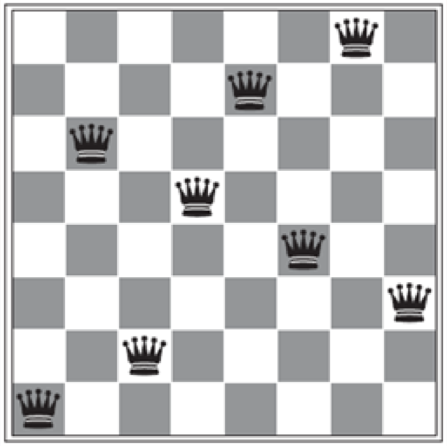
83742516



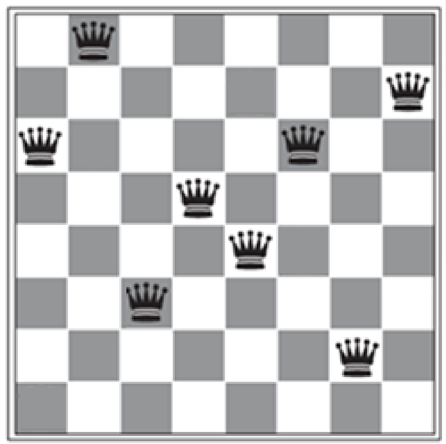
31645372

Fitness (Objective Function)

- Fitness corresponds to objective function
- For 8-Queens problem, we can reuse our old value function
- In order to get a non-negative fitness function, we could count the number of pairs of queens that cannot attack each other (28 – number of attacks between pairs)



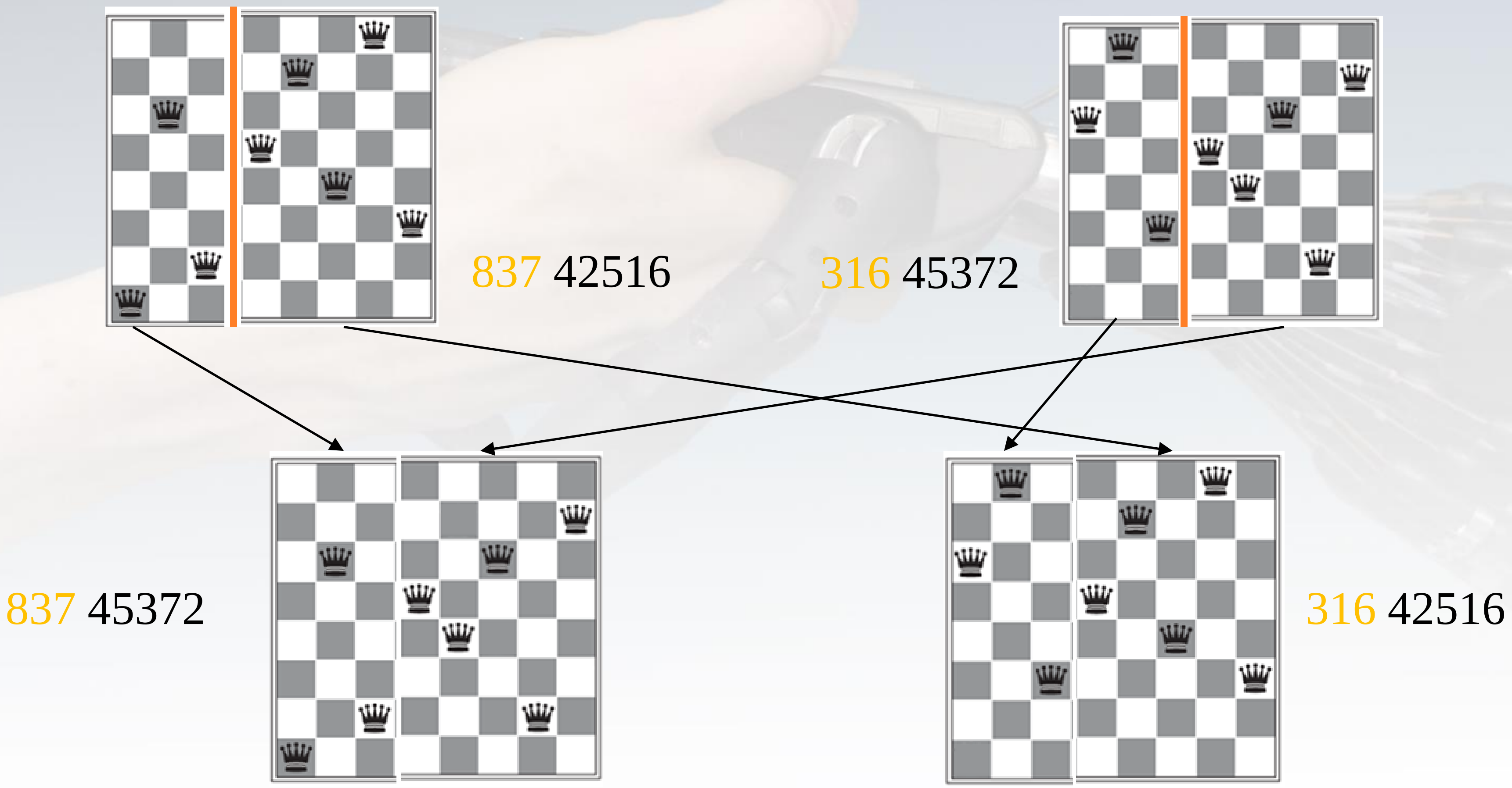
Fitness -1 or 27 (28–1)



Fitness -6 or 22 (28-6)

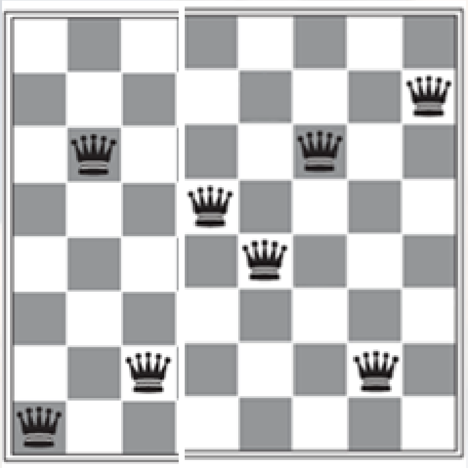
Reproduction (Recombination)

- Reproduction: we choose a random crossover point in chromosome
- Offspring is created by crossing parents at this point

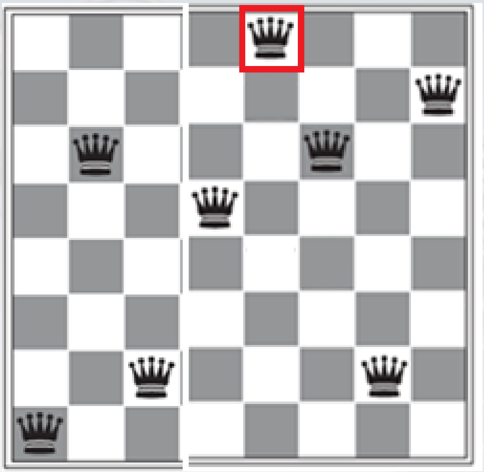


Mutation

- With small probability, mutation of offspring occurs
- E.g. replace one gene in chromosome randomly



837 45372

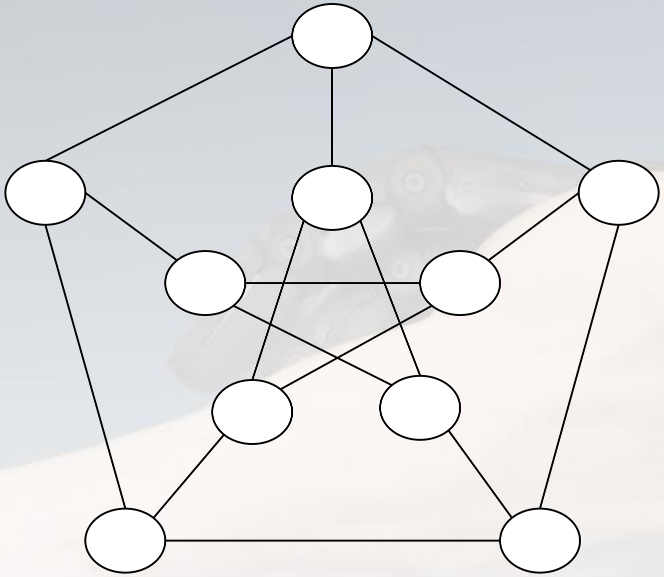


837 41372

A Simple Genetic Algorithm

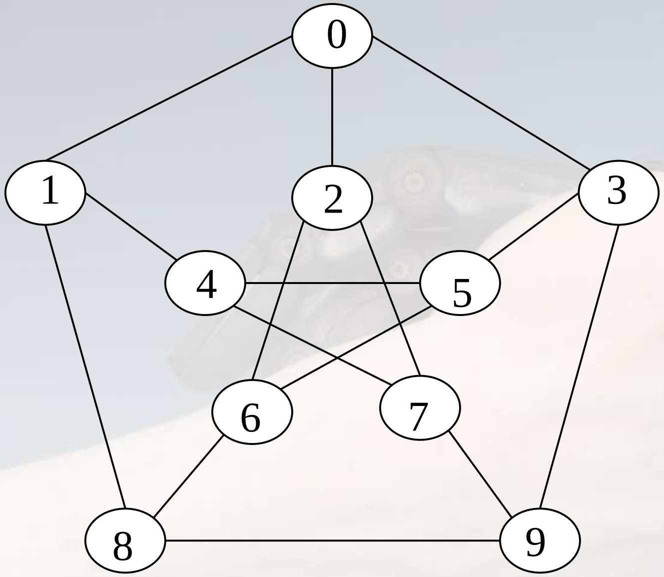
```
repeat
  population ← create k chromosomes at random
  for i = 1 to k do
    x ← select random chromosome based on fitness
    y ← select random chromosome based on fitness
    child ←reproduce(x , y)
    if (random() < 0.05)
      child ←mutate(child )
    add child to population
    remove random chromosome based on fitness
until some chromosome is fit enough, or time limit is reached
return the best chromosome in population
```

Excercise: Graph Coloring



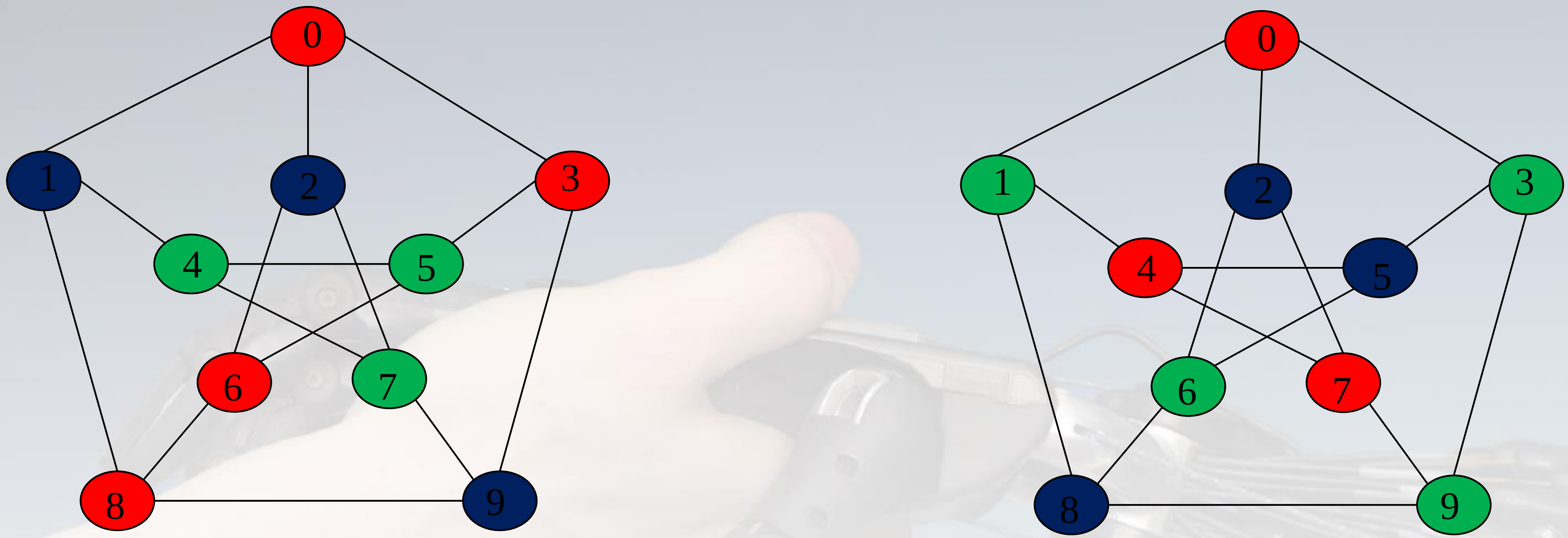
- Problem: assign a color from {1,2,3} to each node such that no two adjacent nodes have the same color
- Define
  - Genes
  - Chromosomes
  - Fitness (value) of individuals
  - Mutation operation
- Illustrate reproduction and mutation by means of a small example

Solution: Representation



- Genes {R,G,B}
- Chromosomes are strings of length 10, where i-th gene (letter) represents color of i-th node
- Fitness is the number of edges such that the corresponding nodes have different colors (0 <= Fitness <= 15)
- Mutation operation changes color of random node (50% chance for each of the other two colors)

Solution: Reproduction



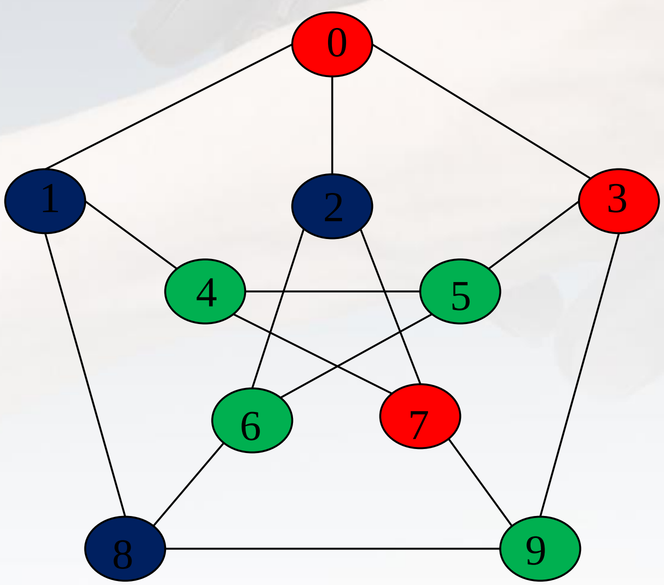
RBBRGGRGRB RGBGRBGRBG

RBBRGG GRBG RGBGRB RGRB

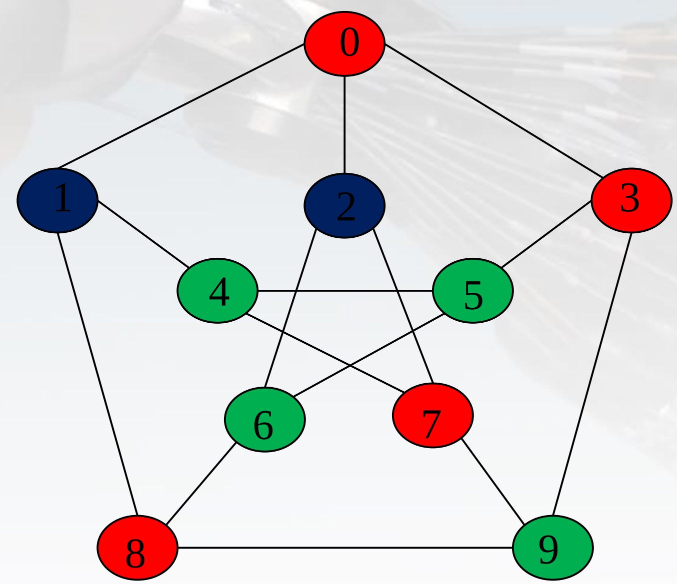
Solution: Mutation

- Mutation at 9-th gene (node 8)

RBBRGG GRBG



RBBRGG GRRG



Set Cover Problem

- Set Cover Problem: given set of n elements $U = \{e_1, \ldots, e_n\}$ and subsets $S_1, \ldots, S_m$ of U, find a minimal number of subsets that contain all elements from U (a minimal set cover of U)
- For example, U can be a shopping list and subsets correspond to shops that offer the desired items
  - $U = \{1, 2, 3, 4, 5\}$
  - $S_1 = \{1, 2, 5\}$, $S_2 = \{1,4\}$, $S_3 = \{3, 5\}$, $S_4 = \{1, 2\}$, $S_5 = \{3,4\}$
  - S_2, S_3 and S_4 contain all elements from U (S_2, S_3 and S_4 form a set cover of size 3)
  - S_1 and S_5 do also contain all elements (S_1, and S_5 form a set cover of size 2)
  - Hence, the set cover S_2, S_3 and S_4 is not minimal

Excercise: Set Cover Problem

- Set Cover Problem: given set of n elements $U = \{e_1, \ldots, e_n\}$ and m subsets $S_1,\ldots,S_m$ of U, find a minimal number of subsets that contain all elements from U
- Define
  - Genes
  - Chromosomes
  - Fitness (value) of individuals
  - Mutation operation
- Illustrate reproduction by means of a small example

Solution: Representation

- Genes {0, 1}
- Chromosomes are strings of length m, where i-th gene (letter) indicates that i-th set is used

  - U = {1, 2, 3, 4, 5, 6}
  - S_1 = {1, 2, 5}, S_2 = {1,4, 6}, S_3 = {2, 3, 5}, S_4 = {1, 2, 6}, S_5 = {3,4}
  - 10101: use S_1, S_3 and S_5
  - 01101: use S_2, S_3 and S_5

Solution: Fitness

- Fitness of an individual is
  - -1 if individual does not represent a set cover
  - m - size of set cover (number of 1s) otherwise

- U = {1, 2, 3, 4, 5, 6}
- S_1 = {1, 2, 5}, S_2 = {1,4, 6}, S_3 = {2, 3, 5}, S_4 = {1, 2, 6}, S_5 = {3,4}
- 10101 covers {1, 2, 3, 4, 5}: fitness -1 (does not cover 6)
- 01101 covers {1, 2, 3, 4, 5, 6}: fitness 5 - 3 = 2
- 01100 covers {1, 2, 3, 4, 5, 6}: fitness 5 – 2 = 3

Solution: Reproduction and Mutation

- Mutation operation flips random gene

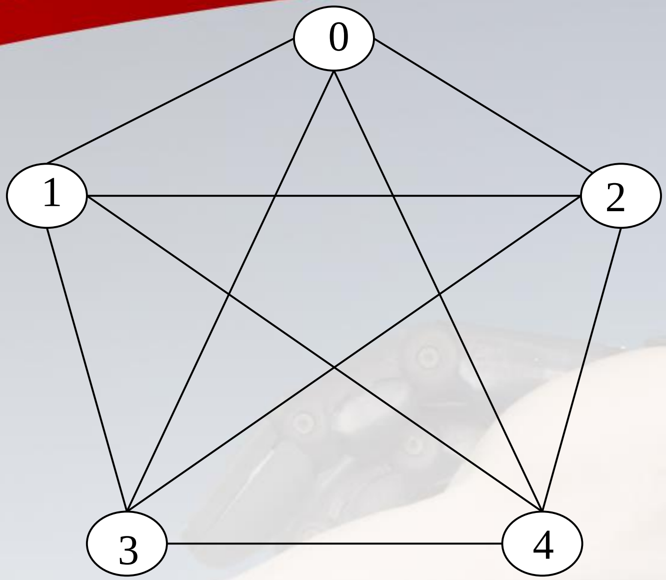
10110 00100

101 00 001 10

Mutation at first gene

101 10

Excercise: Traveling Salesman Problem



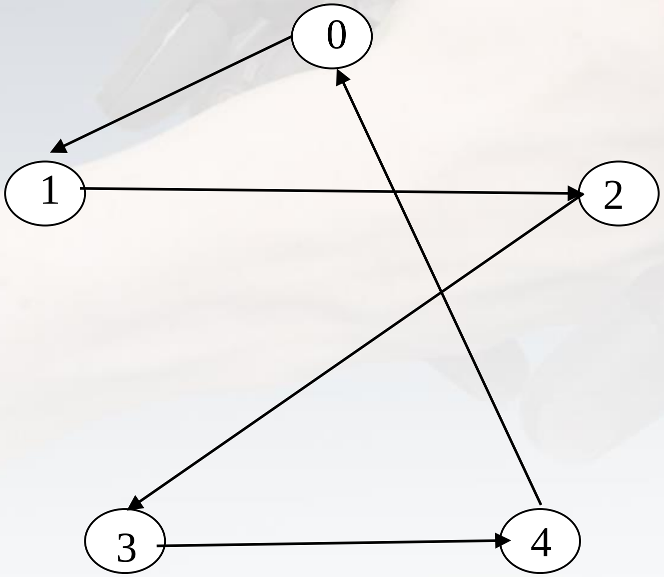
	0	1	2	3	4
0	0	5	6	5	2
1	3	0	6	2	1
2	4	3	0	5	1
3	5	5	3	0	7
4	6	6	4	7	0

(i-th column shows cost from node i to other nodes)

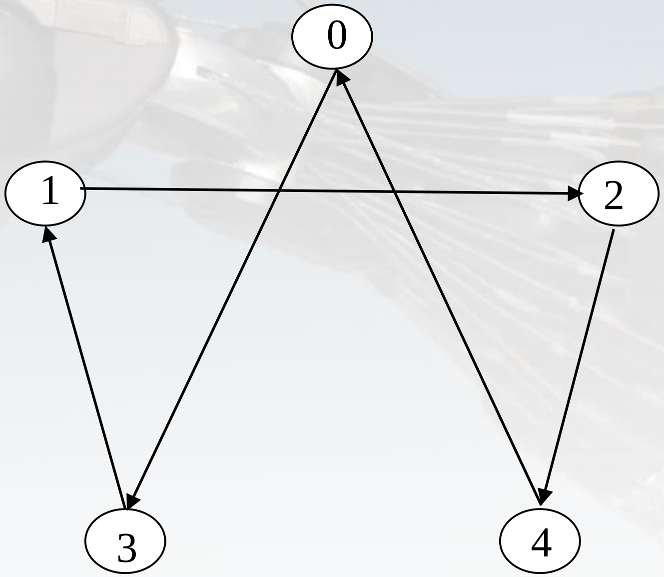
- TSP: find cyclic route that starts from node 0, visits each node exactly once and minimizes the overall edge cost
- Define
  - Genes
  - Chromosomes
  - Fitness (value) of individuals
  - Mutation operation
- Does crossover operation make sense?

Solution: Representation

- Genes {1, 2, 3, 4}
- Chromosomes are strings of length 4, where i-th gene (letter) represents the node that is visited at time i (first node is fixed to be 0)



1234

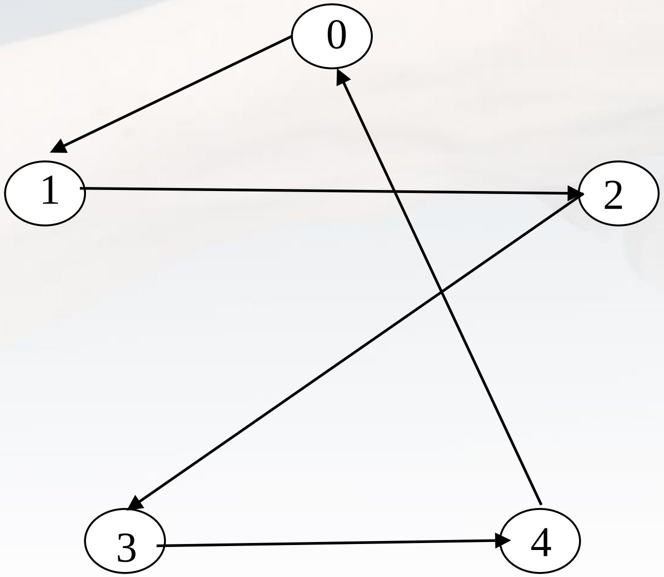


3124

Solution: Fitness

	0	1	2	3	4
0	0	5	6	5	2
1	3	0	6	2	1
2	4	3	0	5	1
3	5	5	3	0	7
4	6	6	4	7	0

- Fitness is 35 (rough upper bound on max. cost) minus the cost of getting from i-th to (i+1)-th node and from fithth node back to first node

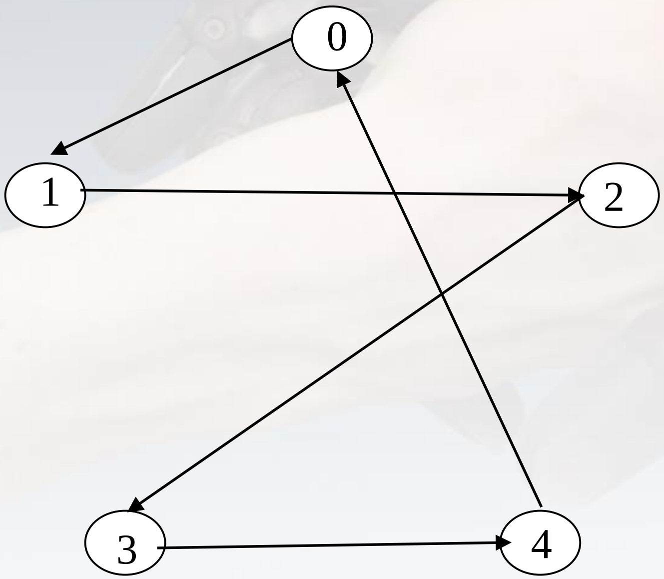


1234

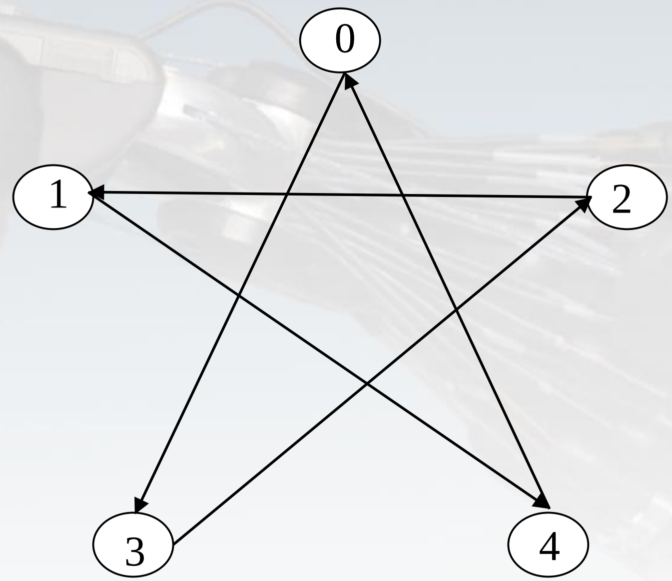
- Cost from 0 to 1: 3
- Cost from 1 to 2: 3
- Cost from 2 to 3: 3
- Cost from 3 to 4: 7
- <u>Cost from 4 to 0: 2</u>
- Overall cost: 18
- **Fitness**: 35-18 = 17

Solution: Mutation

- Mutation operation switches two genes at random
- E.g. switch first and third gene

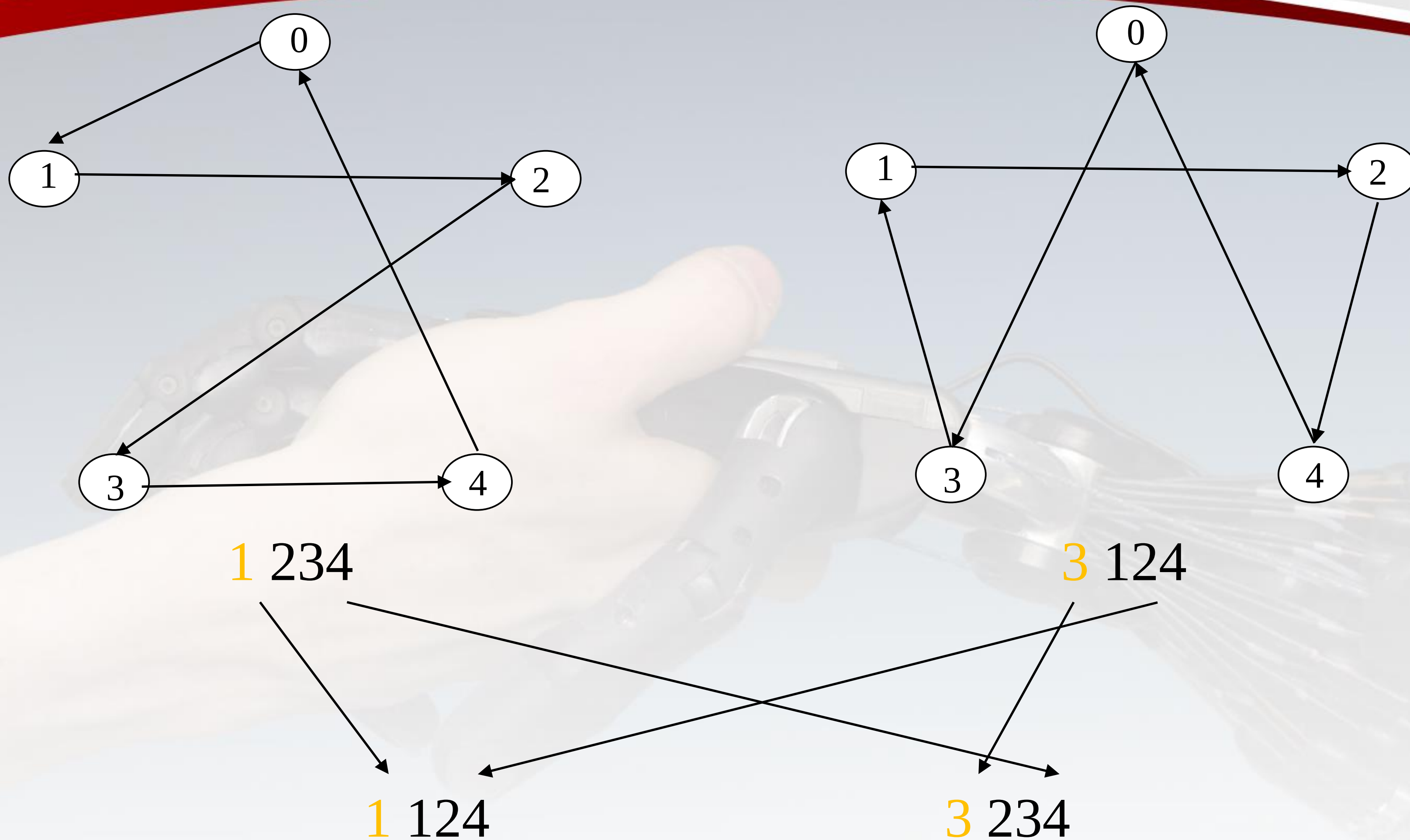


1234



3214

Solution: Problem with Reproduction



Naive reproduction yields invalid solutions

Reproduction of Permutations

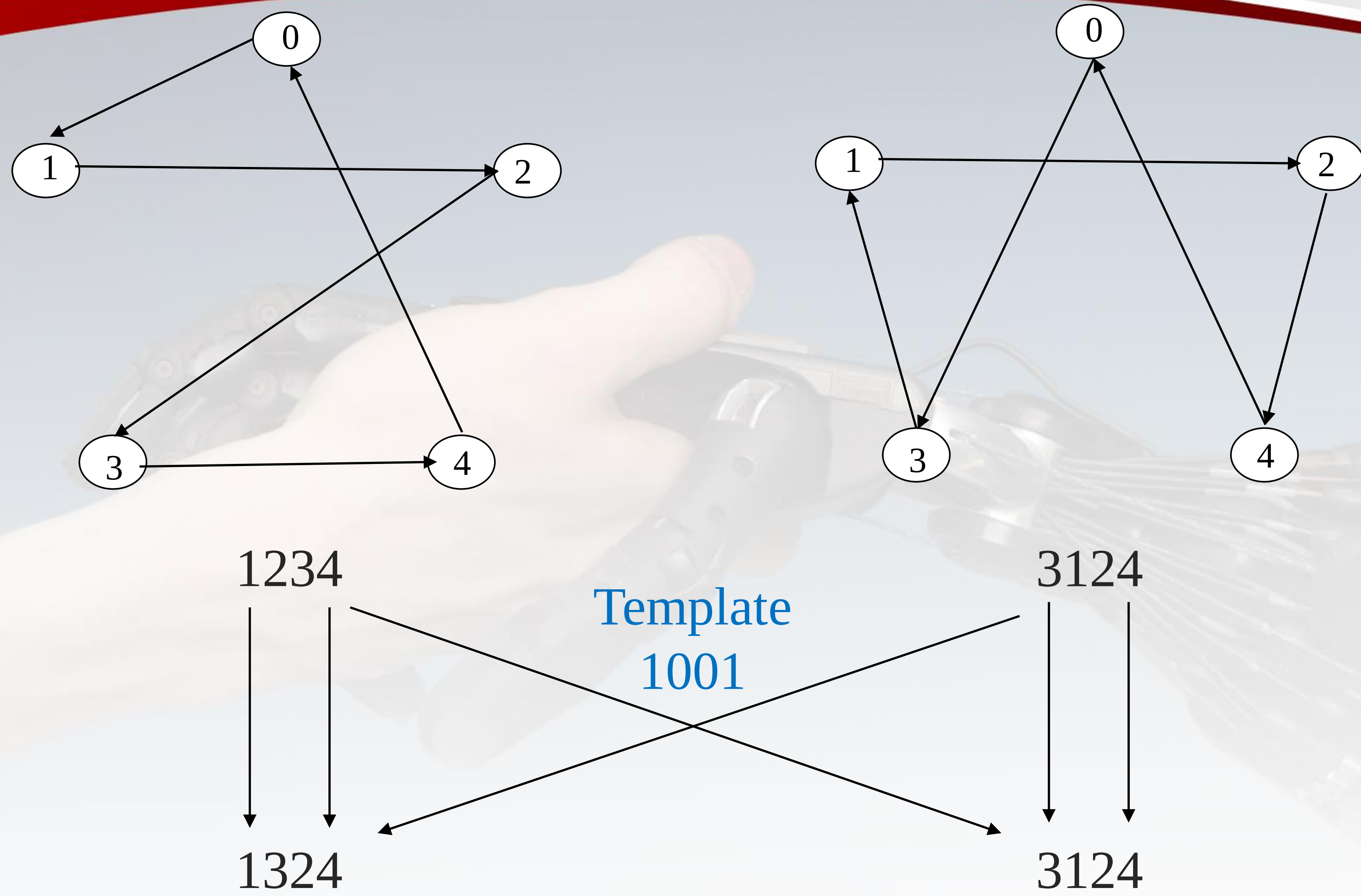
- If chromosomes correspond to permutations like in TSP, naive crossover reproduction can yield invalid solutions
- the problem can be fixed by
  - Changing the representation
  - Designing an additional repair operation that is applied after recombination
  - Using special crossover techniques for permutations

Uniform-Order Crossover

- Uniform-Order Crossover
  - Select two parents at random
  - Create random binary template
  - Child 1 is created by using genes of parent 1 at 1-positions. Fill gaps with the remaining elements according to the order given by parent 2
  - Child 2 is created by switching the roles of parent 1 and parent 2

A	B	C	D	E	F	G	Parent P_1
E	B	D	C	F	G	A	Parent P_2
0	1	1	0	0	1	0	Template
E	B	C	D	G	F	A	Child C_1
A	B	D	C	E	G	F	Child C_2

Solution: Problem with Reproduction



Variants

- Genetic Algorithms can be configured by different
  - Selection operators
  - Reproduction operators
  - Mutation operators
- Hybrid Genetic Algorithms combine genetic algorithms with other techniques
  - for instance, Memetic Algorithms apply a (fast) local search algorithm to each newly created individual to make it locally optimal
  - Hill-Climbing is well suited for this purpose because it is fast

Summary: Local Search

Summary

- If we are only interested in a solution and not in the solution path, Local Search can be better suited than Classical Search
- this is typically the case for optimization and constraint satisfaction problems
- Some important local search algorithms are
  - Hill-Climbing
  - Simulated Annealing
  - Local Beam Search
  - Genetic Algorithms

Further Readings

Lecture is mainly based on:

Russell, S., Norvig, P. Artificial Intelligence - A modern approach. Pearson Education: 2010.

More details on presented (and similar algorithms) can be found in:

Burke, E. K., & Kendall, G. Search methodologies - Introductory Tutorials in Optimization and Decision Support Techniques. Springer US: 2014.