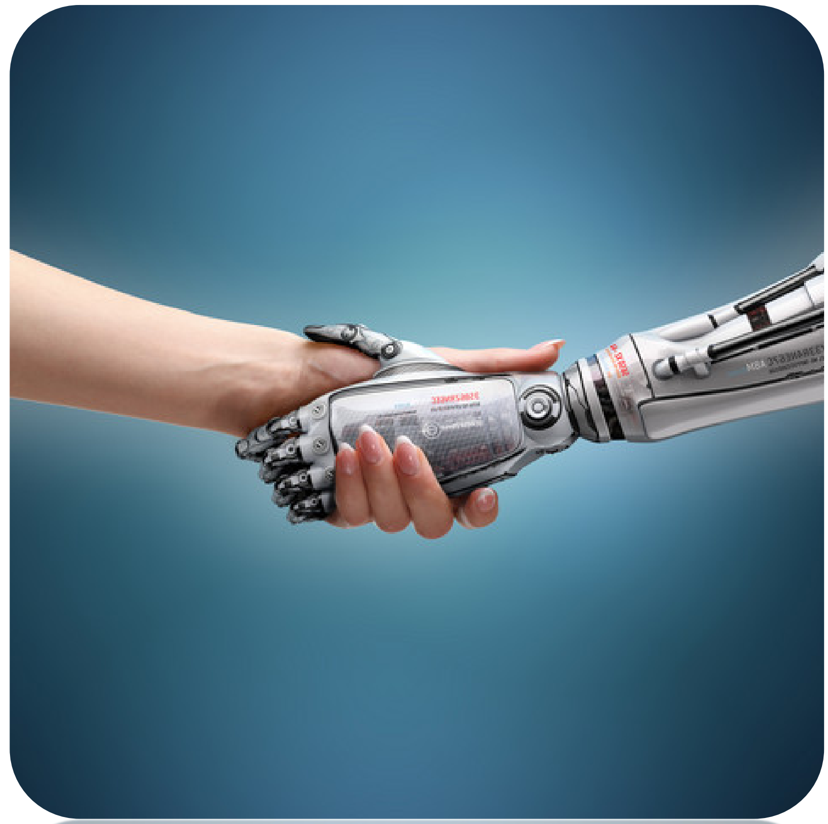
Artificial Intelligence
V04: Local and adversarial search

From hill climbing search to genetic algorithms
Game playing
Resource limits and other difficulties



Based on material by Stuart Russell, UC Berkeley

Zurich University of Applied Sciences and Arts
CAI Centre for Artificial Intelligence (stdm)

2048 leaderboard link

https://goo.gl/meh3Ro

Educational objectives

- **Re-tell the story of** improving **local search** from hill-climbing to genetic algorithms
- **Remember** the **minimax,** $\boldsymbol{\alpha}$**-**$\boldsymbol{\beta}$ and **expectiminimax** algorithms
- **Implement** an **AI agent** for a given simple game

"In which we relax the simplifying assumptions of the previous lecture, thereby getting closer to the real world; including the problems that arise when we try to plan ahead in a world where other agents are planning against us."

➔ Reading: AIMA, [ch. 4.1-4.2 (local search)]; ch. 6 (games)

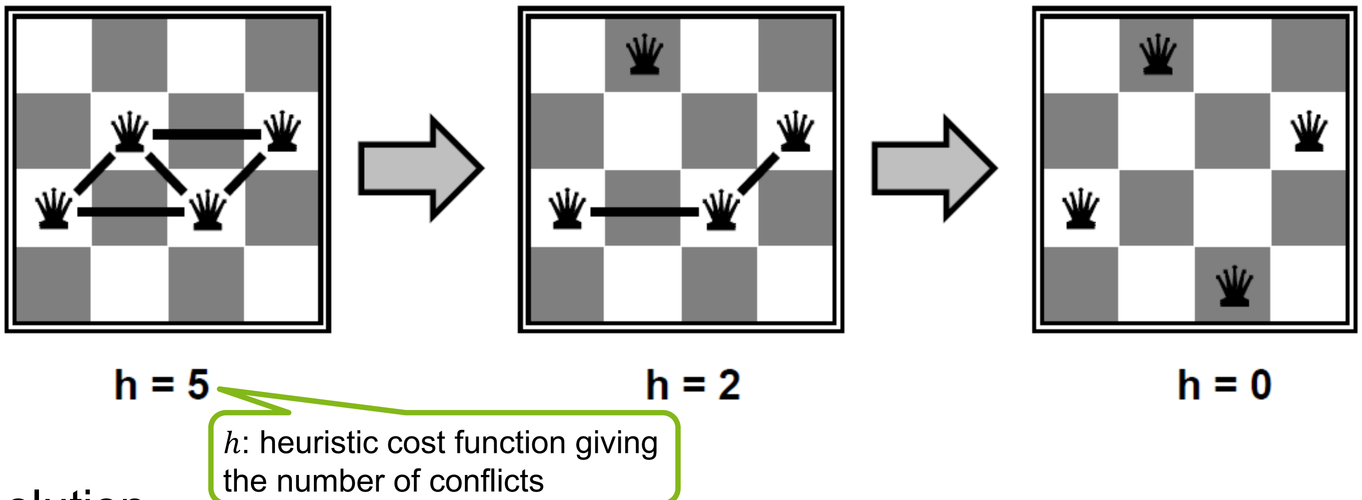
1. FROM HILL CLIMBING SEARCH TO GENETIC ALGORITHMS

Local search

Example: n-queens problem

Task

- Put n queens on a $n \times n$ board with no two queens on the same row, column, or diagonal



Possible solution

- Initialize one queen per column
- Move **one queen up/down at a time** to reduce number of conflicts using heuristic h
- Almost always solves n-queens problems almost instantaneously (#states: n^n)
  ➔ works for very large n, e.g., $n = 1'000'000$

Iterative improvement algorithms

Local search: search for optimal states instead of path’s

- In many optimization problems, **path is irrelevant**; the **goal** state **itself is** the **solution**
  ➔ State space: set of "complete" configurations;
  ➔ Goal: find optimal configuration (or a configuration satisfying constraints)
- Examples: TSP, timetable

Iterative improvement

- In such cases: use iterative improvement algorithms
  ➔ **Keep** a **single** "current" **state**, try to improve it
  ➔ Constant space, suitable for online as well as offline search

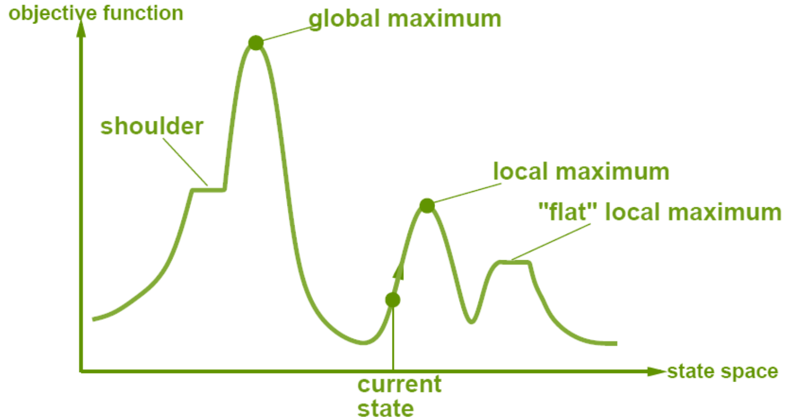
Possible implementations

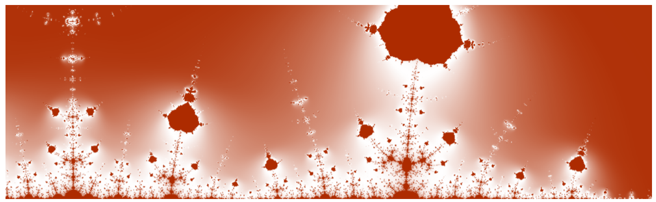
- Hill climbing
- Simulated annealing
- Genetic algorithms

Hill climbing search
(a.k.a. gradient ascent/descent)

Systematic search for an optimum
- Analogy: *«Like climbing Everest in thick fog with amnesia»*
- Result: finds a state that is a local maximum
  …by selecting only the highest-valued successor for expansion **iif** its value is better

The state space landscape
- Practical problems typically have an **exponential number of local maxima** to get stuck in
- **Random-restart** hill climbing overcomes local maxima ➔ trivially complete
- **Random sideways** moves escape from shoulders (good), loop on flat maxima (bad)

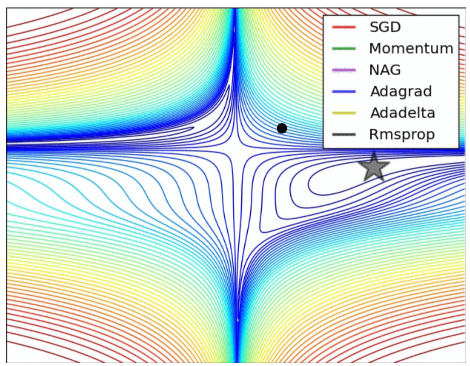




Hill climbing search: an outlook

All previously discussed search algorithms only work in discrete state and action spaces (otherwise the branching factor is infinite)

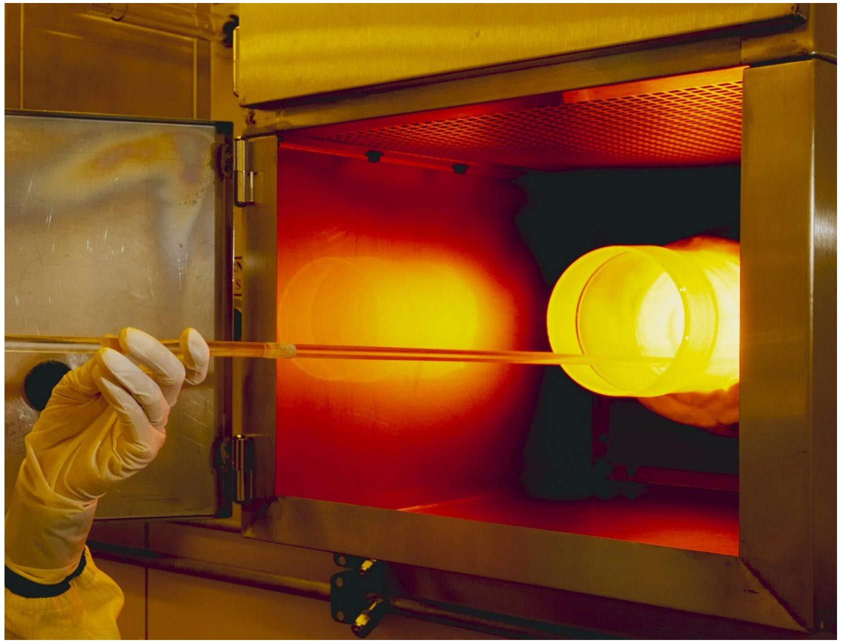
- Hill climbing in continuous space (gradient descent) is the **work horse** of deep learning
- Some pointers:
  → http://sebastianruder.com/optimizing-gradient-descent/ (overview of the gradient descent family)
  → https://stdm.github.io/Some-places-to-start-learning-ai-ml/ (links to courses on deep neural networks)



Simulated annealing
Towards optimizing hill climbing search

Idea (by [Metropolis et al., 1953] for physical process modelling)
- Escape local maxima by allowing some “bad” moves
- …but **gradually decrease** their **size and frequency**

Application
- For **“good” schedule** of decreasing the temperature (→ see appendix), it **always reaches** the **best state**
- **Widely applied** for, e.g., VLSI layout, airline scheduling



Modern variants
- «momentum», «Adam» and other adaptation strategies for a «learning rate»
  → first link on the last slide

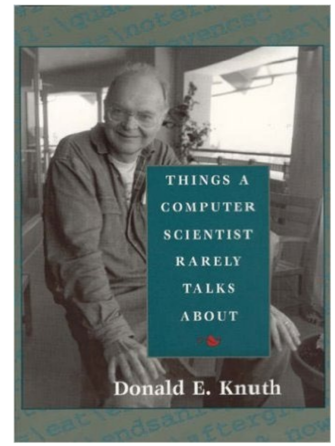
Local beam search
…and still optimizing hill climbing search

Not: each

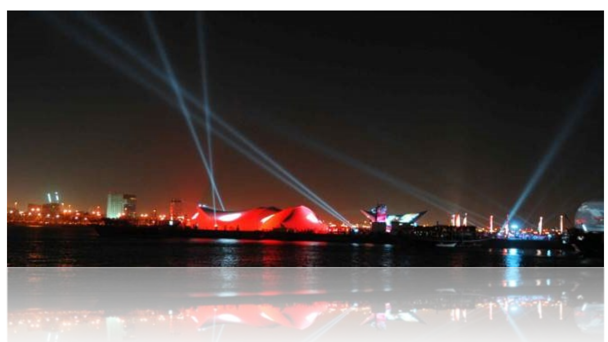
Idea
- **Keep k states** instead of 1; **choose top k** of all their **successors**
  (not the same as k searches run in parallel! → Why?)
- **Searches** that find good states **recruit other** searches **to join** them

Problem
- Quite often, all k states end up on same local hill

Idea contd.
- **Choose** k successors **randomly**, **biased towards good** ones
  ➔ Observe the close analogy to natural selection!



Compare Don Knuth on "the advantages of unbiased sampling as a way to gain insight into a complicated subject" (e.g., ch. 2 in the above book)

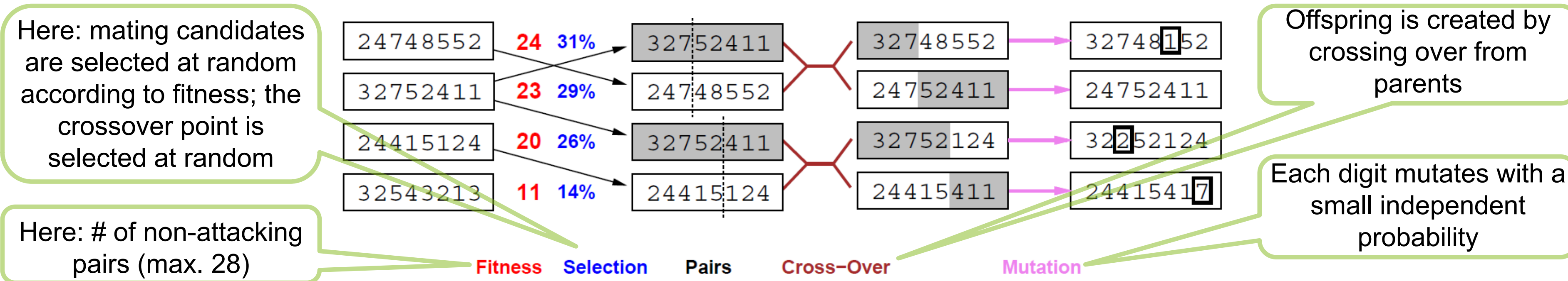


Genetic algorithms (GA)

…improving on the idea of local beam search

Idea

- Combine stochastic local beam search + generating successors from *pairs* of states
  ➔ **uphill tendency** + **random exploration** + **exchange of information** among searches



Example: 8-queens states encoded as digit strings. The original population (left) is ranked by a fitness function, resulting in pairs for mating. The offspring is subject to mutation.

Application

- GAs **require states** encoded **as strings**
- Crossover helps iif **substrings are meaningful** components
- GAs ≠ evolution

2. GAME PLAYING

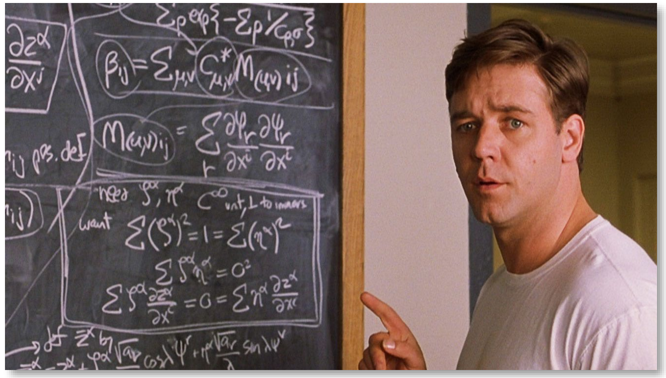
Adversarial search

Games vs. search problems

“Unpredictable" opponent
➔ solution is a **strategy** (specifying a move for every possible opponent reply)

Time limits
➔ unlikely to find goal, must **approximate**

Which movie?

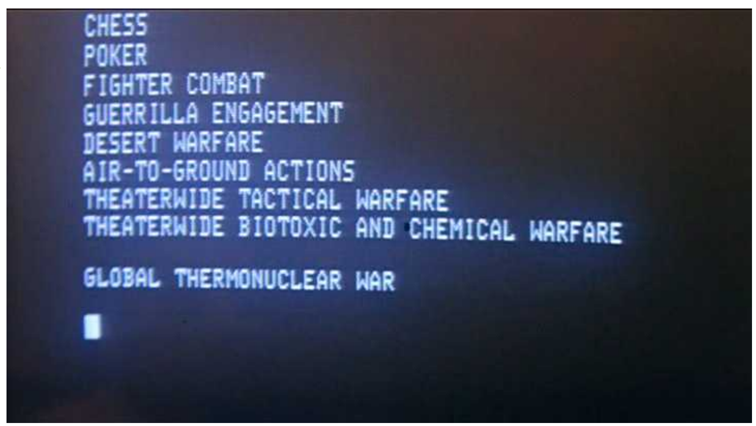


Early history:
- Computer considers possible lines of play (computer chess: Babbage, 1846)
- Algorithm for perfect play (minimax: Zermelo, 1912; game theory: von Neumann, 1944)
- Finite horizon, approximate evaluation (depth cut-off: Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (RL for checkers: Samuel, 1952-57)
- Pruning to allow deeper search (α-β search: McCarthy, 1956)

Types of games

	deterministic	stochastic
perfect information	**chess**, checkers («Dame»), go, othello («Reversi»)	backgammon, **monopoly**
only partial observability	**battleship**, kriegspiel (chess without seeing enemy pieces)	bridge (~ «Jass», «Skat»), **poker**, scrabble, *global thermonuclear war*

Which movie?



Deterministic (turn-based, 2-player) games

The search tree, e.g. of tic-tac-toe

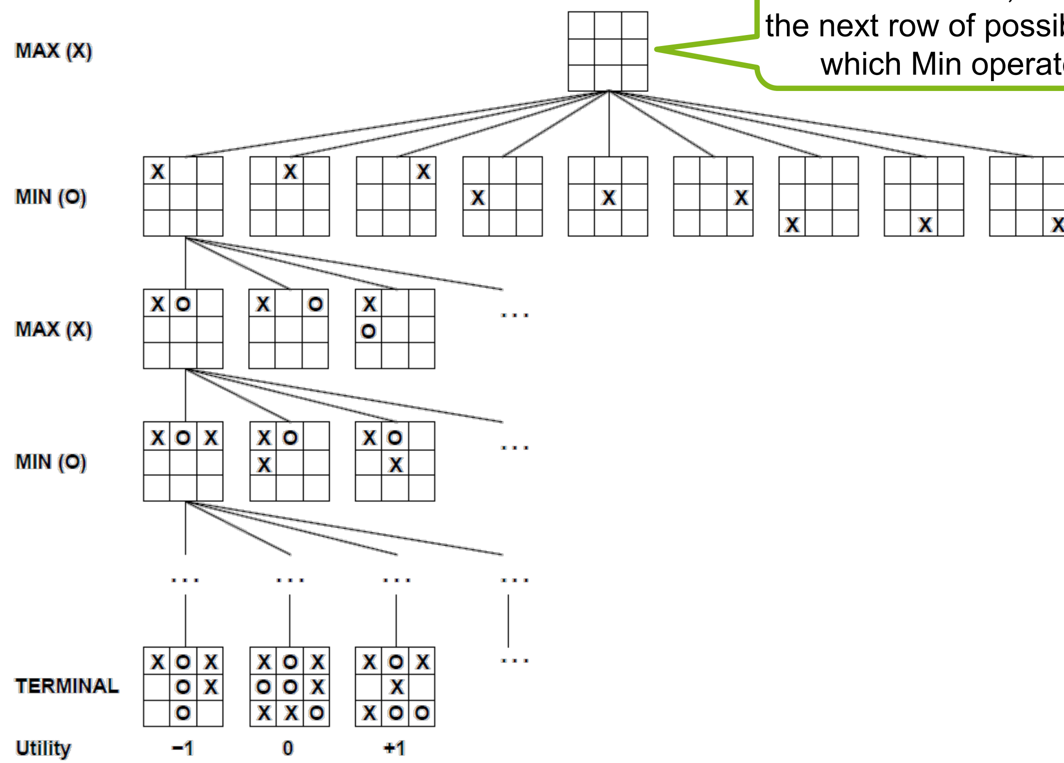
Player's name

„Max"
- 1st player (moves first)
- Wants to **maximize utility** of terminal states
- Tree shows Max´s perspective

„Min"
- 2nd player
- Wants to **minimize** (Max´s) **utility**

Utility
- Numeric value („payoff") of terminal state
- „zero-sum game" iif total payoff (to all players) is constant over all game instances

Max moves first, resulting in the next row of possible states which Min operates on



Minimax: depth-first exploration of game tree

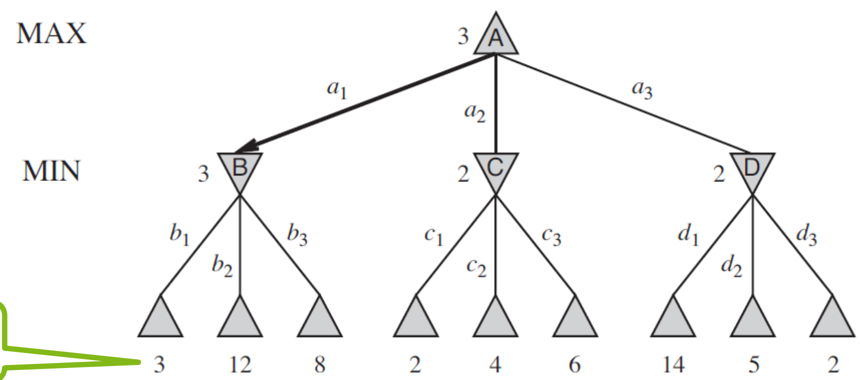
Perfect play for deterministic, fully observable games

Idea

- Choose move to position with highest minimax value
- Minimax value: **highest value among options minimized** by adversary
  ➔ **best** achievable payoff **against best play**

Example

- Any 2-ply game tree (i.e., each player moves once)
- Max´s best move at root: a_1 (leading to highest minimax value of 3)
- …because Min´s best reply will be b_1 (leading to lowest minimax value / utility of 3)



Min is going to do the least valuable thing here for Max (3).

Minimax (contd.)
Algorithm and properties

```
function Minimax-Decision(state) returns an action
    inputs: state, current state in game
    return the a in Actions(state) maximizing Min-Value(Result(a, state))

function Max-Value(state) returns a utility value
    if Terminal-Test(state) then return Utility(state)
    v ← −∞
    for a, s in Successors(state) do v ← Max(v, Min-Value(s))
    return v

function Min-Value(state) returns a utility value
    if Terminal-Test(state) then return Utility(state)
    v ← ∞
    for a, s in Successors(state) do v ← Min(v, Max-Value(s))
    return v
```

Min-Value

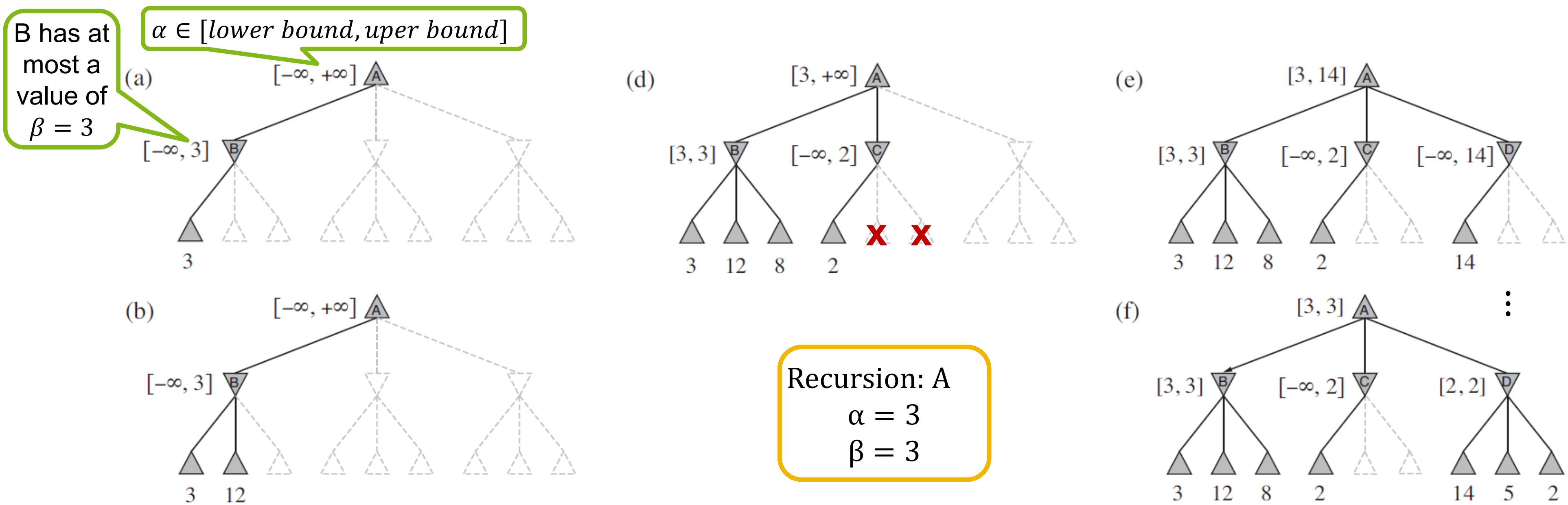
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

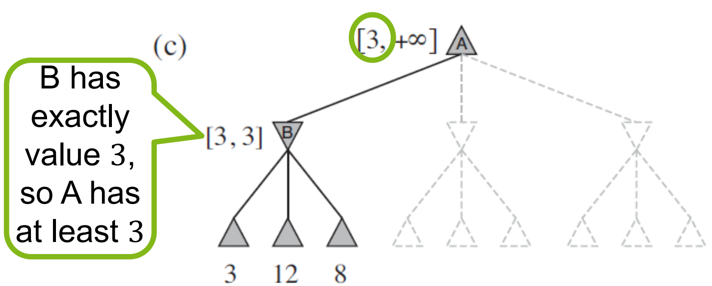
Max-Value

Properties
- Complete? **Yes**, if tree is finite (e.g., chess has specific rules for this)
- Optimal? **Yes**, against an optimal opponent (Otherwise?)
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$ (for depth-first exploration)
- For chess, $b = 35$, $m = 100$ for “reasonable" games
  ➔ exact solution completely **infeasible**; but do we need to explore every path?

α-β pruning example

Overcoming exponential (b^m) number of states to be explored





Successively tightening bounds on minimax values

- α **is** the **best value** (to Max) found so far in current subtree of a Max node (A→α)
- If any node v is worse than α, Max will not choose it
  ➔ **prune** that **branch**
- Similarly: β is best score Min is assured of in current subtree of a Min node (B, C, D→ β)

α-β pruning (contd.)
Algorithm and properties (changes to minimax in *bold-italic*)

```
function Alpha-Beta-Search(state) returns an action
    v ← Max-Value(state, −∞, ∞)
    return the a in Actions(state) with value v

function Max-Value(state, α, β) returns a utility value
    if Terminal-Test(state) then return Utility(state)
    v ← −∞
    for a in Actions(state) do
        v ← Max(v, Min-Value(Result(state, a), α, β))
        if v ≥ β then return v
        α ← Max(α, v)
    return v

function Min-Value(state, α, β) returns a utility value
    if Terminal-Test(state) then return Utility(state)
    v ← ∞
    for a in Actions(state) do
        v ← Min(v, Max-Value(Result(state, a), α, β))
        if v ≤ α then return v
        β ← Min(β, v)
    return v
```

Properties
- Pruning **does not affect** final **result**
- Good **move ordering improves** effectiveness of pruning
- With "perfect ordering", time complexity $= O(b^{m/2})$
  ➔ **doubles solvable depth**
  ➔ a simple example of the value of reasoning about which computations are relevant (**metareasoning**)
  ➔ unfortunately, $35^{100/2}$ (for chess) is still impossible!

3. RESOURCE LIMITS AND OTHER DIFFICULTIES

Adversarial search (contd.)

Resource limits
Towards real-world conditions

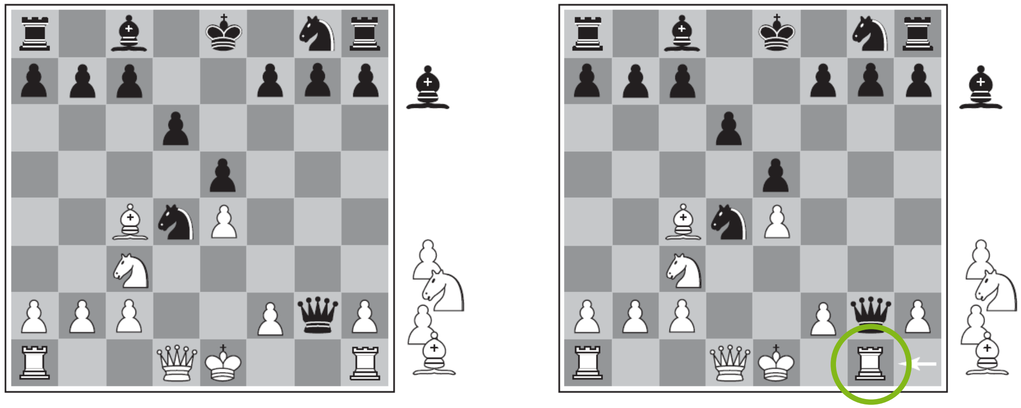
Standard approach

- Use **Cutoff-Test** instead of Terminal-Test
  e.g., depth limit (perhaps add **quiescence search**: only cut off search at positions that don´t drastically change their value in the near future, e.g. captures in chess; otherwise continue search)
- Use **Eval** instead of Utility
  i.e., evaluation function that estimates desirability of position
- **Lookup** of start/end games

Example

- Suppose we have 100 seconds, explore 10^4 nodes/second
  ➔ 10^6 nodes per move $\approx 35^{8/2}$
- α-β reaches depth 8
  ➔ pretty good chess program

Eval(uation) functions
Designing or learning effective cutoff tests



(a) White to move (b) White to move

Example: The two chess positions **differ only in the position of the rook** (“Turm”) at lower right.
In (**a**), **Black has an advantage** of a knight (“Springer”) and two pawns (“Bauern”) → should be enough to win.
In (**b**), **White will capture the queen** (“Dame”) → should be strong enough to win.

For chess, typically **linear weighted sum of features**

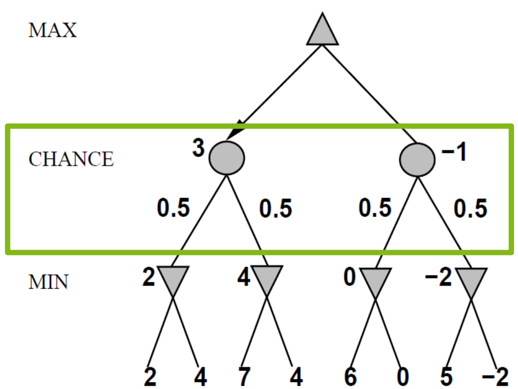
- $\mathrm{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
- Example:
  $w_1=9$ and $f_1(s)=(\text{number of white queens})-(\text{number of black queens})$
- **Can be learned** with machine learning techniques

Nondeterministic (stochastic) games

Chance is introduced by e.g. dice-rolling or card-shuffling

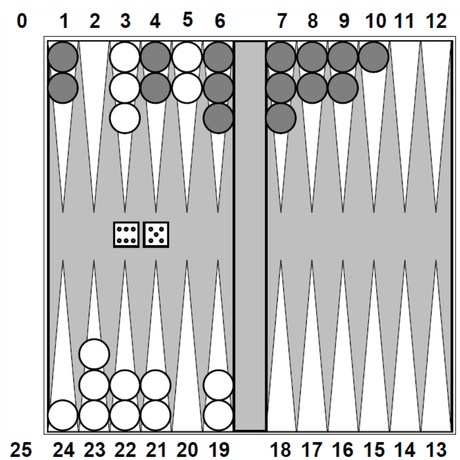
Simplified example

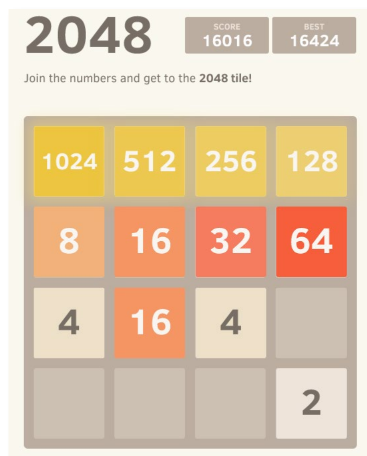
- A game with coin-flipping
- Nondeterminism **is handled by an additional level in the tree**, consisting of chance nodes



Real-world example

- **2048**: numbers appear with probability $P(2) = \frac{9}{10}$ and $P(4) = \frac{1}{10}$ at random free board positions
- **Backgammon**: Before each move, dice-rolls determine the legal moves





Expectiminimax – maximizing expected value
Algorithm and properties

```
function ExpectiMinimax-Decision(state) returns an action
    inputs: state, current state in game
    return a in Actions(state) maximizing
           ExpectiMinimax-Value(Result(a, state))

function ExpectiMinimax-Value(state) returns a utility value
    if Terminal-Test(state) then
        return Utility(state)
    if state is a Max node then
        return highest ExpectiMinimax-Value of Successors(state)
    if state is a Min node then
        return lowest ExpectiMinimax-Value of Successors(state)
    if state is a chance node then
        return average of ExpectiMinimax-Value of Successors(state)
```

$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$

Properties

- Algorithm **works** just **like Minimax** – except chance-nodes are also handled
- Expectiminimax gives **perfect play**
- In case of only **1 player**, Expectiminimax becomes **Expectimax**
- Time complexity: $O(b^m n^m)$ (where n is the number of distinct random events, e.g. dice rolls)
  ➔ **Possibilities** are **multiplied enormously** in games of chance
  ➔ Simultaneously, **no** likely sequences exist to do effective ***α-β* pruning**

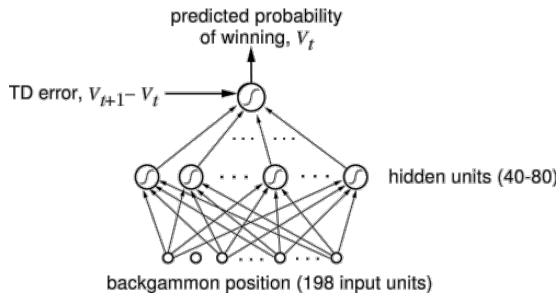
Nondeterministic games in practice

Example Backgammon

- Dice rolls increase b (21 possible rolls with 2 dice)
- Ca. 20 legal moves (can be $6{,}000$ with 1-1 roll)
  ➔ at depth 4: $20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$ nodes
- As depth increases, probability of reaching a given node shrinks
  ➔ value of lookahead is diminished (see appendix for consequences)

But

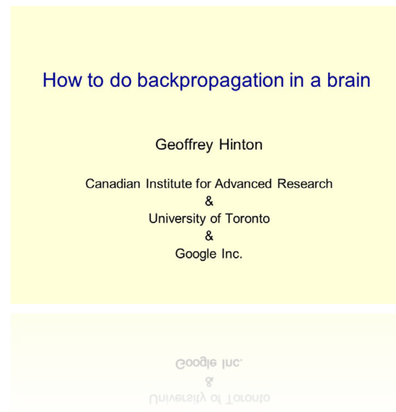
- «TDGammon» (Tesauro, 1992) uses depth-2 search ≈ world-champion level
  ➔ uses neural network and reinforcement learning (RL) to train Eval function via games against itself



- ➔ see also http://webdocs.cs.ualberta.ca/~sutton/book/ebook/node108.html

Where's the intelligence?
Man vs. machine

- Searching over the right state spaces leads to **emergent "clever" behavior** in games (information gathering, alliances, bluffs)
- Many local and adversarial search methods can be **enhanced by learning** (e.g., learn good Eval functions by playing games against oneself)
- The brain might perform local (hill climbing) search as well
  → see https://www.cs.toronto.edu/~hinton/backpropincortex2014.pdf



➔ Hill-climbing, minimax, α-β, etc. are still pure computation
➔ Intelligent behavior emerges from their composition on **suitable** data structures

Review

- **Local search** algorithms **evolve** a **small number of states** towards better utility (typical: 1 state)

- In **continuous space**, local search by linear programming or convex optimization is extremely efficient in practice (polynomial time complexity!)

- In non-deterministic environments, **keeping track of** one´s **belief state** is paramount

- Games are fun to work on – and dangerous

- They illustrate several important points about AI
  - **perfection** is **unattainable** ➔ must **approximate**
  - good idea to **think about what to think** about (metareasoning ➔ pruning)
  - **uncertainty** constrains the assignment of values to states
  - optimal decisions depend on information state (belief state), not real state

➔ Games are to AI as grand prix racing is to automobile design

Zurich University
of Applied Sciences

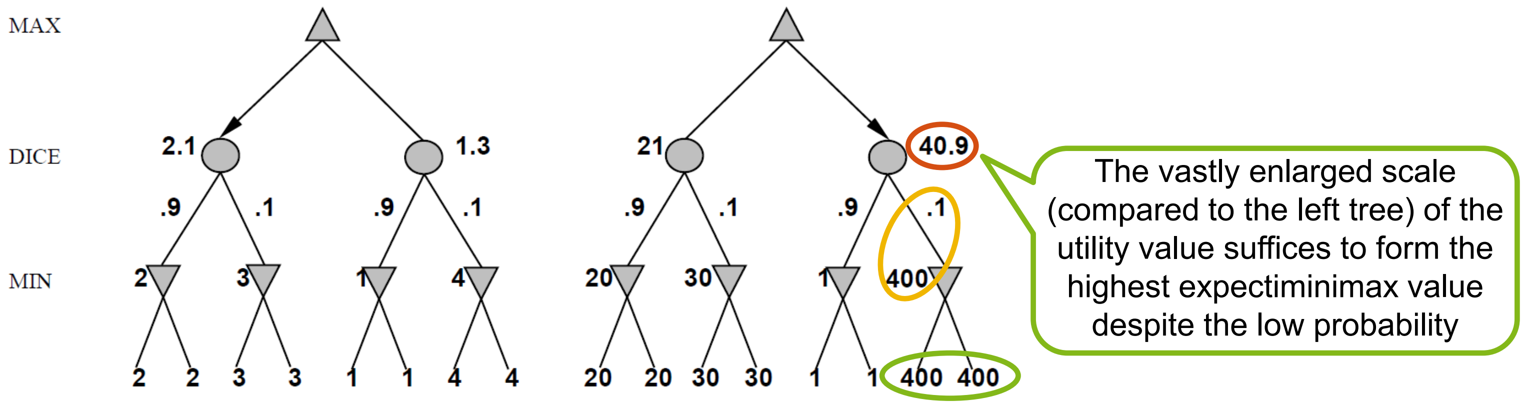
APPENDIX

Pseudocode for hill climbing search and simulated annealing

```
function Hill-Climbing(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbour, a node
    current ← Make-Node(Initial-State[problem])
    loop do
        neighbour ← a highest-valued successor of current
        if Value[neighbour] ≤ Value[current] then return State[current]
        current ← neighbour
    end
```

```
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 the probability 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 Δ E > 0 then current ← next
        else current ← next only with probability e^(ΔE/T)
```

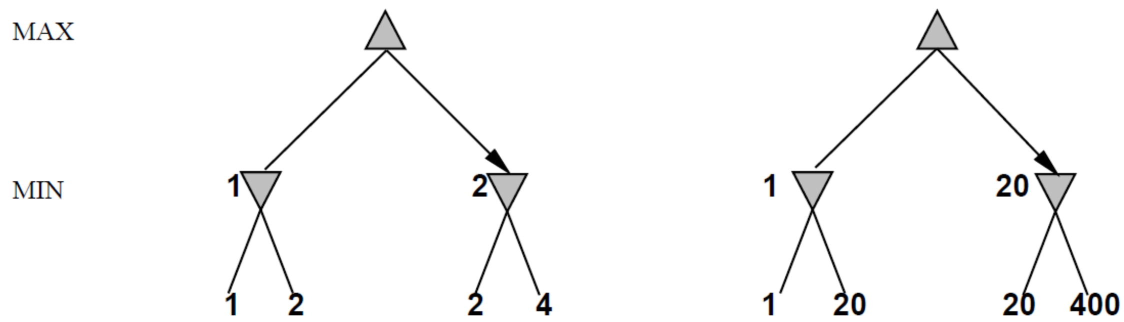
Digression: Exact Eval values do matter
…for nondeterministic games



- Behavior is **preserved only by positive linear transformation** of Eval
- Hence **Eval** should be **proportional to** the expected **payoff**

➔ Exact values **don´t matter for deterministic** games → see appendix

Digression: Exact Eval values don't matter
…for deterministic games



- Behavior is preserved under any **monotonic** transformation of Eval
- Only the **order matters**: payoff in deterministic games acts as an **ordinal utility** function

Zurich University
of Applied Sciences

Deterministic games in practice

Checkers

- *«Chinook»* ended 40-year-reign of human world champion Marion Tinsley in 1994
- Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board (total: $443{,}748{,}401{,}247$)

Chess

- *«Deep Blue»* defeated human world champion Gary Kasparov in a six-game match in 1997
- Searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply

Othello

- Human champions refuse to compete against computers, who are **too good**

Go

- 2010: human champions refuse to compete against computers, who are **too bad**
- 2016: Google DeepMind´s *«AlphaGo»* unexpectedly defeats world champion Lee Sedol
- $b > 300$ ➔ most programs use pattern knowledge bases to suggest plausible moves

Expectations & expected values
adapted from U Washington`s CSE473, lecture 8

The expectation operator $E()$
- We can define a function $f(X)$ of a random variable X
- The expected value of a function is its **average** value **under** the **probability distribution over** the function's **inputs:**

$$E\big(f(X)\big) = \sum_{x} f(X = x)P(X = x)$$

Example
- How long to drive to the airport?
- Driving time D (in mins) as a function of traffic T:
  $D(T = none) = 20, D(T = light) = 30, D(T = heavy) = 60$
- What is your expected driving time?
  Let probability $P(T) = \{none\colon 0.25,\ light\colon 0.5,\ heavy\colon 0.25\}$
  ➔ $E\big(D(T)\big) = D(none)P(none) + D(light)P(light) + D(heavy)P(heavy)$
  ➔ $E\big(D(T)\big) = 20 * 0.25 + 30 * 0.5 + 60 * 0.025 = 35$ mins

Why averaging over clairvoyance is wrong
A common sense example of a journey

Day 1
- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - the left fork **leads to** a bigger heap of gold;
  - take the right fork and you'll be run over by a bus.

➔ „B“ (and then „left“) is the optimal choice

Day 2
- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - **take** the left fork and you'll be run over by a bus;
  - the right fork leads to a bigger heap of gold.

➔ „B“ (and then „right“) is the optimal choice

Day 3
- Road A leads to a small heap of gold pieces
- Road B leads to a fork:
  - **guess** correctly and you'll find a bigger heap of gold;
  - guess incorrectly and you'll be run over by a bus.

➔ „B“ still seems optimal; but this **ignores** the resulting **belief state** (that includes ignorance & possibility of death!)

Averaging over clairvoyance **will never** select actions to **gather information** because it assumes future states to be of perfect knowledge after the initial deal.