CSC 480: Artificial Intelligence

4- Local Search

Rodrigo Canaan
Assistant Professor
Computer Science Department
Cal Poly, San Luis Obispo
rcanaan@calpoly.edu

Adapted with permission from class notes by Prof. Julian Togelius (NYU)

Search so far

- We have focused so far in problems where **the path to a desired state is the solution**
    - Maze navigation (actual path)
    - Sequence of tiles we slide in 8-puzzle
    - Sequence of moves in chess, go
- Usually, we were interested in finding the shortest path, or the path that leads to a high-value terminal node.
- Can you think of problems where the solution does not rely on the order of operations? Instead, the solution **is the desired state itself?**

Example: n-queens

- Place n queens on an n x n board
- Each queen attacks other queens in the same row, column or diagonal
- Find a placement with no attacks

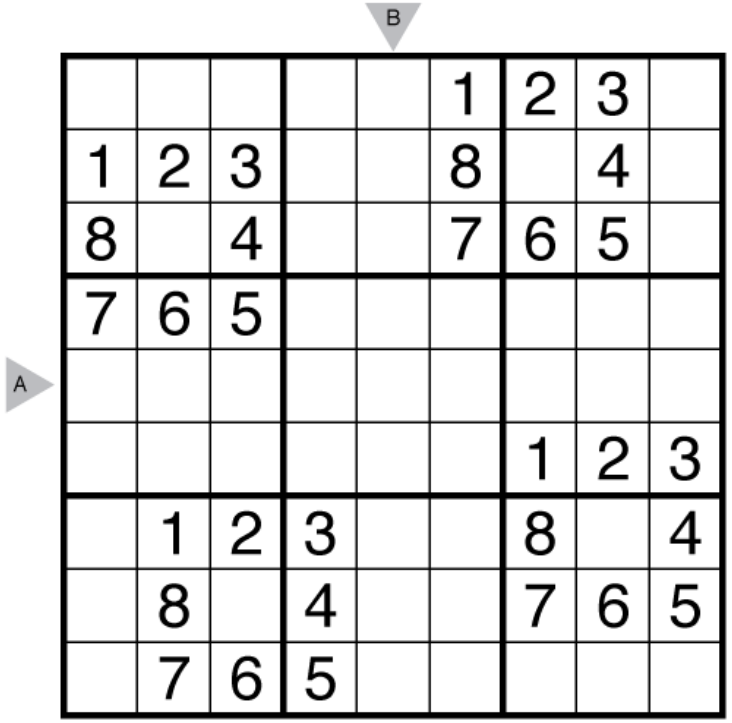
Example: Cryptarithmetic

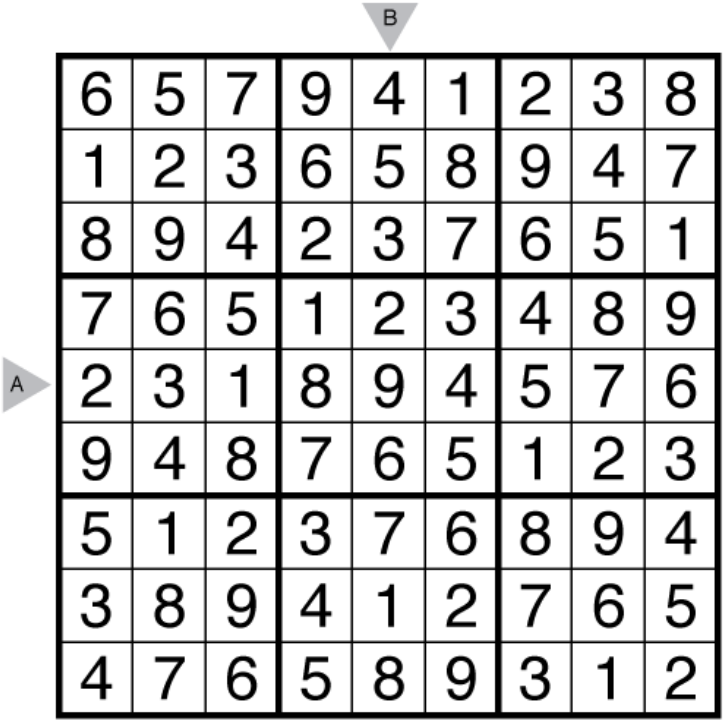
- Assign value to each letter to satisfy a mathematical expression (interpreting the letters as digits)

$$\begin{array}{cccccc} & & \mathrm{S} & \mathrm{E} & \mathrm{N} & \mathrm{D} \\ + & & \mathrm{M} & \mathrm{O} & \mathrm{R} & \mathrm{E} \\ \hline = & \mathrm{M} & \mathrm{O} & \mathrm{N} & \mathrm{E} & \mathrm{Y} \end{array}$$

Example: Sudoku

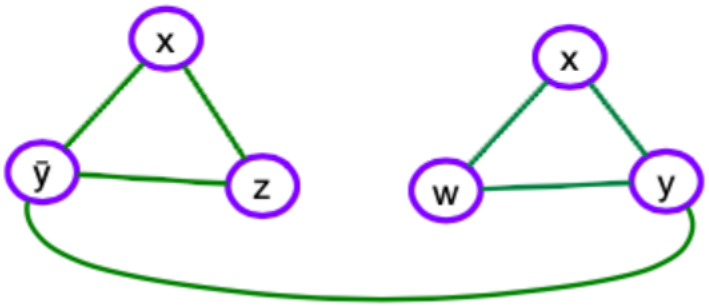
- Every cell has a number 1-9
- No line, row or 3x3 square has same number





Example: Boolean Satisfiability (SAT)

- Input: a boolean expression such as $(x \vee \neg y \vee z) \wedge (x \vee y \vee w)$
- Assign each of the variables (x, y, z, w) a value in {TRUE, FALSE} that logically satisfies the expression
- Can be reduced to a Graph problem of Independent sets (find 2 vertices below that are not neighbors)



Constraint Satisfaction Problems (CSPs)

- Each of these problems can be seen as a problem of **assigning values to variables** while **respecting certain constraints**

Problem	Assignment	Constraints
n-Queens	Queens are placed on n^2 squares	• No queens in the same row • No queens in the same column • No queen in the same diagonal
Cryptarithmetic	Each letter is assigned a number 0-9	• Digits in the operands add up to digits in the result (with carry) • (optionally) No two letters have same digit
Sudoku	Each cell is assigned a number 1-9	• No repeat numbers in the same row • No repeat numbers in the same column • No repeat numbers in the same 3x3 square
Satisfiability	Each variable is assigned a value {TRUE,FALSE}	• a OR b is satisfied if either a or b is TRUE • a AND b is satisfied if both a and b TRUE • etc (see chapter on Logic later) • Whole sentence must be satisfied

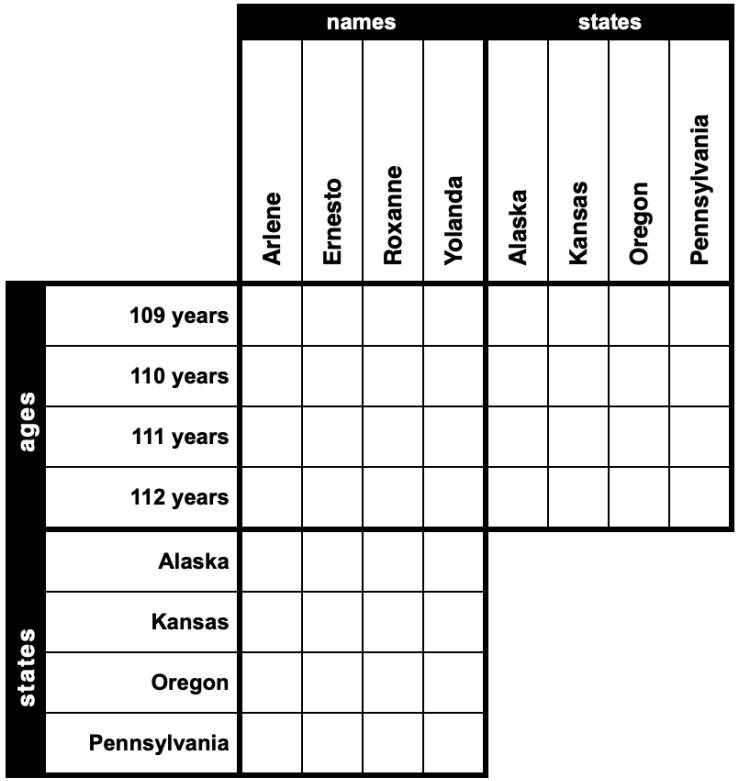
CSP Formulation

A CSP is defined by:

- A set of variables $X = X_1, X_2, \ldots, X_n$
- A set of domains $D = D_1, D_2, \ldots, D_n$
    - One domain per variable
    - Each domain is a set of values e.g. $D_1 = v_1^1, v_2^1, \ldots, v_k^1$
- A set of constraints C that specify which variable combinations are valid

CSP Formulation Exercise

- How would you formulate the puzzle below? List:
  - The variables
  - The domain of each variable
  - The constraints implied by the board (even with no clues!)

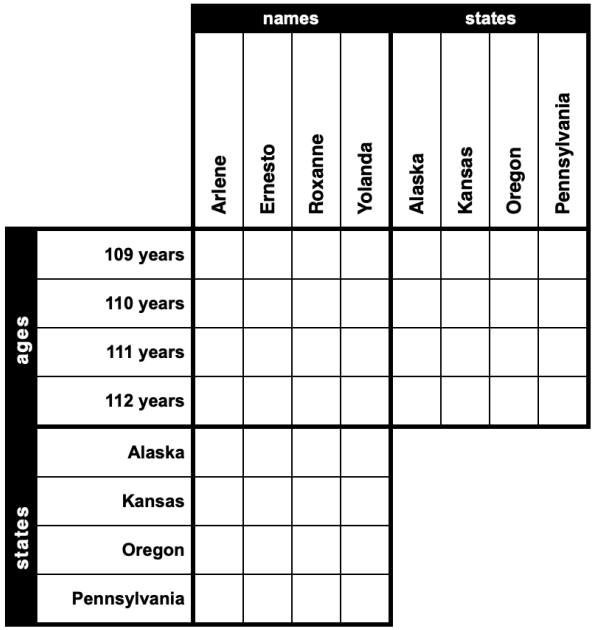


CSP Definitions

- An **assignment** (or candidate) a mapping of each variable to one of the values in its domain
- An assignment is **complete** if every variable has a value assigned to it
- An assignment is **consistent** if no constraints are violated
- The goal is to find a **complete and consistent** assignment

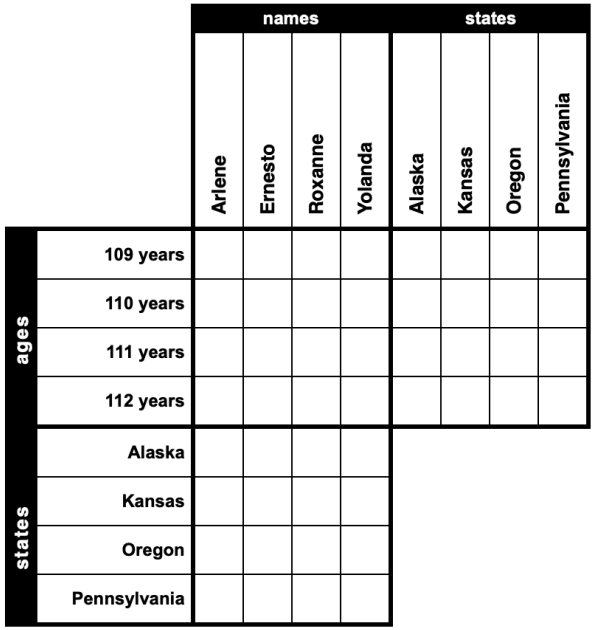
CSP Constraints

- Constraints can be:
  - unary e.g. “Ernesto does not live in Kansas”
  - binary e.g “The person who lives in Alaska is not 111 years old”
  - ternary or more (n-ary) e.g “the sum of Arelene, Ernesto and Yolanda’s age is 330”
  - global e.g. “Everybody has a different age” (note: global involves an arbitrary number of values, but not necessarily all variables



CSP Constraints

- In practice, we will typically consider only binary constraints
- Unary ones can be treated as a domain restriction
- Ternary and higher order constraints can always be reduced to binary constraints (with enough helper variables)



Optimization Problems

Optimization problems can be seen as an extension of CSPs:

- A set of variables $X = X_1, X_2, \ldots, X_n$
- A set of domains $D = D_1, D_2, \ldots, D_n$
    - One domain per variable
    - Each domain is a set of values e.g. $D_1 = v_1^1, v_2^1, \ldots, v_k^1$
- A set of constraints C that specify which variable combinations are valid
- A “cost” or “fitness” function $F(X_1, X_2, \ldots, X_n)$ that needs to be minimized or maximized, respectively.

Example of Optimization Problems

- **Geometric Set Cover:** Given a set of cities and communication towers with a certain range, place towers such as:
  - to cover all cities (constraint)
  - while minimizing the number of towers (cost)

Example of Optimization Problems

- **Knapsack:** Given a set of items (v_i, w_i) and a capacity W select a subset S of items such that:
  - combined weight less than capacity: $\sum_{j \in S} w_j \leq W$
  - total value is maximum: $\sum_{j \in S} v_j$

Optimization Problems

- We call an assignment **feasible** if it satisfies all constraints
- We call an assignment **optimal** if it has the lowest cost / highest fitness of any feasible assignment
- You can usually turn a CSP into an optimization problem by assigning a cost to each violated constraint.

Solving CSPs and Optimization Problems

Two main approaches:

- **Local search - No backtracking**
  - Define a “cost” for candidate solution
  - Maintain a small number of promising candidates
  - No backtracking
  - Modify them randomly until you find a (complete, consistent, high fitness) solution

Solving CSPs and Optimization Problems

Two main approaches:

- **Constraint satisfaction**
  - Use constrains to directly generate feasible assignments.
  - Often combined with backtracking search
  - Can be done naively (e.g. DFS) but very inefficient
  - More sophisticated algorithms avoid expanding nodes that won't lead to solutions by taking constraints into account ahead of time

How to solve CSPs?

We will focus first on **Local search**

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

Local Search

Local Search Overview

- Local Search algorithms follow the following template
    - Maintain one (or a small number) of candidate solutions
        - Often initialized randomly
    - Create a heuristic (or loss) that tells how good (or bad) a candidate is
        - Example: number of constraints violated
    - Define a number of operations available to modify a solution
        - Example: assign a new value to a single variable
    - Randomly apply operations until you find a consistent & complete (or good enough) solution
- Search is “local” in the sense that only immediate neighbors of the current candidates are considered. No backtracking.

Hill Climbing

The simplest Local search algorithm:

- Initialize a single candidate randomly
- Neighbors = the set of candidates reached by applying any operation to the current candidate
- Evaluate each neighbor with the heuristic
- Repeat using the best neighbor

Analogy: climbing a hill going in the direction of steepest slope *

* Note that this is typically done without explicitly computing the gradient (derivative) of the heuristic function. If compute it, this becomes gradient descent/ascent.

Hill-climbing search

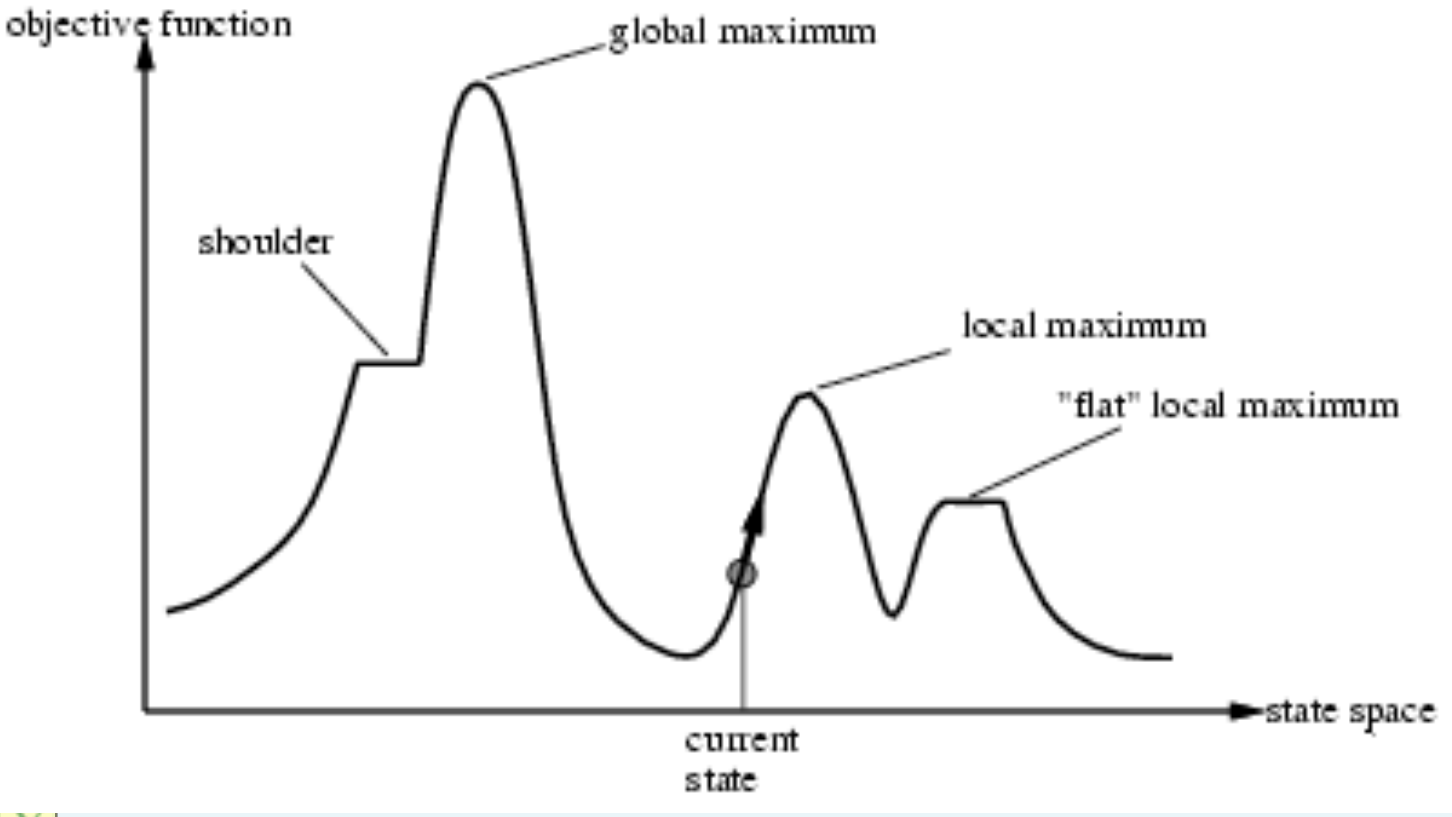
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```


http://aima.eecs.berkeley.edu/slides-ppt/

Hill-climbing search

- **Problem: depending on initial state, can get stuck in local maxima**



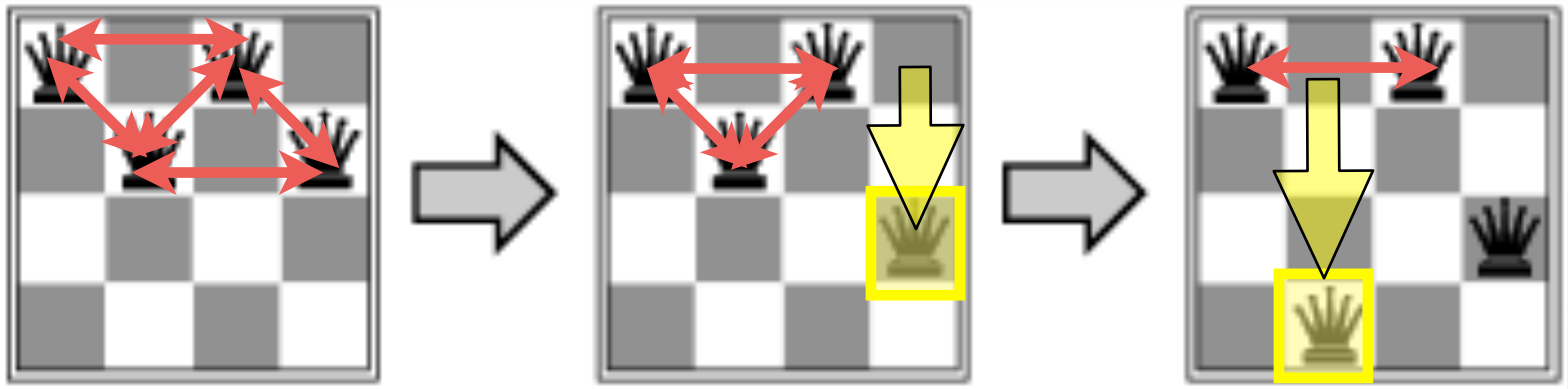
http://aima.eecs.berkeley.edu/slides-ppt/

Example: *n*-queens

- **Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal**
- **Assumptions**
  - all queens are on the board
  - each queen in a different column
  - queens move only vertically

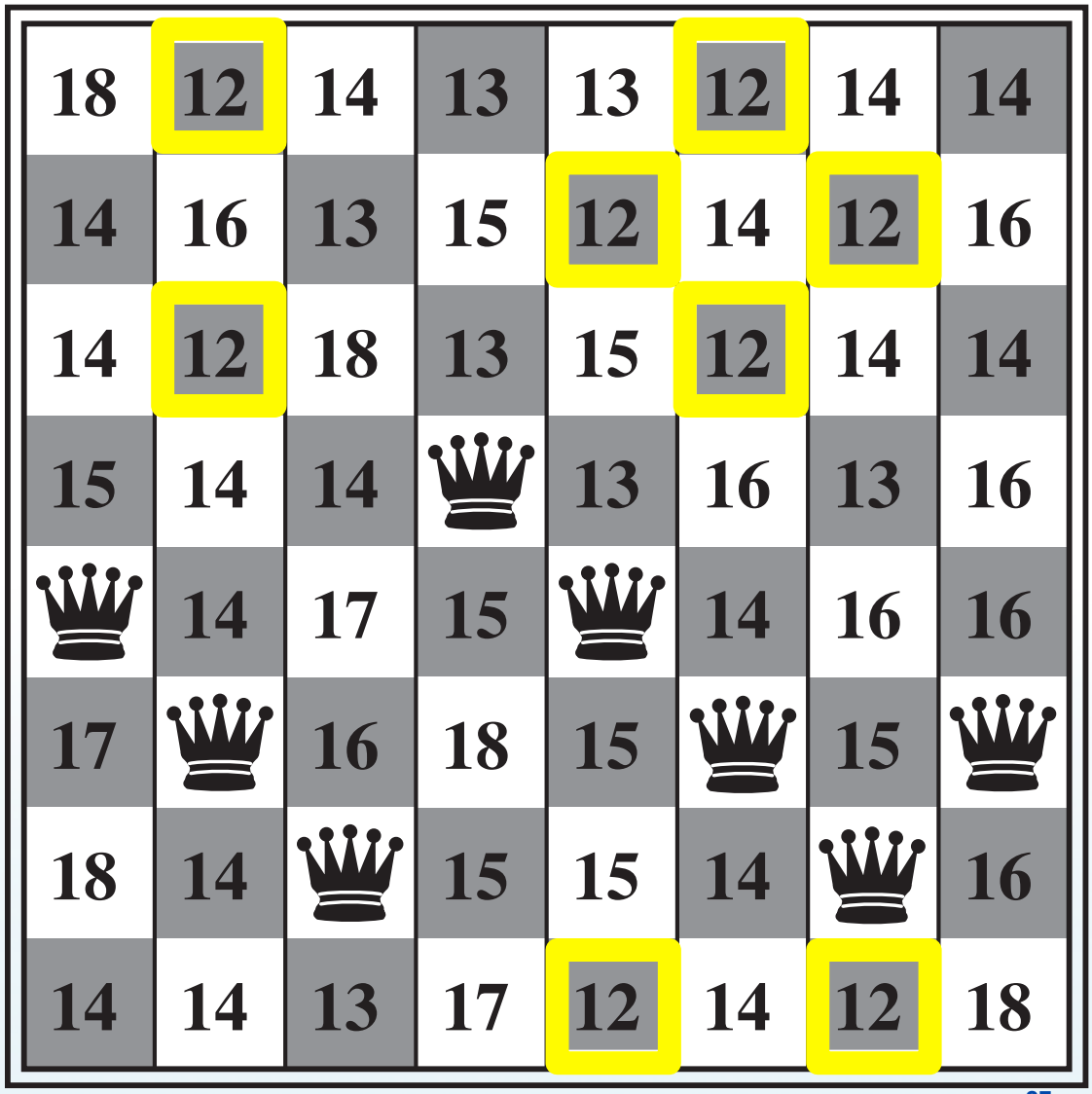
Hill-climbing: 4-queens

- 4 queens on 4x4 board
- optimization criterion: number of attacked pairs of queens
  - illustrated through red lines
- Operations: move a single queen to a new row.



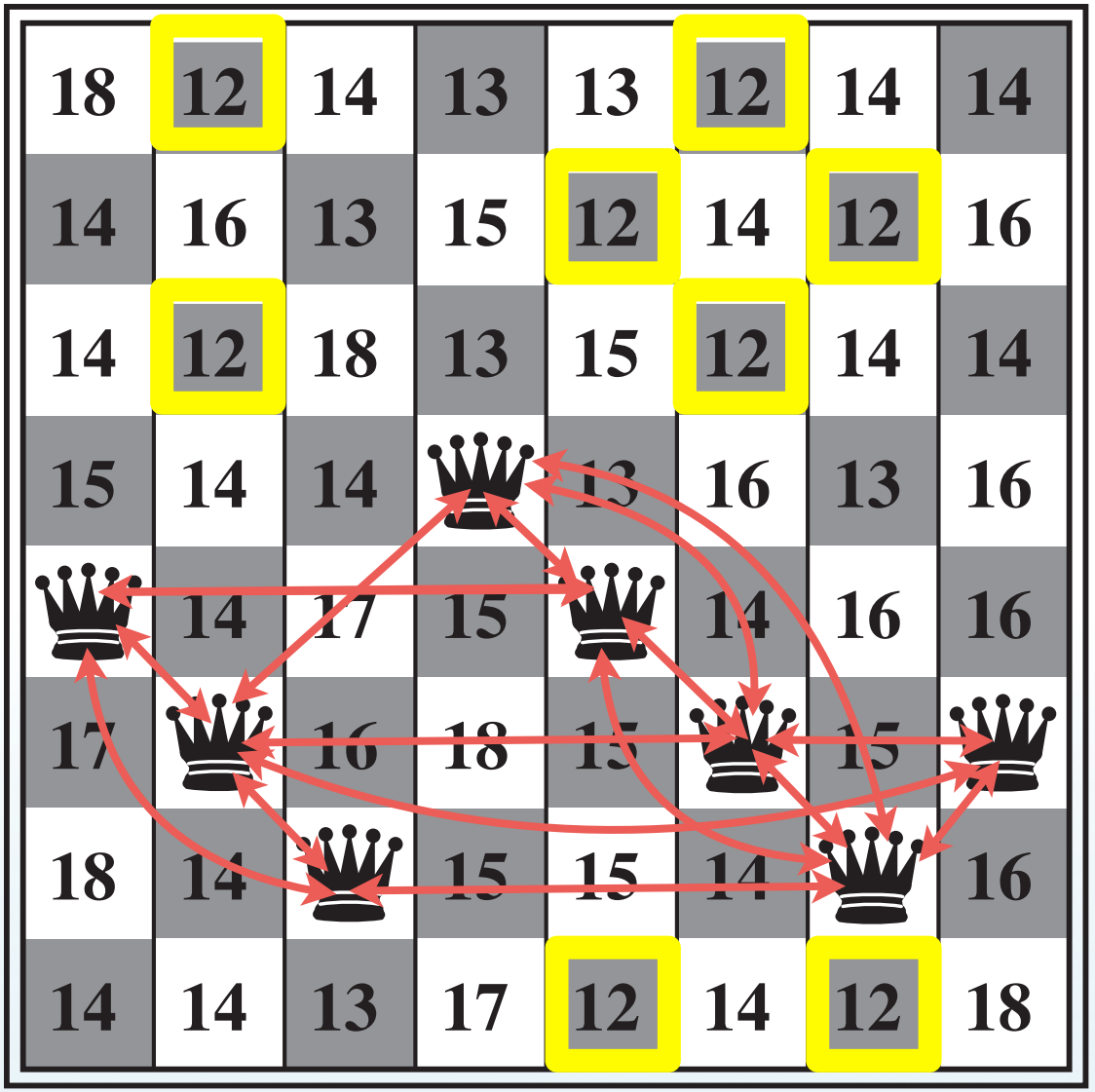
Hill-climbing search: 8-queens

- h = number of pairs of queens that are attacking each other
- h = 17 for the above state
- best moves are marked (h=12)

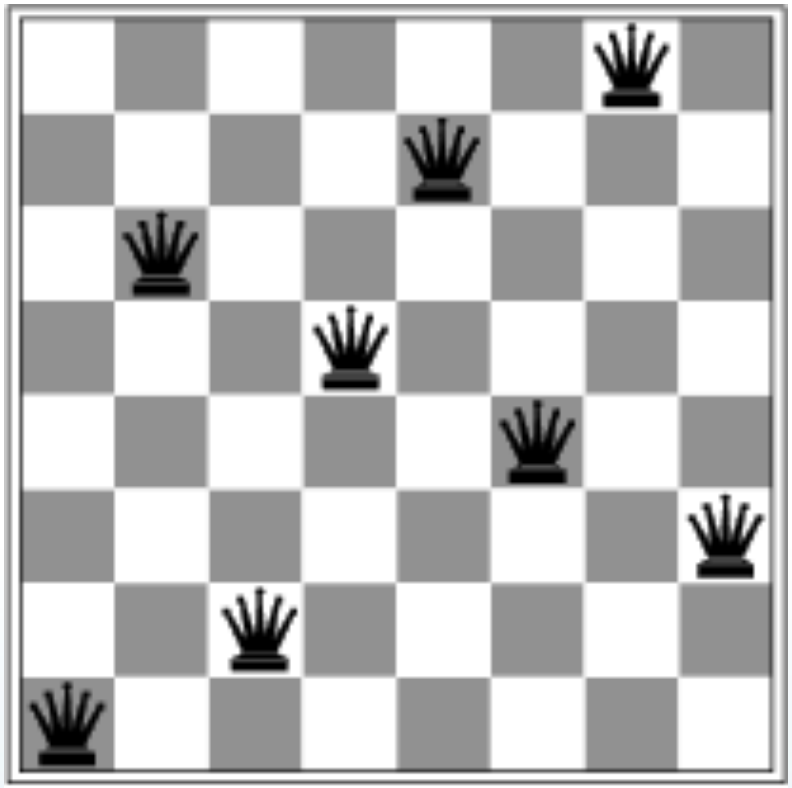


8-queens: Attacked Pairs

- h = number of pairs of queens that are attacking each other
- h = 17 for the above state
- best moves are marked (h=12)

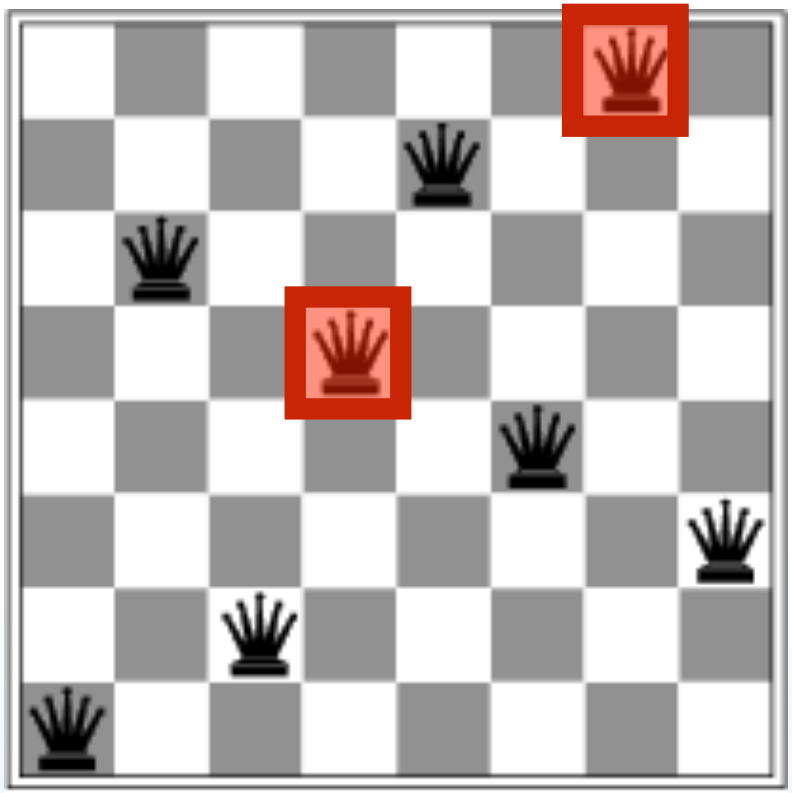


Hill-climbing search: 8-queens problem



- A local minimum with h = 1

Hill-climbing search: 8-queens problem



- **A local minimum with h = 1**
  - one pair of queens attacking each other

Local Search in Continuous Domains

- Bad news: In continuous domains, we cannot enumerate all "neighbors" of a state
- Good news: we can model the cost or fitness as a differentiable function, its **gradient** points in the direction of fastest increase at each point
- **Gradient Descent/Ascent** is thus analogous to hill climbing: at each step, move your variables a small distance in the direction of the gradient
- **Stochastic Gradient Descent (SGD):** We can often approximate the gradient by sampling some small perturbations of the current state
-

Local Search in Continuous Domains

- Gradient descent plays a major role in modern machine learning
- For now, we will assume our domain is discrete and review this topic later
- But note that many of the following algorithms can be adapted to use gradients if available
- Conversely, some discrete problems with certain regularity in its fitness landscape can borrow techniques from SGD

Simulated Annealing

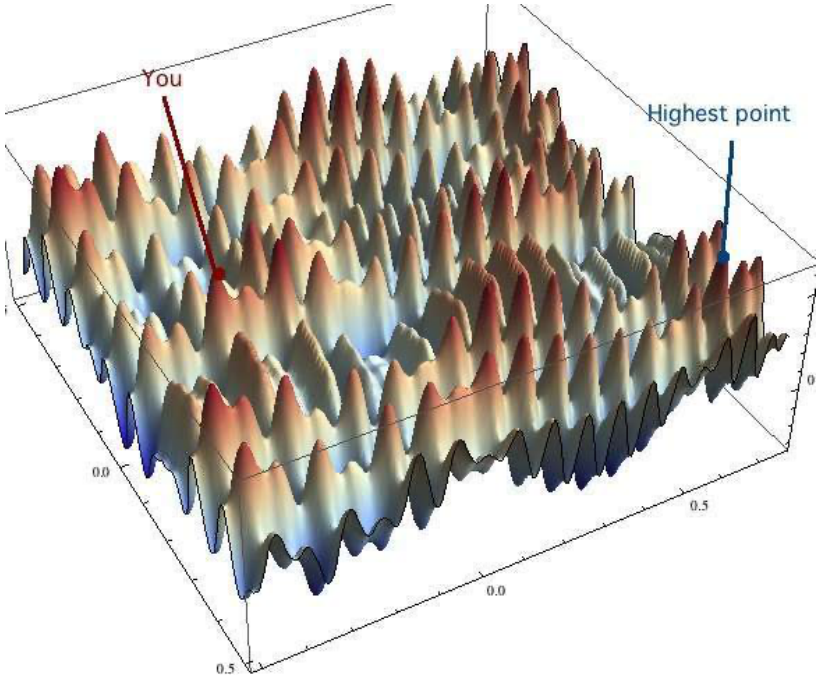
Hill Climbing doesn't allow "downward" movement

- The problem is you sometimes need to descend a bit before finding an even higher hill
- Solution: allow random movement with some probability
- Probability depends on:
  - **Difference in heuristic** ("energy") of two states
    - Higher difference -> lower probability
  - The current **temperature**
    - This is a parameter the algorithm keeps track of based on a **schedule**
    - Higher temperature -> higher probability
    - Schedule starts temperature high, "cools down" over time

Simulated Annealing

Original inspiration: annealing (wiki), a heat treatment process of materials that starts by heating it , then slowing it to gradually cool

- A more intuitive analogy: shake the fitness landscape until you manage to place a ball in the deepest hole



Simulated Annealing Search

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(Δ E/T)
```

Simulated Annealing Properties

- The probability of finding a global optimum approaches 1 when the schedule is extended indefinitely
- In practice, may not always find the global optimum, but usually finds good approximate solutions
- Less vulnerable to local optima than Hill Climbing

Local Beam Search

- Maintain a small number (k) of candidates
- Generate all neighbors of each current candidate
- Rank all neighbors, keep the best k
- Repeat until a solution is found or out of budget

Local Beam Search Properties

- Analogous to Breadth-First Search, but you only visit the top k children of the nodes at each level
    - If k is infinite, reduces to BFS
- Candidates can “exchange information”:
    - If a candidate has lots of good neighbors, it will force the search to abandon previous candidates altogether
- Can suffer from lack of diversity: all candidates become very similar
    - If this happens, explores a very small corner of the search space
    - This is similar to a very slow form of Hill Climbing
- Alternative: stochastic beam search (pick neighbors with some probability depending on their heuristic)

Evolutionary Algorithms

- A family of algorithms inspired by biological evolution
- Similar to beam search in that you keep a (usually) fixed-size "population" of candidates
- In addition, you can "combine" two or more candidates to make "offspring" via "reproduction"
    - Typical operators: mutation and crossover
        - Mutation random changes to a single candidate before reproduction
        - Crossover makes a new candidate by combining parts of two or more "parents"

Evolutionary Algorithms

- A family of algorithms inspired by biological evolution. Heuristic = “fitness”
- Similar to beam search in that you keep a (usually) fixed-size “population” of candidates
- In addition, you can “combine” two or more candidates to make “offspring” via “reproduction”
    - Typical operators: mutation and crossover
        - Mutation random changes to a single candidate before reproduction
        - Crossover makes a new candidate by combining parts of two or more “parents”
- After that, you make a new population with the current best k individuals

Evolutionary Algorithms Outline

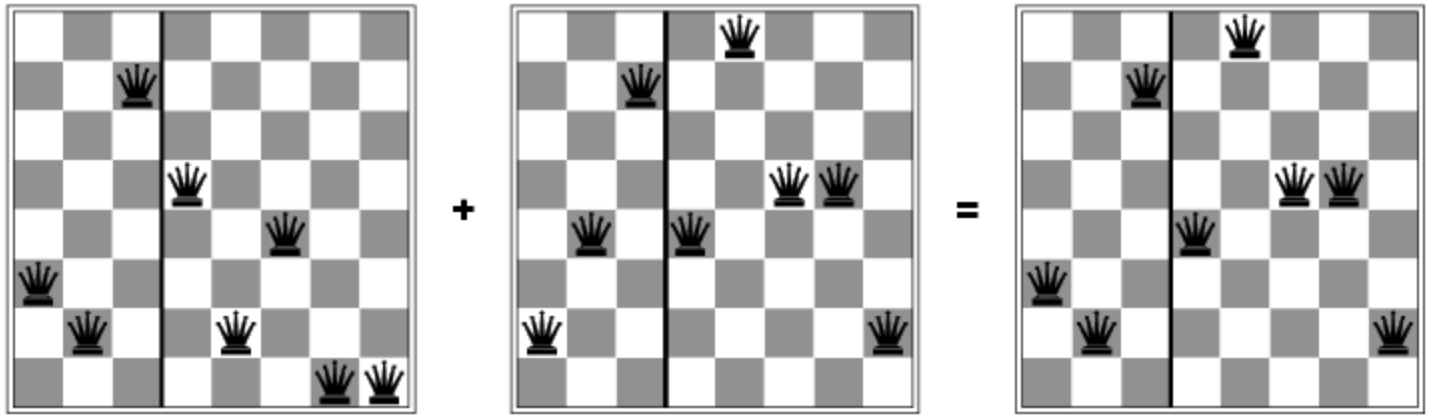
- Start a population of k individuals (generated randomly)
- While within budget:
  - Select parents to produce offspring (higher fitness = higher probability to be selected)
  - Generate one or more children via mutation and/or crossover per parent
  - Select k individuals (among parents and/or children) to make up the next population (again, higher fitness = higher probability)

Example: Genetic algorithm for N-Queens

- **Fitness function: number of non-attacking pairs of queens**
  - (min = 0, max = (8*7)/2 = 28)
- **probabilities**
  - 24/(24+23+20+11) = 31%
  - 23/(24+23+20+11) = 29% etc

http://aima.eecs.berkeley.edu/slides-ppt/

Genetic Algorithm for N-Queens - Crossover



http://aima.eecs.berkeley.edu/slides-ppt/

Evolutionary Algorithms - Variations

- Evolutionary algorithms can vary:
  - Based on the type of representation (strings, grids, trees, graphs (and weights))
  - Based on the size of population
  - Based on how mutation and crossover work
    - Also, mutation/crossover rate
  - Based on how individuals are selected for reproduction and/or for remaining in the population
    - Elitism is a common choice: the top few individuals always remain in the population, the rest are selected with some probability

Evolutionary Algorithms - Other notes

- Evolutionary Algorithms are a family of algorithm
    - Specific examples: Genetic Algorithm, Genetic Strategies, Genetic Programming…
    - Distinctions are based on the type of representation and operators used (details out of scope for this course)
- The main difference between EAs and local beam search is the existence of crossover (recombination)
    - Tends to work better if representation has “blocks” that perform related functions.

My phd Research: Hanabi

The goal: build piles of colored cards by playing them in order

The catch:
Players do not see their own hands!

The catch:
Players do not see their own hands!
Players cannot communicate freely

Historical Context

Early 2018

Historical Context
Early 2018

- Hanabi Competition at CIG organized by Walton-Rivers, *et al.*

Historical Context
Early 2018

- Hanabi Competition at CIG organized by Walton-Rivers, *et al.*
    - Mirror track (self-play)

Historical Context
Early 2018

- Hanabi Competition at CIG organized by Walton-Rivers, *et al.*
    - Mirror track (self-play)
    - Mixed track (ad-hoc teamplay, but no adaptation between games)

Historical Context
Early 2018

- Hanabi Competition at CIG organized by Walton-Rivers, *et al.*
    - Mirror track (self-play)
    - Mixed track (ad-hoc teamplay, but no adaptation between games)
        - Evaluation pool was not known by participants

Historical Context
Early 2018

- Hanabi Competition at CIG organized by Walton-Rivers, *et al.*
    - Mirror track (self-play)
    - Mixed track (ad-hoc teamplay, but no adaptation between games)
        - Evaluation pool was not known by participants
- Competition framework provided rules (heuristics) for implementing various agents from previously-published papers.

Historical Context
Early 2018

- Hanabi Competition at CIG organized by Walton-Rivers, *et al.*
    - Mirror track (self-play)
    - Mixed track (ad-hoc teamplay, but no adaptation between games)
        - Evaluation pool was not known by participants
- Competition framework provided rules (heuristics) for implementing various agents from previously-published papers.
- Best rule-based agent in the framework: *Piers* (Walton-Rivers, *et al. 2017*)

Historical Context
Early 2018

- Hanabi Competition at CIG organized by Walton-Rivers, *et al.*
    - Mirror track (self-play)
    - Mixed track (ad-hoc teamplay, but no adaptation between games)
        - Evaluation pool was not known by participants
- Competition framework provided rules (heuristics) for implementing various agents from previously-published papers.
- Best rule-based agent in the framework: *Piers* (Walton-Rivers, *et al. 2017*)
    - ~17 points in 2-player self-play Hanabi

Rule-based agents

Piers (Walton-Rivers *et al*., 2017)

Rule-based agents

Piers (Walton-Rivers *et al*., 2017)

1. “Hail Mary”

Rule-based agents

Piers (Walton-Rivers *et al*., 2017)

1. “Hail Mary”
2. Play card known to be safe

Rule-based agents

Piers (Walton-Rivers *et al*., 2017)

1. “Hail Mary”
2. Play card known to be safe
3. If lives > 1, play card with probability of being playable at least 0.6

Rule-based agents

Piers (Walton-Rivers *et al*., 2017)

1. “Hail Mary”
2. Play card known to be safe
3. If lives > 1, play card with probability of being playable at least 0.6
4. Give hint about playable card

Rule-based agents

Piers (Walton-Rivers *et al*., 2017)

1. “Hail Mary”
2. Play card known to be safe
3. If lives > 1, play card with probability of being playable at least 0.6
4. Give hint about playable card
5. If hint tokens < 4, give hint about a useless card

Rule-based agents

Piers (Walton-Rivers *et al*., 2017)

1. “Hail Mary”
2. Play card known to be safe
3. If lives > 1, play card with probability of being playable at least 0.6
4. Give hint about playable card
5. If hint tokens < 4, give hint about a useless card
6. Discard a card known to be useless

Rule-based agents

Piers (Walton-Rivers *et al*., 2017)

1. “Hail Mary”
2. Play card known to be safe
3. If lives > 1, play card with probability of being playable at least 0.6
4. Give hint about playable card
5. If hint tokens < 4, give hint about a useless card
6. Discard a card known to be useless
7. Give a random hint

Rule-based agents

Piers (Walton-Rivers *et al*., 2017)

1. “Hail Mary”
2. Play card known to be safe
3. If lives > 1, play card with probability of being playable at least 0.6
4. Give hint about playable card
5. If hint tokens < 4, give hint about a useless card
6. Discard a card known to be useless
7. Give a random hint
8. Discard a card randomly

Historical Context

Rule-based agents at the time

Historical Context

Rule-based agents at the time

- Most agents at the time: handcrafted rules (e.g. Piers), in a fixed order

Historical Context

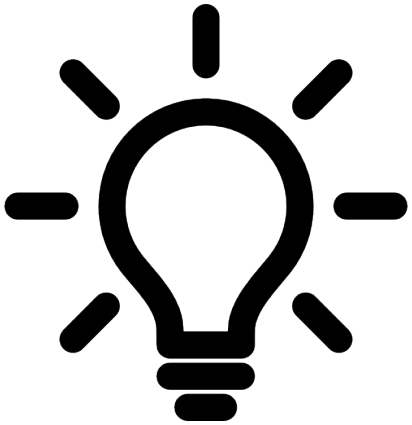
Rule-based agents at the time

- Most agents at the time: handcrafted rules (e.g. Piers), in a fixed order
    - One agent by (Van den Bergh, 2016) used exhaustive search to look for best combinations of parameters for variations of pre-defined rules.

Historical Context

Rule-based agents at the time

- Most agents at the time: handcrafted rules (e.g. Piers), in a fixed order
    - One agent by (Van den Bergh, 2016) used exhaustive search to look for best combinations of parameters for variations of pre-defined rules.



Use a Genetic Algorithm (GA) to search both for *which* rules to use and in *which order.*

Genetic Algorithm

Overview

Genetic Algorithm

Overview

1. Initialize population

Genetic Algorithm

Overview

1. Initialize population
2. For G generations:

Genetic Algorithm

Overview

1. Initialize population
2. For G generations:
    (a) Evaluate population

Genetic Algorithm

Overview

1. Initialize population
2. For G generations:
    (a) Evaluate population
    (b) Keep *e* best individuals (elitism)

Genetic Algorithm

Overview

1. Initialize population
2. For G generations:
    (a) Evaluate population
    (b) Keep *e* best individuals (elitism)
    (c) Generate children by crossover (w/ tournament selection) and mutation

Genetic Algorithm

Initialization

Ruleset (100+ rules)

1 - Hint randomly	5 - Play if certain	9 - Discard useless
2 - Hint playable	6 - Play just hinted	10 - Discard oldest
3 - Hint useless	7 - Play if (p>0.7)	11 - Discard newest
4 - Hint most cards	8 - Play most recent	12 - Discard randomly

Chromosome (genotype)

5 - Play if certain
2 - Hint playable
7 - Play if (p>0.7)
9 - Discard useless
10 - Discard oldest

Policy (phenotype)

Genetic Algorithm

Fitness evaluation

Genetic Algorithm

Fitness evaluation

5 - Play if certain
2 - Hint playable
7 - Play if (p>0.7)
9 - Discard useless
10 - Discard oldest

Policy (phenotype)

Genetic Algorithm

Fitness evaluation

5 - Play if certain
2 - Hint playable
7 - Play if (p>0.7)
9 - Discard useless
10 - Discard oldest

Policy (phenotype)

Play *n* games
with desired partners

Genetic Algorithm

Fitness evaluation

5 - Play if certain
2 - Hint playable
7 - Play if (p>0.7)
9 - Discard useless
10 - Discard oldest

Policy (phenotype)

Play *n* games
with desired partners

- **Mirror**: self-play
- **Mixed**: 7 agents from (Walton-Rivers, *et al*., 2017)

Genetic Algorithm

Fitness evaluation

5 - Play if certain
2 - Hint playable
7 - Play if (p>0.7)
9 - Discard useless
10 - Discard oldest

Policy (phenotype)

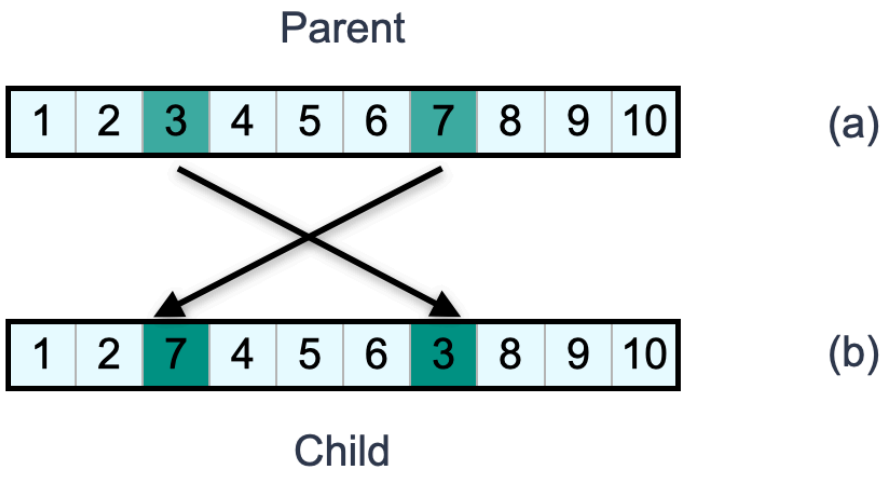
Play *n* games
with desired partners

- **Mirror**: self-play
- **Mixed**: 7 agents from (Walton-Rivers, *et al*., 2017)

Score (Fitness)

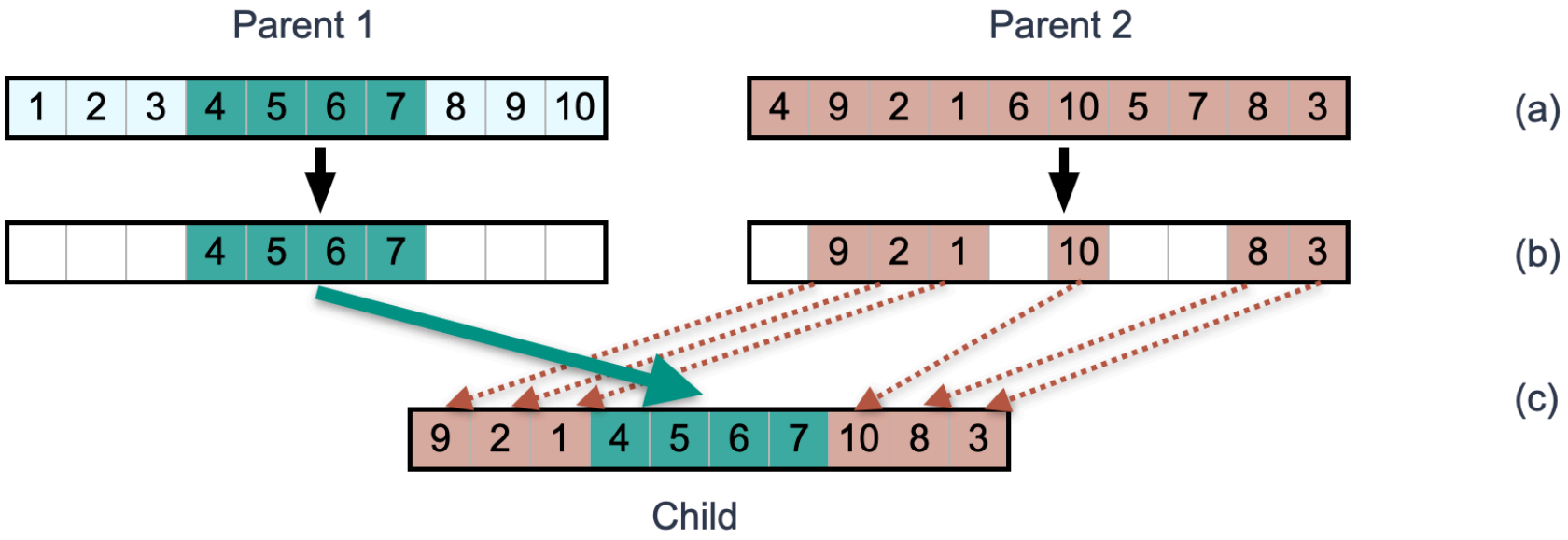
Genetic Algorithm

Swap Mutation Operator



Genetic Algorithm

Ordered Crossover Operator



Genetic Algorithm

Results - 2 players

Agent	Mirror	Mixed (Our Training Pool)	Mixed (Competition Pool)
Piers	16.99	11.28	N/A
GA (existing rules)	19.35	12.13	N/A
GA (new rules)	20.07	12.29	13.40
IS-MCTS (competition winner)	20.5	N/A	13.99

Variation of Piers discovered by the GA

Piers (Walton-Rivers *et al.*, 2017) **Score** = 17.09

1. “Hail Mary”
2. Play card known to be safe
3. If lives > 1, play card with probability of being playable at least 0.6
4. Give hint about playable card
5. If hint tokens < 4, give hint about a useless card
6. Discard a card known to be useless
7. Give a random hint
8. Discard a card randomly

Variation of Piers discovered by the GA

Piers (Walton-Rivers *et al*., 2017) **Score** = 17.09

1. “Hail Mary”
2. Play card known to be safe
3. If lives > 1, play card with probability of being playable at least 0.6
4. Give hint about playable card
5. If hint tokens < 4, give hint about a useless card
6. Discard a card known to be useless
7. Give a random hint
8. Discard a card randomly

Variation of Piers **Score** =17.31

1. “Hail Mary”
2. Play card known to be safe
3. If lives > 1, play card with probability of being playable at least 0.6
4. Give hint about playable card
5. [redacted]
6. Discard a card known to be useless
7. Give a random hint
8. Discard a card randomly