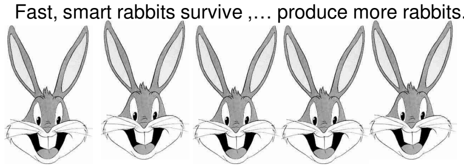
The Genetic Algorithm (Evolutionary Analogy)

- Consider a population of rabbits:
  - some individuals are faster and smarter than others
  - Slower, dumper rabbits are likely to be caught and eaten by foxes

  - Fast, smart rabbits survive ,… produce more rabbits.



Evolutionary Analogy

- The rabbits that survive generate offspring, which start to mix up their genetic material
- Furthermore, nature occasionally throws in a wild properties because genes can mutate
- In this analogy, an individual rabbit represents a solution to the problem(i.e. Single point in the space)
- The foxes represent the problem constraints (solutions that do more well are likely to survive)

Evolutionary Analogy

- For selection, we use a fitness function to rank individuals of the population
- For reproduction, we define a crossover operator which takes state descriptions of individuals and combine them to create new ones
- For mutation, we can choose individuals in the population and alter part of its state.

The Genetic Algorithm

- Directed search algorithms based on the mechanics of biological evolution
- Developed by John Holland, University of Michigan (1970's)
- To design artificial systems software that retains the robustness of natural systems
- Provide efficient, effective techniques for search problems, optimization and machine learning applications
- Widely-used today in business, scientific and engineering circles

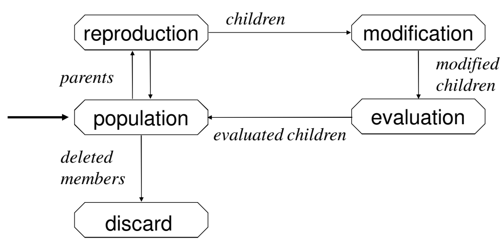
Terminology

- *Evolutionary Computation (EC)* refers to computer-based problem solving systems that use computational models of evolutionary process.
- *Chromosome* – It is an individual representing a candidate solution of the optimization problem.
- *Population* – A set of chromosomes.
- *gene* – It is the fundamental building block of the chromosome, each gene in a chromosome represents each variable to be optimized. It is the smallest unit of information.
- Objective: To find "a" best possible chromosome for a given problem.

Overview of GAs

- GA emulate genetic evolution.
- A GA has distinct features:
  - A string representation of chromosomes.
  - A selection procedure for initial population and for off-spring creation.
  - A cross-over method and a mutation method.
  - A fitness function.
  - A replacement procedure.
- Parameters that affect GA are initial population, size of the population, selection process and fitness function.

The GA Cycle of Reproduction

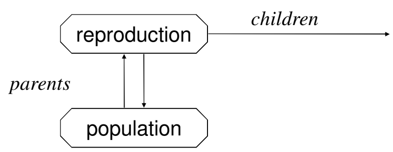


Chromosomes

Chromosomes could be:

Bit strings	(0101 ... 1100)
Real numbers	(43.2 -33.1 ... 0.0 89.2)
Permutations of element	(E11 E3 E7 ... E1 E15)
Lists of rules	(R1 R2 R3 ... R22 R23)
Program elements	(genetic programming)
... any data structure ...	

Reproduction



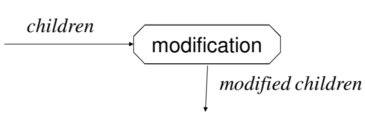
Parents are "selected" at each iteration.

Selection Process

- Selection is a procedure of picking parent chromosome to produce off-spring.
- Types of selection:
  - Random Selection – Parents are selected randomly from the population.
  - Proportional Selection – probabilities for picking each chromosome is calculated as:

$$P(\mathbf{x}_i) = f(\mathbf{x}_i)/\Sigma f(\mathbf{x}_j) \quad \text{for all } j$$

Chromosome Modification



- Operator types are:
  - Mutation
  - Crossover (recombination)

Crossover

P1 (0 1 1 0 1 0 0 0) (1 1 0 1 1 0 0 0) C1
P2 (1 1 0 1 1 0 1 0) (0 1 1 0 1 0 1 0) C2

Crossover is a critical feature of genetic algorithms:

- It greatly accelerates search early in evolution of a population
- It leads to effective combination of schemata (subsolutions on different chromosomes)

Mutation: Local Modification

Before: (1 0 1 1 0 1 1 0)

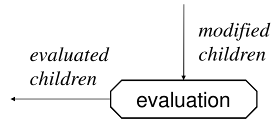
After: (1 0 1 1 1 1 1 0)

Before: (1.38 -69.4 326.44 0.1)

After: (1.38 -67.5 326.44 0.1)

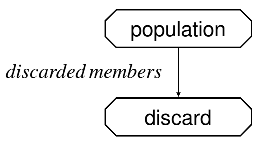
- Causes movement in the search space (local or global)
- Restores lost information to the population

Evaluation

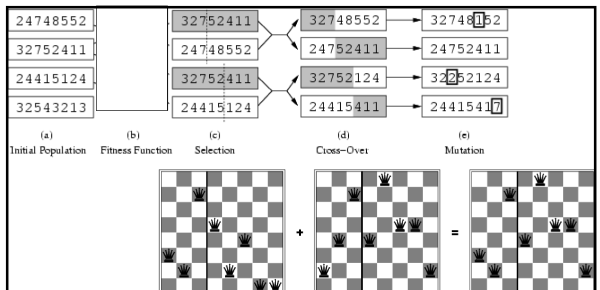


- The evaluator decodes a chromosome and assigns it a fitness measure

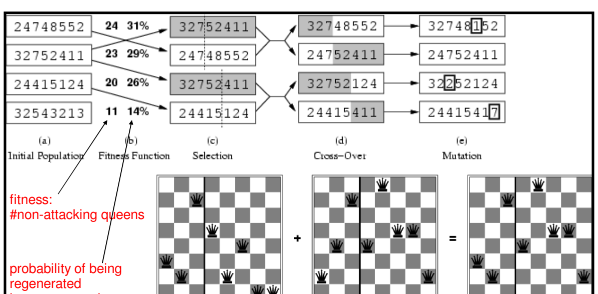
Deletion



- *Generational* GA: entire populations replaced with each iteration
- *Steady-state* GA: a few members replaced each generation



- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)



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

Creativity in GA

- ✓ GAs can be thought of as a simultaneous, parallel hill climbing search --- The population as a whole is trying to converge to an optimal solution
- ✓ Because solutions can evolve from a variety of factors, very novel solutions can be discovered

Game Playing

A (pure) strategy:

a complete set of advance instructions that specifies a definite choice for every conceivable situation in which the player may be required to act.

In a two-player game, a strategy allows the player to have a response to every move of the opponent.

Game-playing programs implement a strategy as a software mechanism that supplies the right move on request.

Two-Person Perfect Information Deterministic Game

- Two players take turns making moves
- Call one Min and the other Max
- Deterministic moves: Board state fully known,
- One player wins by defeating the other (or else there is a tie)
- Want a strategy to win, assuming the other person plays rationally

A logic-based approach to games

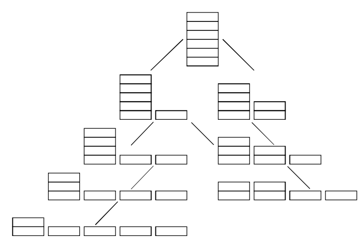
Find a winning strategy by *proving* that the game can be won -- use backward chaining.

A very simple game: nim.

- initially, there is one stack of chips;
- a move: select a stack and divide it in two unequal non-empty stacks;
- a player who cannot move loses the game.

(The player who moves first can win.)

Nim



Static evaluation

A **static evaluation function** returns the value of a move without trying to play (which would mean simulating the rest of the game but not playing it).

"Usually" a static evaluation function returns positive values for positions advantageous to **Player 1**, negative values for positions advantageous to **Player 2.**

If player **Player 1** is rational, he will choose the maximal value of a leaf.
Player **Player 2** will choose the minimal value.

Static evaluation

If we can have (guess or calculate) the value of an internal node **N**, we can treat it as if it were a leaf. This is the basis of the minimax procedure.

No tree would be necessary if we could evaluate the initial position statically. Normally we need a tree, and we need to look-ahead into it. Further positions can be evaluated more precisely, because there is more information, and a more focussed search.

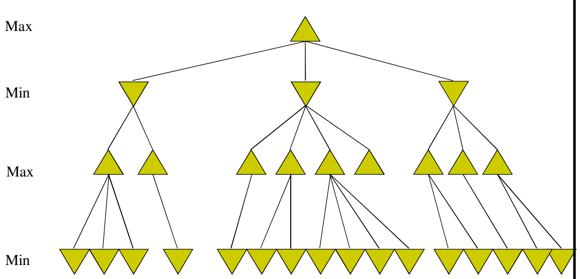
Minimax Tree

- Create a utility function
  - Evaluation of board/game state to determine how strong the position of each player.
  - Player 1 wants to maximize the utility function
  - Player 2 wants to minimize the utility function
- Minimax tree
  - Generate a new level for each move
  - Levels alternate between “max” (player 1 moves) and “min” (player 2 moves)

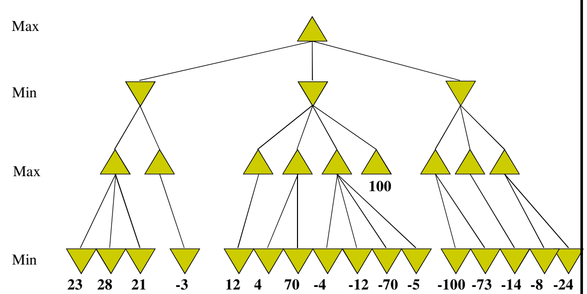
Minimax Tree Evaluation

- Assign utility values to leaves
  - If leaf is a “final” state, assign the maximum or minimum possible utility value (depending on who would win)
  - If leaf is not a “final” state, must use some other heuristic, specific to the game, to evaluate how good/bad the state is at that point

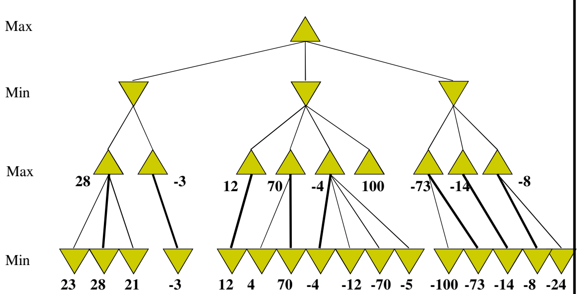
Minimax tree



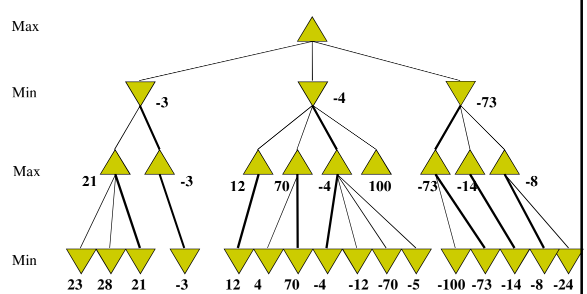
Minimax tree



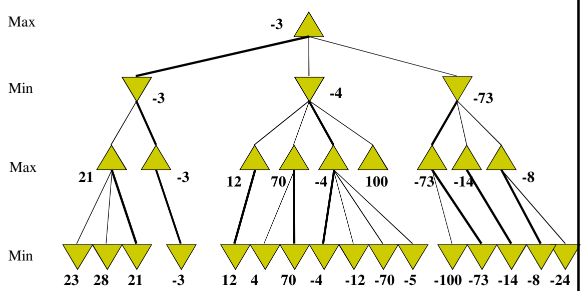
Minimax tree



Minimax tree



Minimax tree



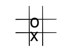
Tic-Tac-Toe

Let player **A** be x and let open(x), open(o) mean the number of lines open to x and o. There are 8 lines. An evaluation function for position P:

$f(P) = -\infty$ if o wins

$f(P) = +\infty$ if x wins

$f(P) = open(x) - open(o)$ otherwise

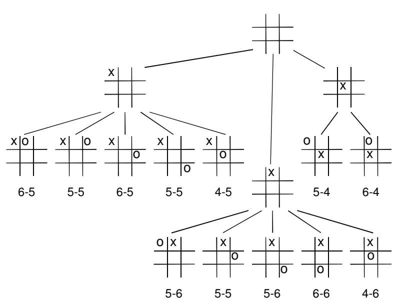


Example:

$open(x) - open(o) = 4 - 6$

Assumptions:

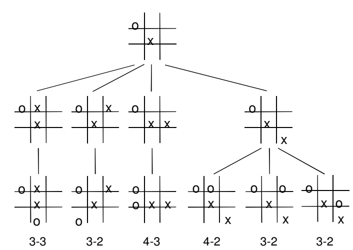
only one of symmetrical positions is generated;

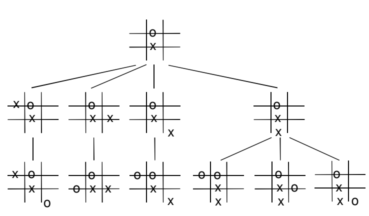


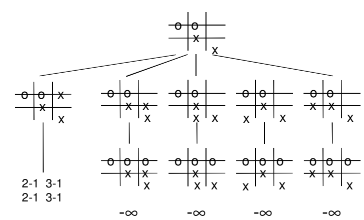
Player **B** chooses the minimal backed-up value among level 1 nodes.

Player **A** chooses the maximal value, and makes the move.

Player **B**, as a rational agent, selects the optimal response.







Building complete plies is usually not necessary. If we evaluate a position when it is generated, we may save a lot.

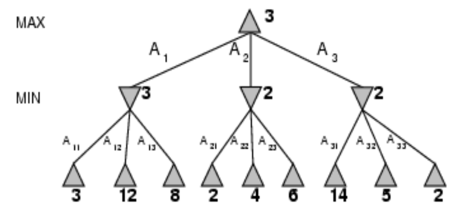
Assume that we are at a minimizing level. If the evaluation function returns $-\infty$, we do not need to consider other positions: $-\infty$ will be the minimum.

The same applies to $+\infty$ at a maximizing level.

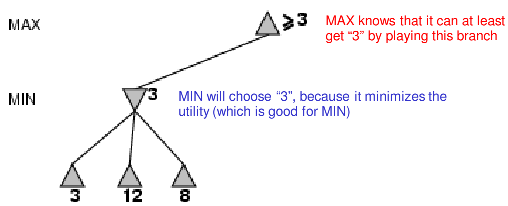
Pruning the Minimax Tree

- Minimax works best for large trees, but it can be useful even in mini-games such as tic-tac-toe.
- Since we have limited time available, we want to avoid unnecessary computation in the minimax tree.
- Pruning: ways of determining that certain branches will not be useful. Then cut of these branches

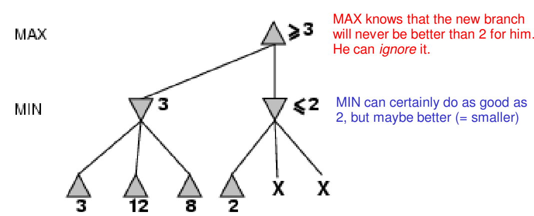
pruning



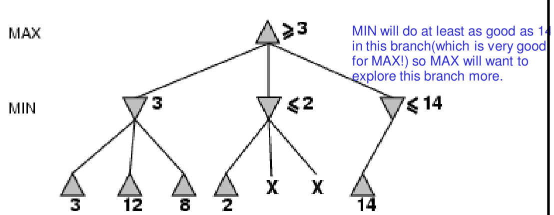
α pruning



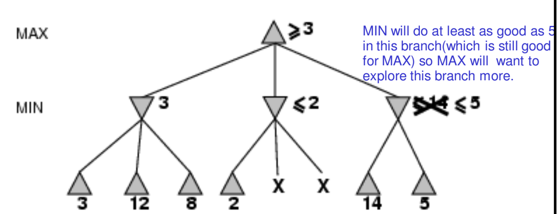
α pruning



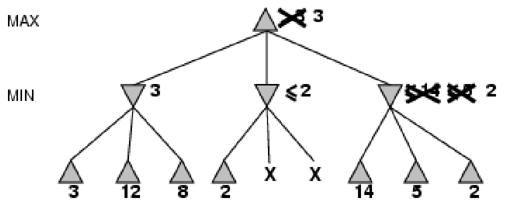
α pruning



α pruning



α pruning

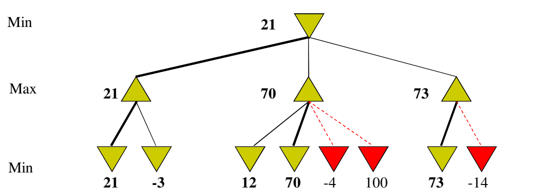


MIN will be able to play this last branch and get 2. This is worse than 3, so MAX will play 3.

β pruning

- Similar idea to α pruning, but the other way around
- If the current minimum is less than the successor's max value, don't look down that max tree any more

β pruning example



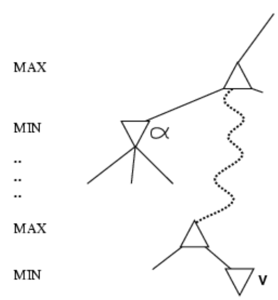
- Some subtrees at second level already have values > min from previous, so we can stop evaluating them.

Why is it called α-β?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
- If *v* is worse than α, *max* will avoid v

→ prune that branch

- Define β similarly for *min*



α-β Pruning properties

- Pruning by these cuts does not affect final result
  - May allow you to go much deeper in tree
- Properties:
  - Evaluating "best" branch first yields better likelihood of pruning later branches
  - Perfect ordering reduces time to $b^{m/2}$

Properties of minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an rational opponent)
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$ (depth-first exploration)

- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  → exact solution completely infeasible