Carleton

Catan Presentation

Bat-Orgil Batjargal, Alvin Bierley, Andrew Fitch, and Daniel Kleber

Advisor: Aaron Bauer

Department of Computer Science
Carleton College, Northfield, MN 55057

February 2021

Game Playing AI

Why AI?

- We want to be able to solve problems in varied domains
- We don’t know all the domains
- Heuristic algorithms need writing
- Alternative: general-purpose AI models
- Need to figure out ideal model structure

AI Classification Problems

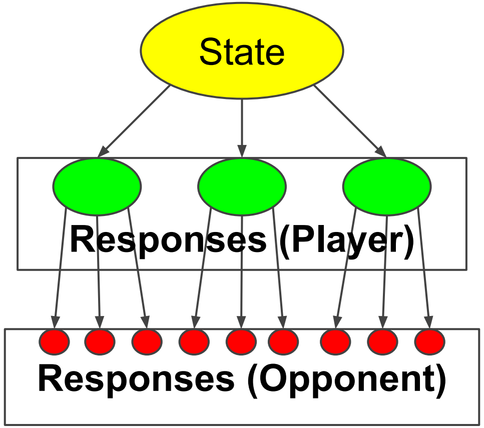
- Many simple AI are run on classification problems
- Features:
  - Relatively consistent
  - Few secrets
  - Deterministic outcome
  - Risk of overfitting
- Not necessarily best match for the real world

Carleton

Game Playing

- Games are efficient to run
- Often dense in imperfect information
- Often dense in random outcomes
- Less consistency means models learn differently
- Possibly better match for reality

Types of Game AI

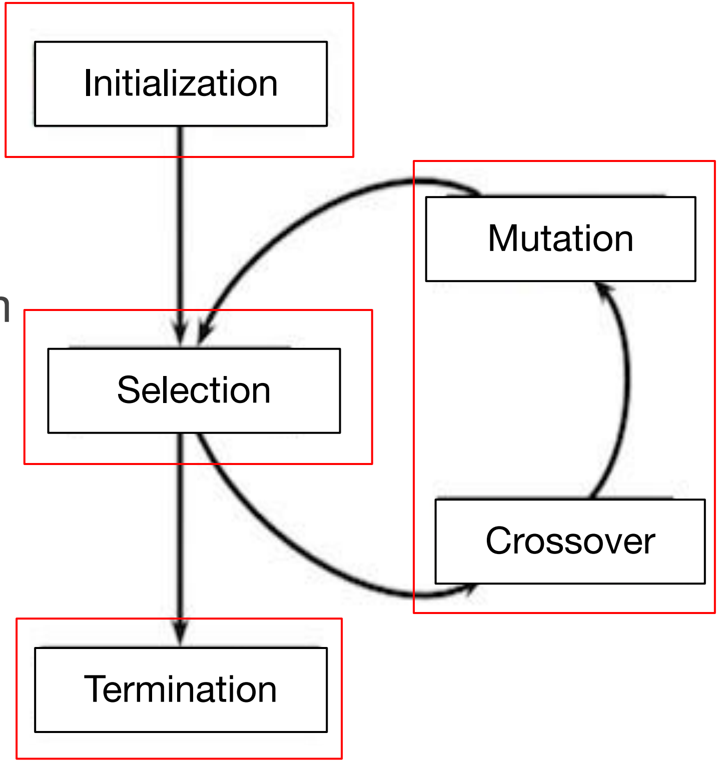


- Many strategies for how to play games
- Solve the game: Unlikely to work
- Assign scores to states
  - Minimax: move to maximize the minimum value
  - AB Pruning: Extension of minimax which cuts off subpar moves rather than exploring them
- Monte Carlo Tree Search
  - Full game rollouts
- Neural Nets
- Evolutionary Algorithms

Evolutionary AI

Overview

- Family of algorithms for global optimization
- Candidate solutions for an optimization problem
  - Fitness function determines quality of solutions
  - Evolution of candidate solutions
- Inspired by biological evolution
  - Reproduction, mutation, crossover, selection
  - Survival of the fittest

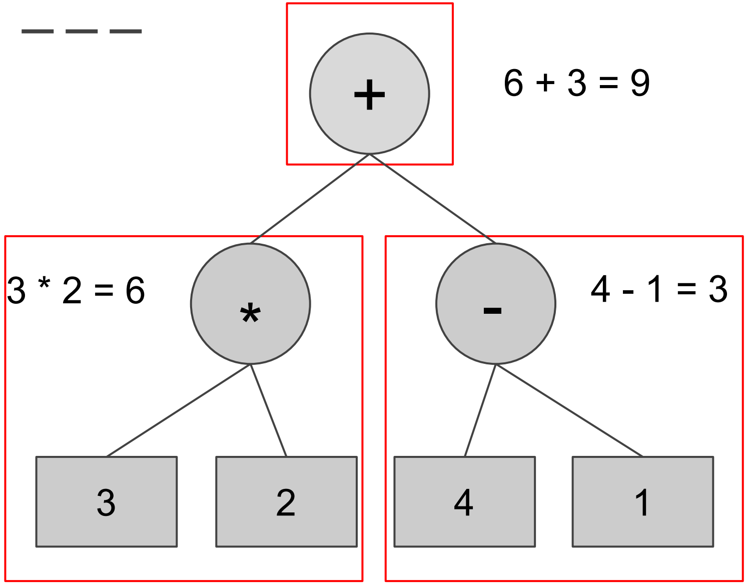


Examples

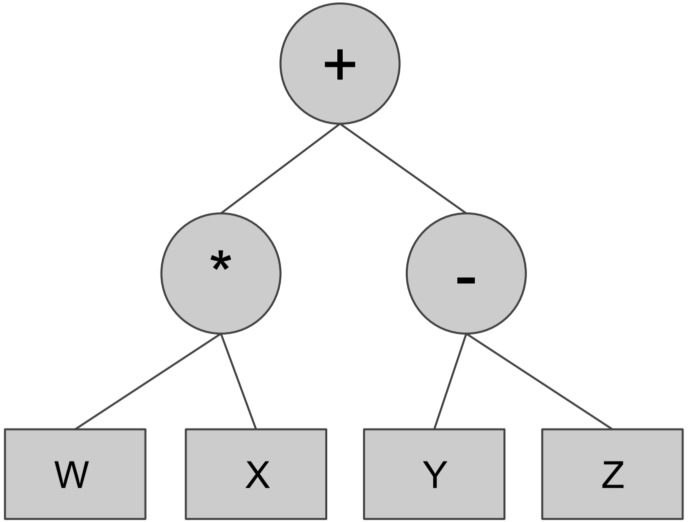
- Genetic algorithms
  - Most popular
  - Solutions are represented as a string of numbers
- Genetic programming
  - Solutions are computer programs
  - Fitness determined by ability to solve a computational problem
- Evolutionary programming
  - Structure of program is fixed, numerical parameters evolve

Genetic Programming

Genetic programming uses parse trees



(3 * 2) + (4 - 1)

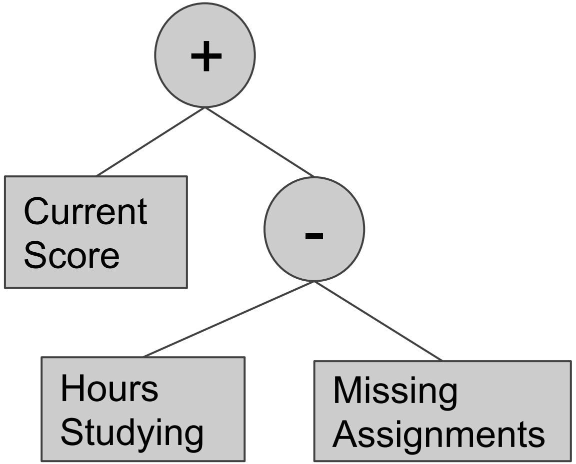


(W * X) + (Y - Z)

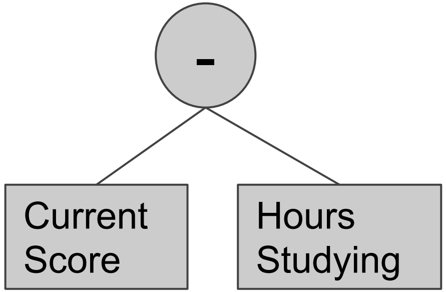
Genetic programming estimates some value

Predict a student's exam score using:

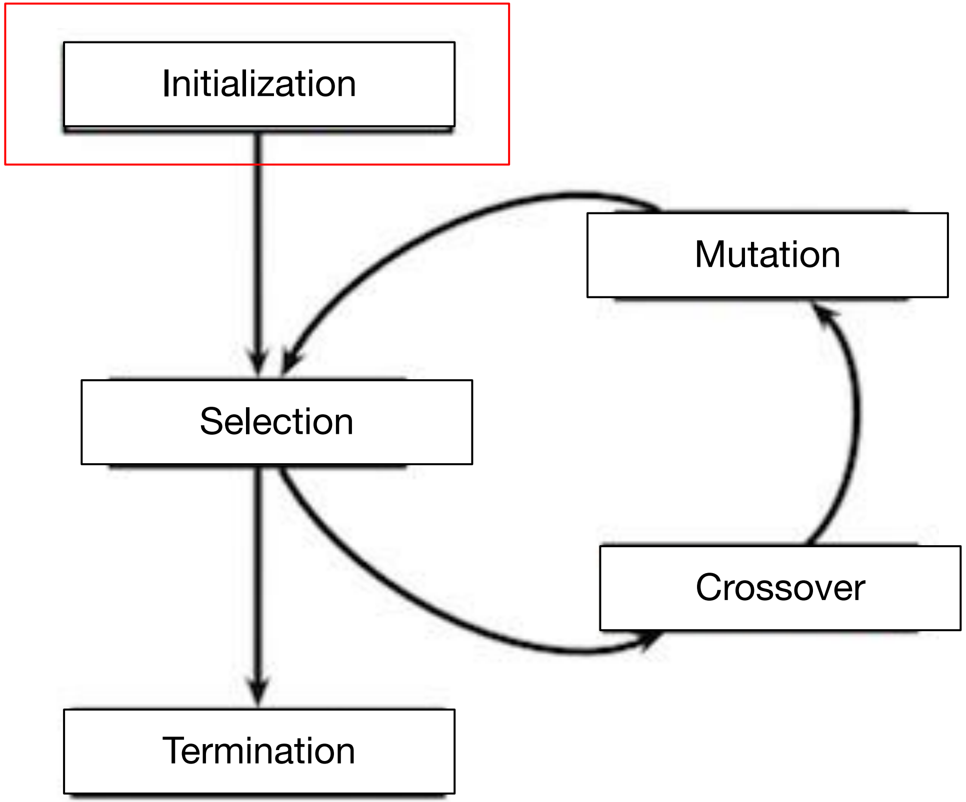
- Current score in class
- Number of hours spent studying
- Number of missing assignments



Current Score +
(Hours Studying - Missing Assignments)



Current Score - Hours Studying


Initialization
Mutation
Selection
Crossover
Termination

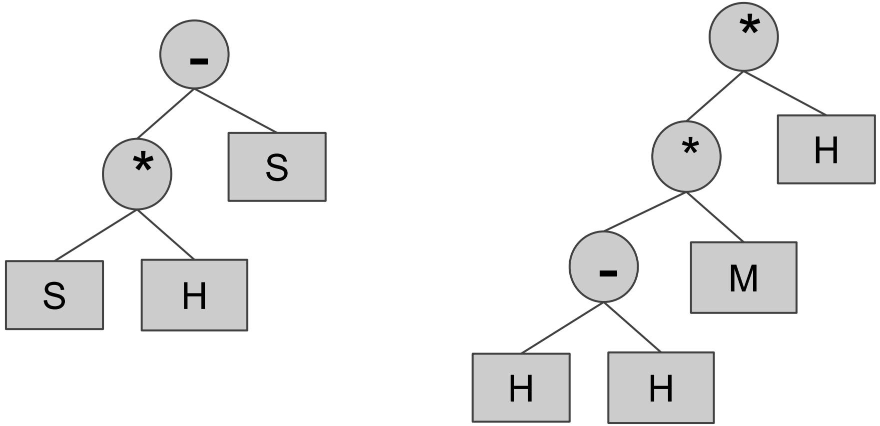
Randomly initialize parse trees using inputs and operators

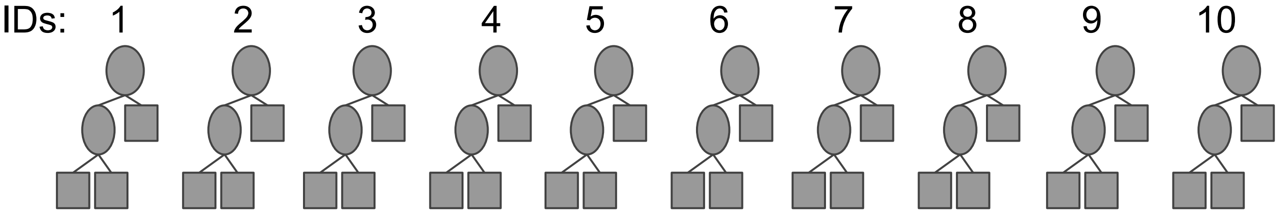
Inputs:

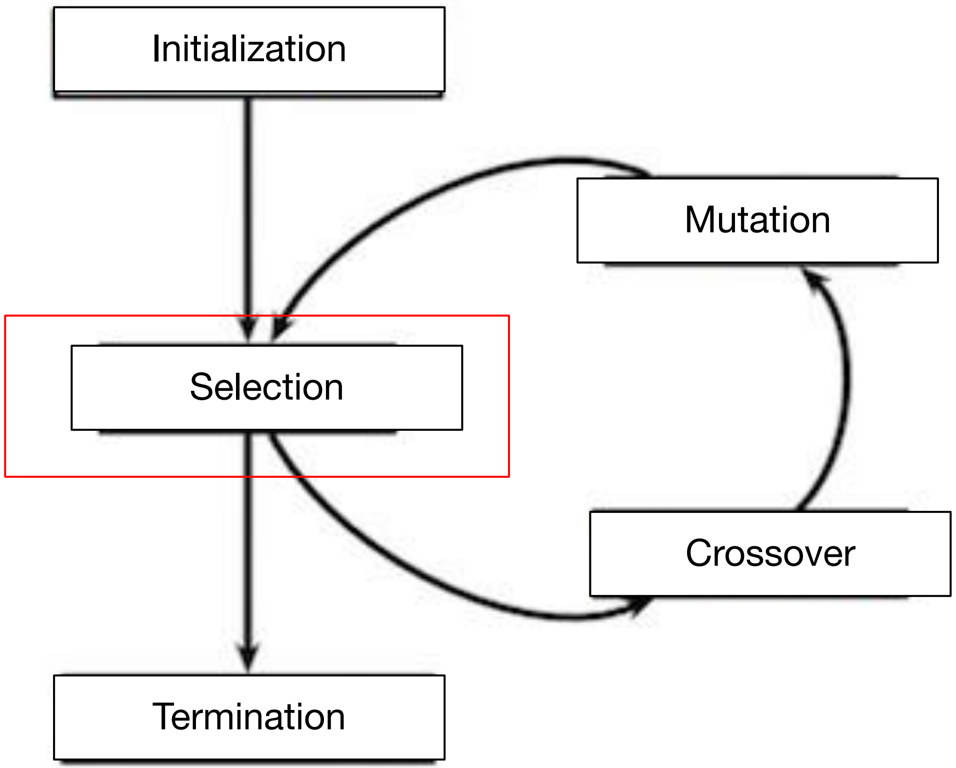
- Current score in class (S)
- Number of hours spent studying (H)
- Number of missing assignments (M)

Operators:

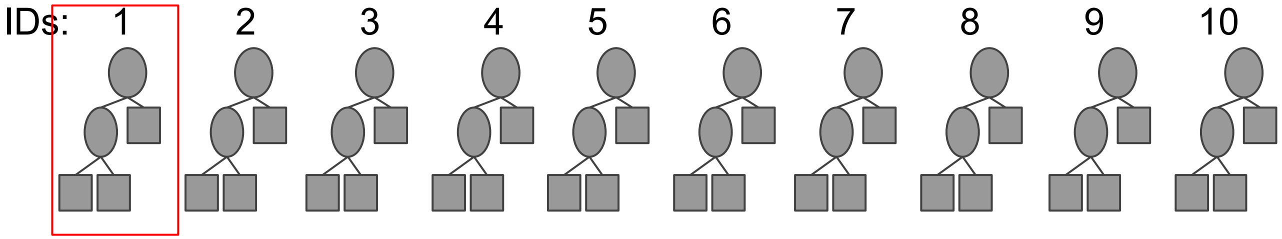
- +, -, *






Initialization
Mutation
Selection
Crossover
Termination

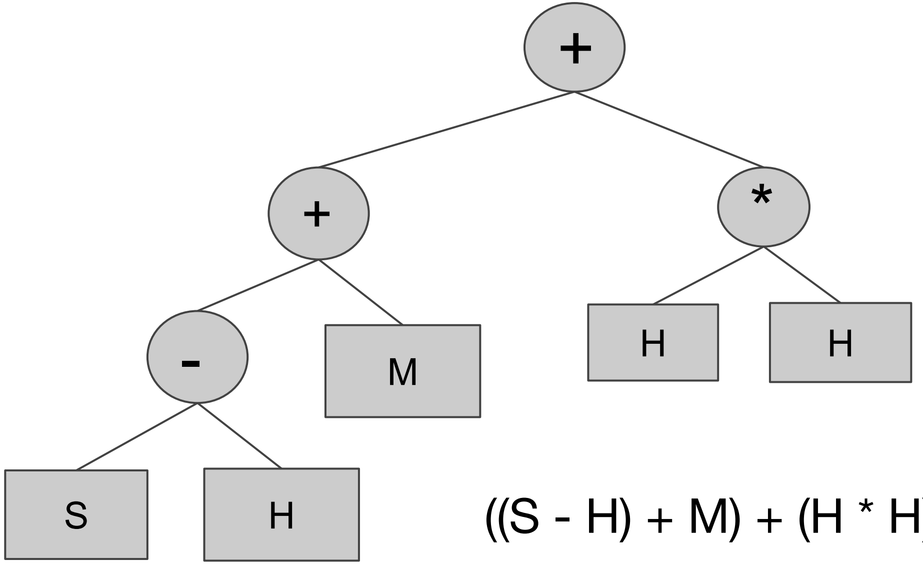
Each tree is evaluated and assigned a score



Score = $(|\text{Predicted Result} - \text{Actual Result}|)^{-1}$

Scoring a parse tree

Current score in class = S
Number of hours spent studying = H
Number of missing assignments = M



$((S - H) + M) + (H * H)$

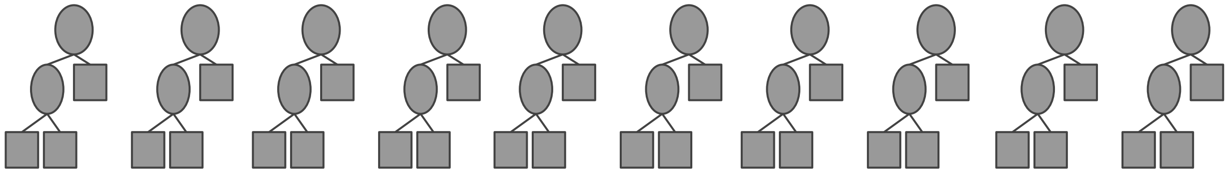
Actual Test Score = 94
H = 6
M = 1
S = 90

Predicted Test Score: $((90 - 6) + 1) + (6 * 6) = 121$

Final Tree Score: $(| 121 - 94 |)^{-1} = 0.037$

Each tree is evaluated and assigned a score

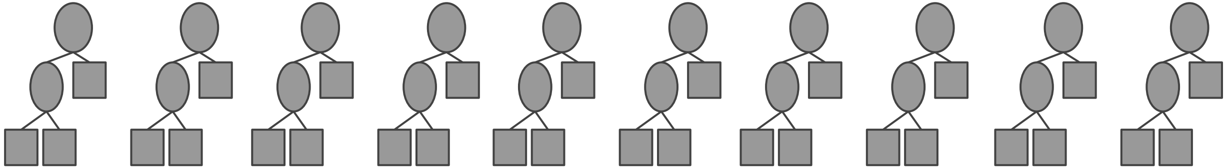
IDs: 1 2 3 4 5 6 7 8 9 10



Scores: .037 .05 .015 .013 .01 .012 .041 0.11 .027 0.002

Trees are weighted by their scores and then randomly selected for repopulation

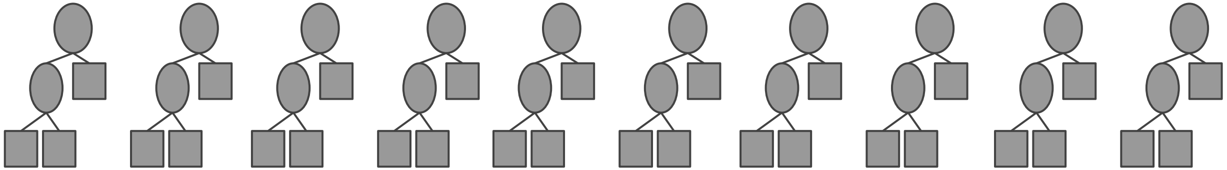
IDs:	1	2	3	4	5	6	7	8	9	10

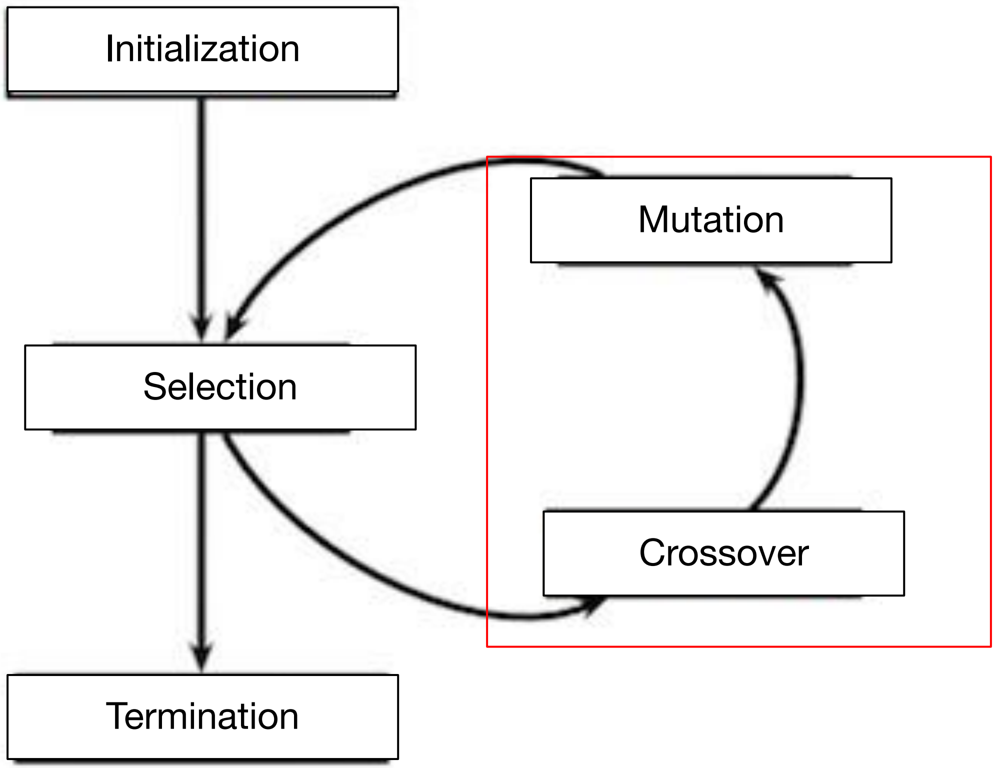


Scores:	.037	.05	.015	.013	.01	.012	.041	0.11	.027	0.002
Weights:	0.12	0.16	0.05	0.04	0.03	0.04	0.13	0.35	0.09	0.01

IDs:	2	8	8	2	1	8	7	8	9	8

Selected Trees:




Initialization
Selection
Termination
Mutation
Crossover

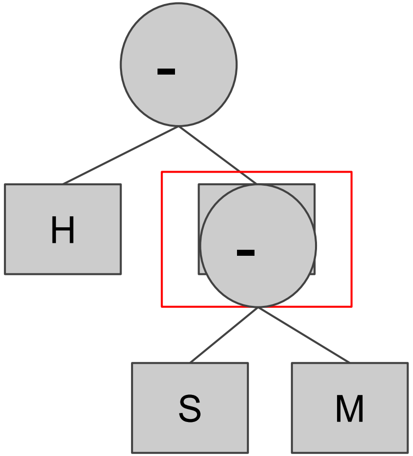
Crossover swaps two nodes and their subtrees

Current score in class = S

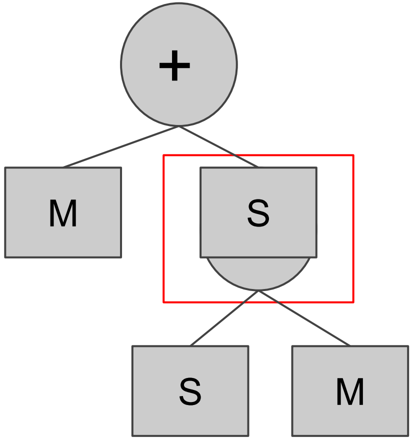
Number of hours spent studying = H

Number of missing assignments = M

ID: 2

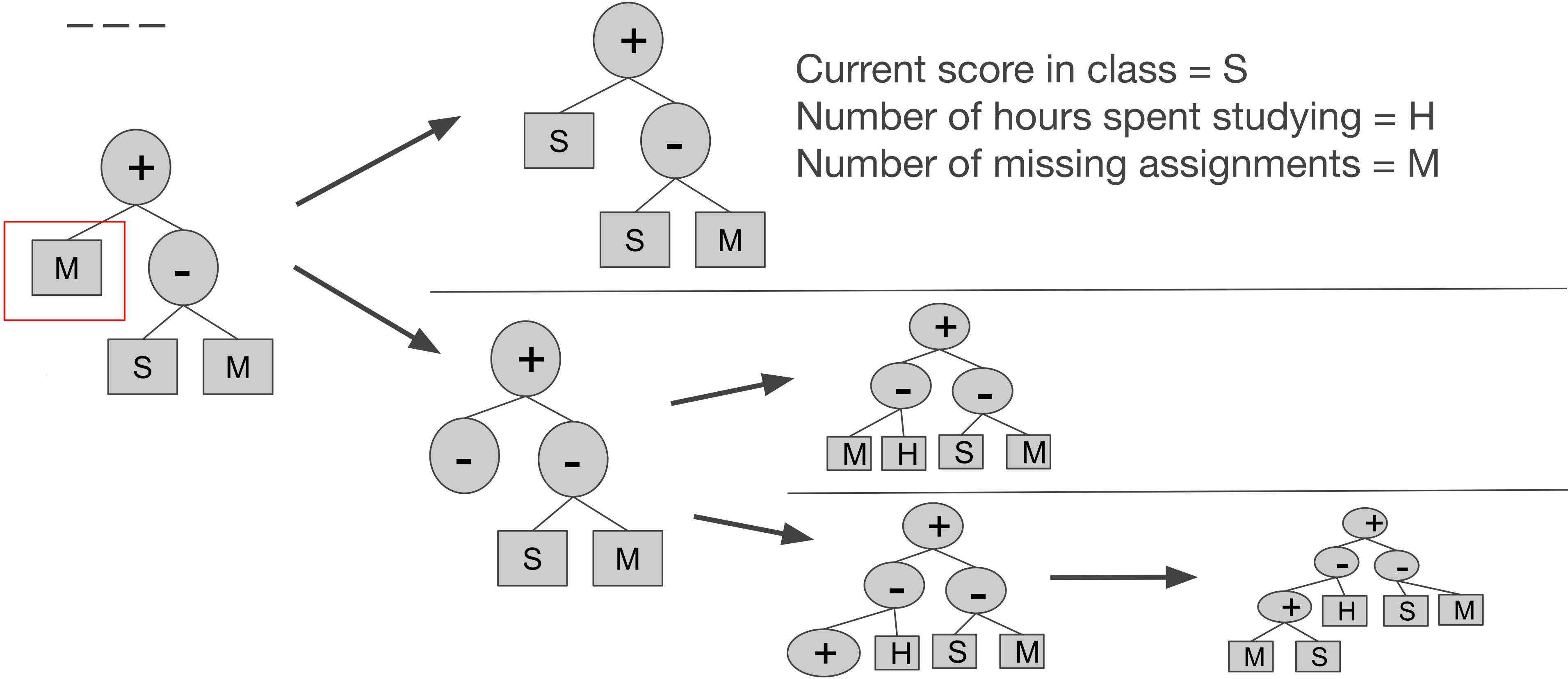


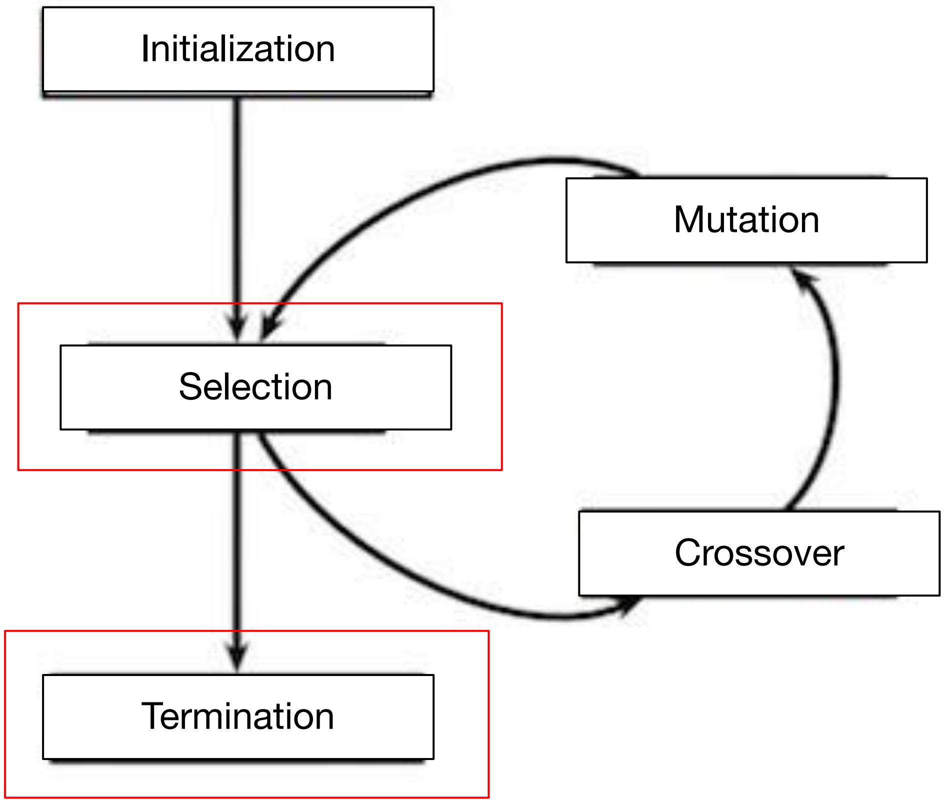
ID: 8



Mutation randomly changes a single node

Current score in class = S
Number of hours spent studying = H
Number of missing assignments = M




Initialization
Mutation
Selection
Crossover
Termination

Settlers of Catan

Players gain resources to build things

- Resource gathering strategy game (2-6 players)
- Terrain tiles produce resources
- Ports by water tiles
- Dice rolls earn resources
    - Wood, ore, brick, wheat, sheep
- Resources build settlements, cities, and roads
    - Can also obtain development cards

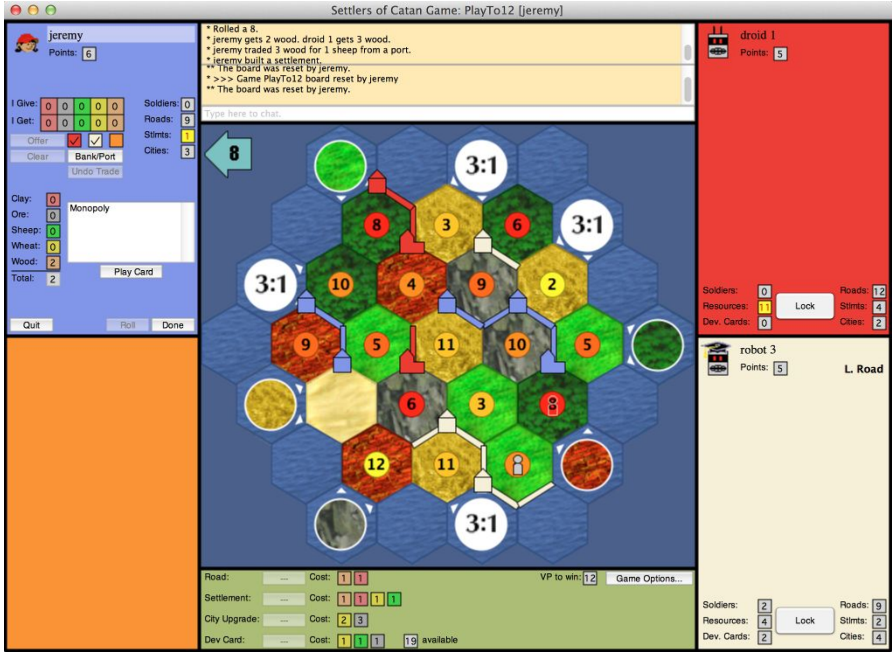
Goal of the game is to win 10 victory points

- Three ways to earn victory points:
  1. Building settlements (1 victory point) and cities (2 victory points)
  2. Largest army or longest road (2 victory points each)
  3. Victory point development card (1 victory point)

Jsettlers + our strategy

Carleton

JSettlers: Open Source Java Settlers of Catan

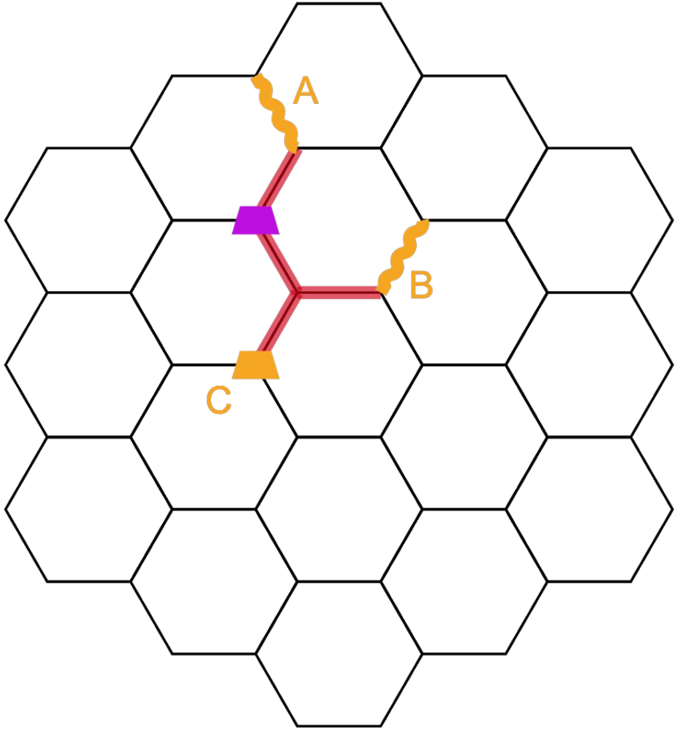


Screenshot of JSettlers Game between Jeremy and two bots

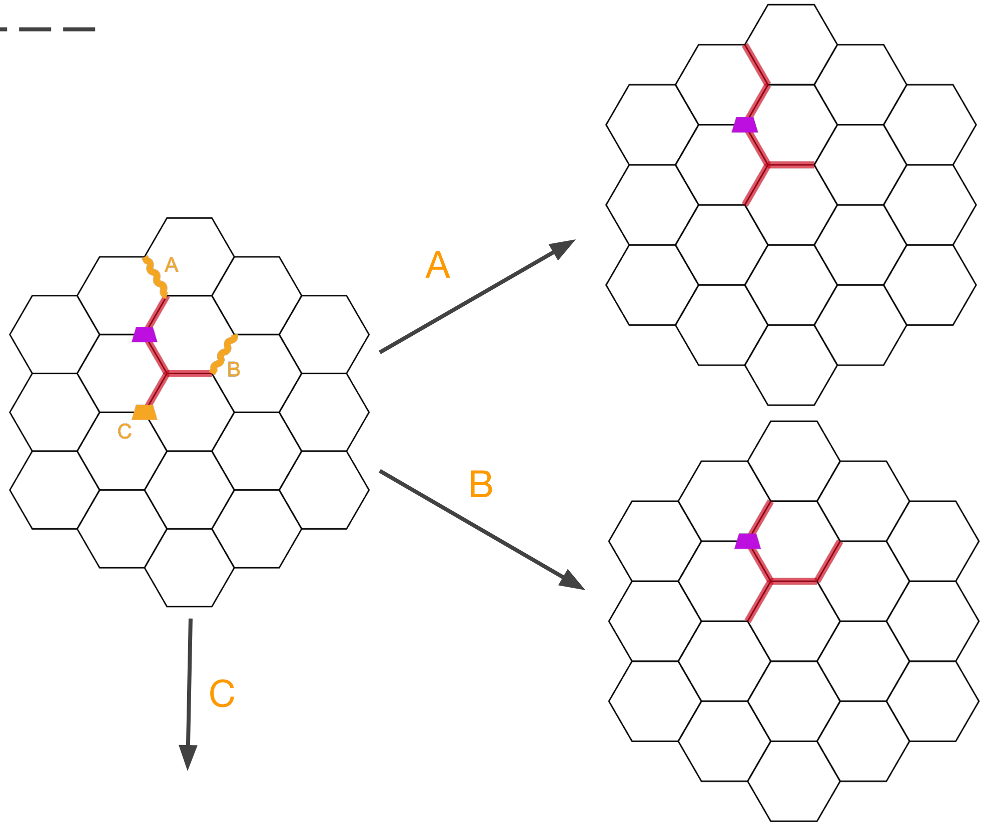
Types of game-playing bots:

1. Fast (simple heuristics)
2. Smart (more intelligent heuristics)

Smart bot simulates simplified games to find Win-Game-ETA



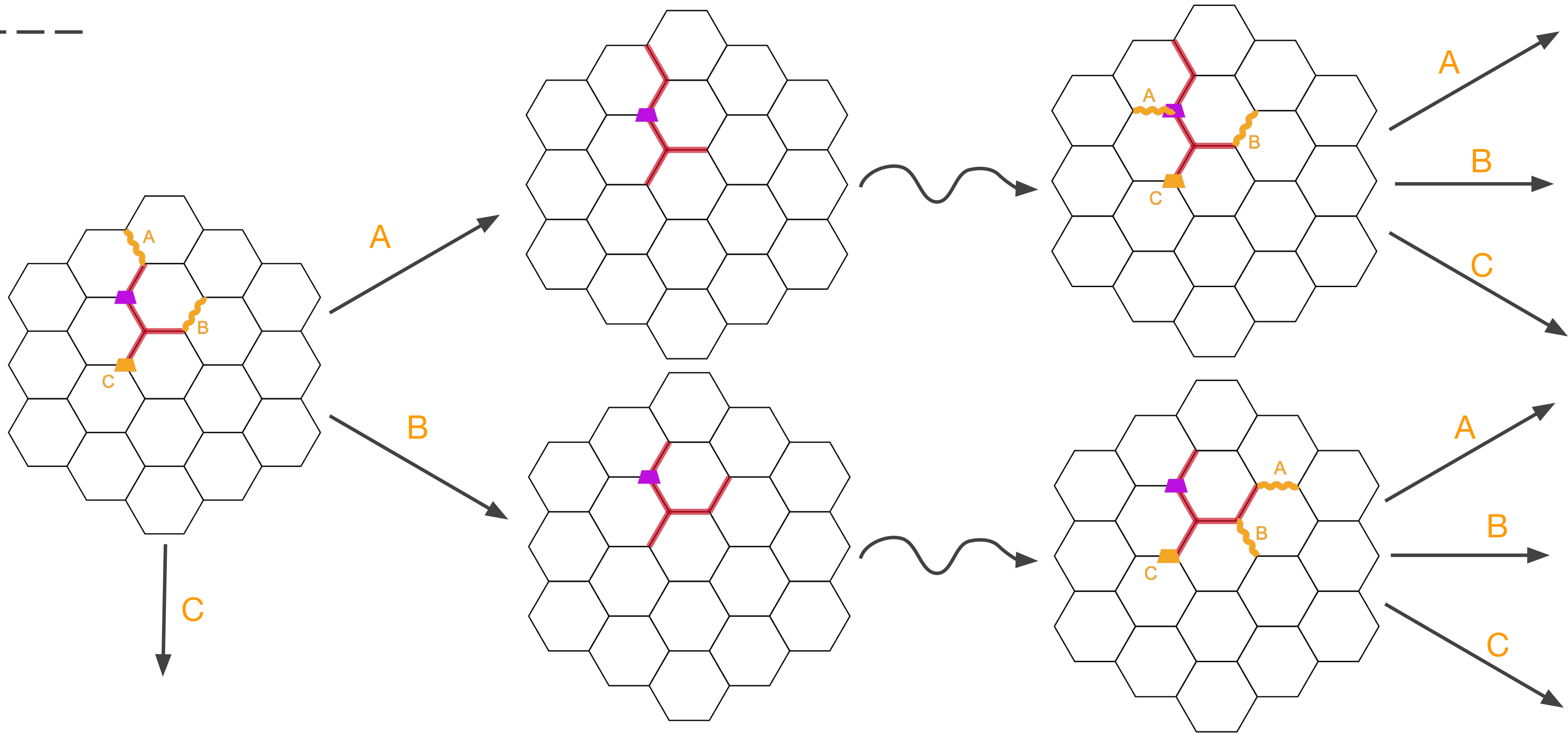
Each simulation considers 6 scenarios with while loop



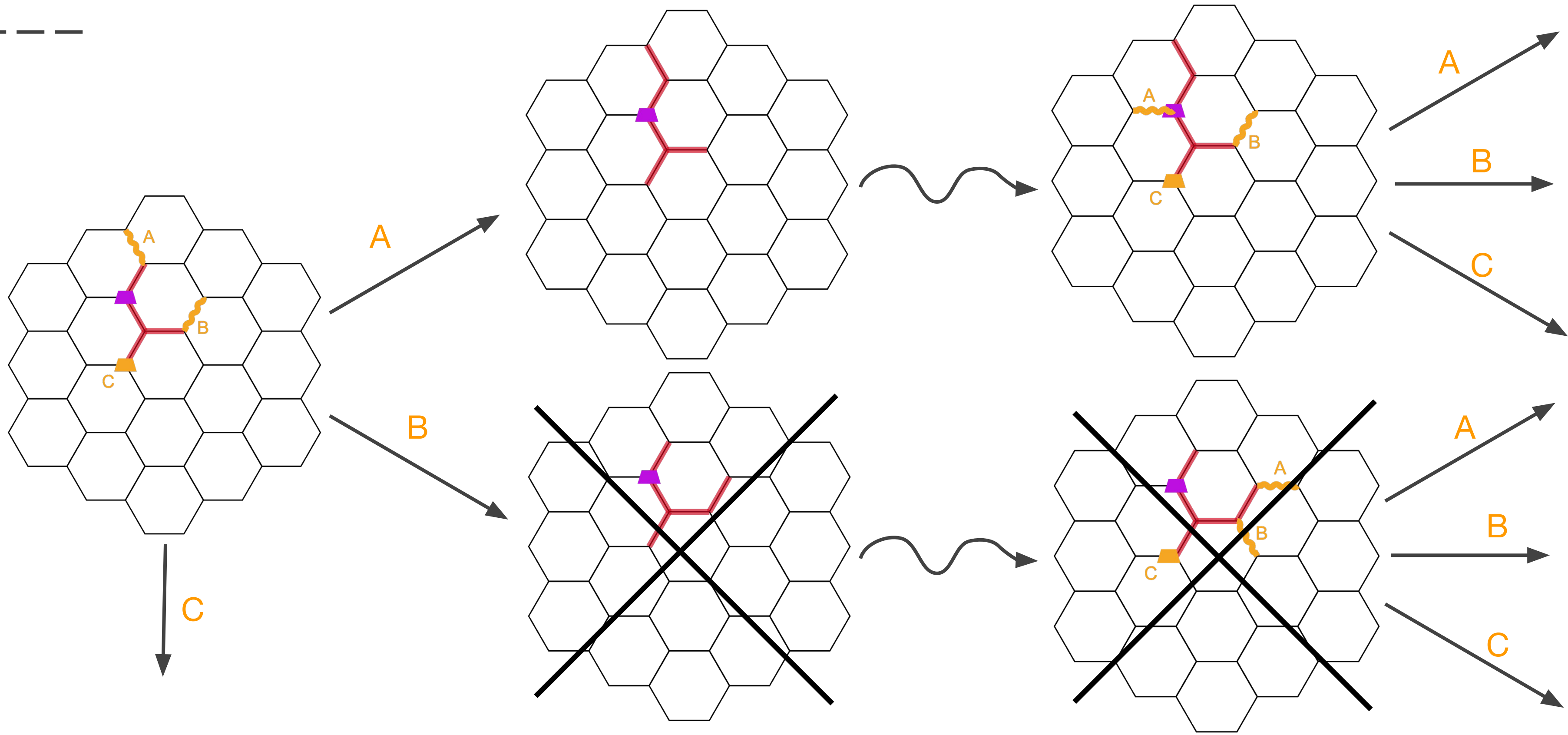
1. 2 settlements (including necessary roads' ETA)
2. 2 cities
3. 1 city, 1 settlement (+ roads)
4. 1 settlement (+ roads), 1 city
5. Buy enough cards for Largest Army
6. Build enough roads for Longest Road

Carleton

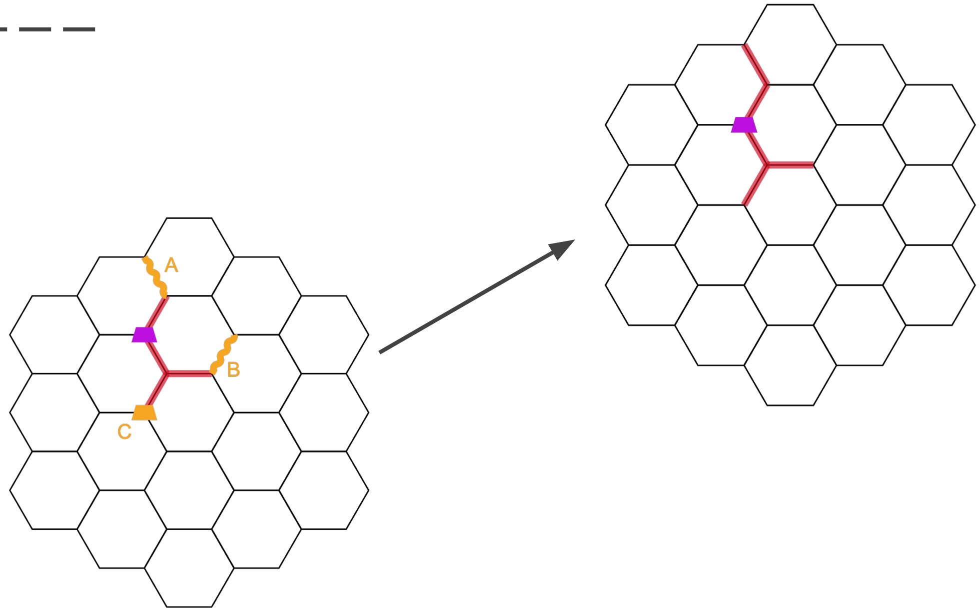
Tree-like simplified-game simulation



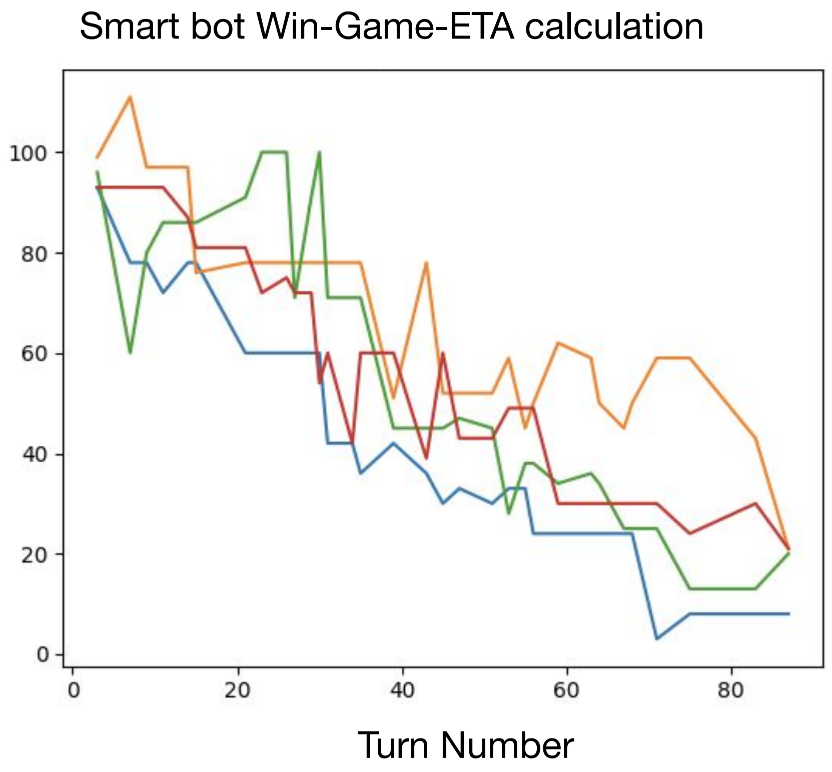
Pick strategy with the best Win-Game-ETA



Pick strategy with the best Win-Game-ETA

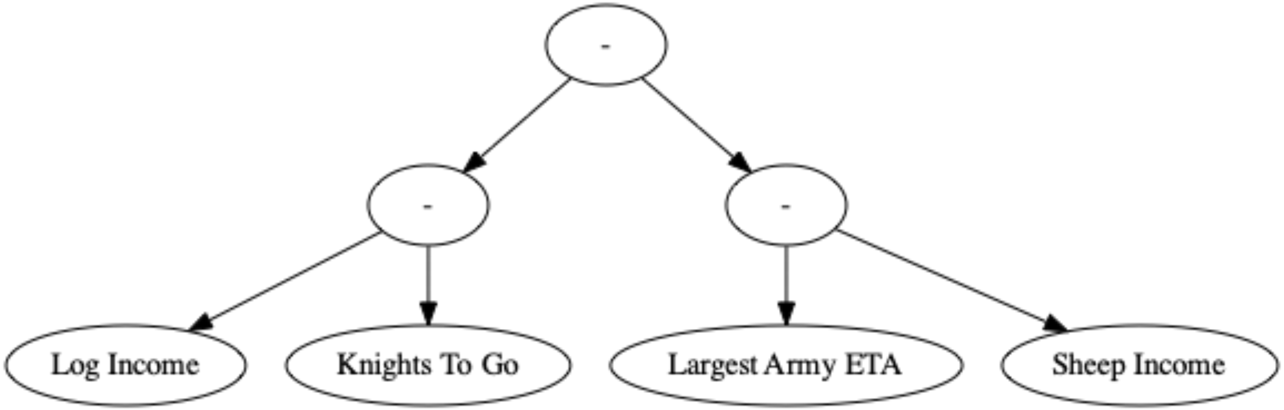


Win-Game-ETA calculation is primary in Smart bot decision



Carleton

Evolutionary AI with genetic programming



$$WinGameETA = [\,(Log\ Income - Knights\ To\ Go) - (Largest\ Army\ ETA - Sheep\ Income)\,]$$

Training

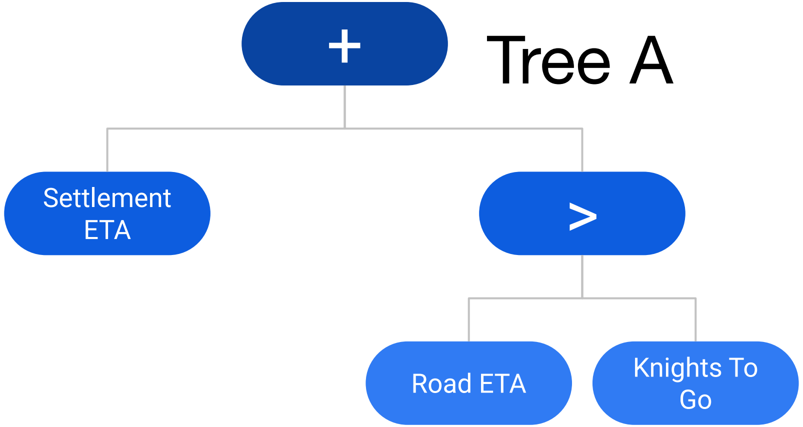
Training Procedures

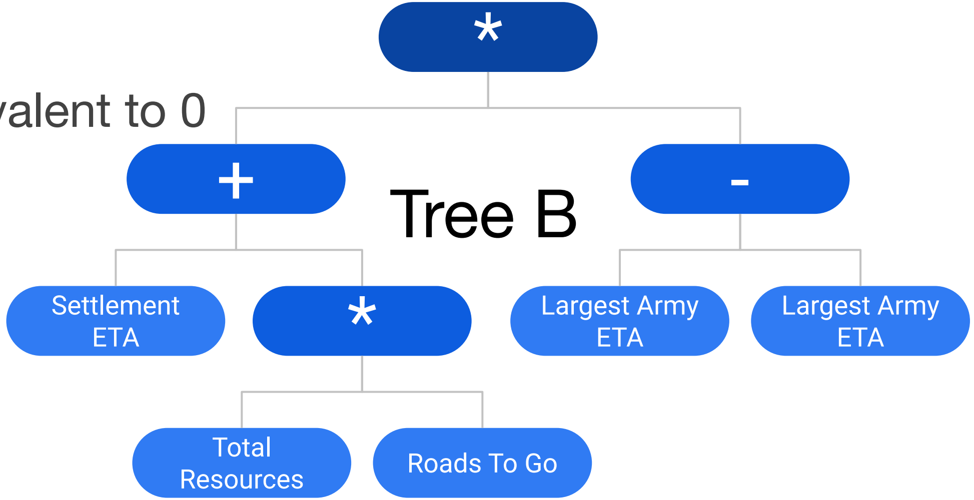
- Trainer function in Python, calling Java games
- Play games with each tree in a generation
- Score based on game scores
- Trained on Carleton CS servers
- Ran at least 1,000,000 games of Catan
- About 40 experiments with various settings for trainer

Carleton

Initial Results

- First attempts to train had issues
    - Could look at the trees to solve them
- Many possible operations
- Operations where they shouldn’t be: “<” as a root node
- Failure to use constant factors
- Large trees with messy bits equivalent to 0

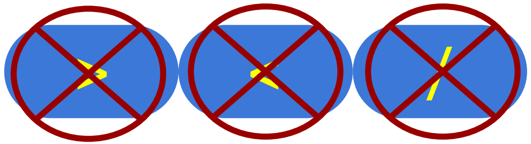




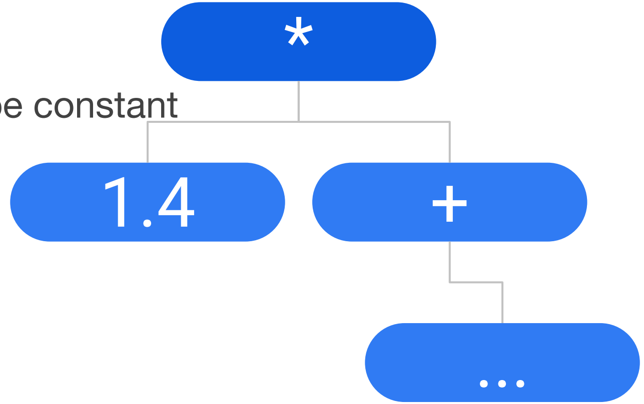
Carleton

Solutions

- Poor root node choices
  - Removed operations
  - Cut division and boolean comparatives



- Lack of constants
  - Mandated right leaves of multiplication operators to be constant
  - Created training settings for tuning constants



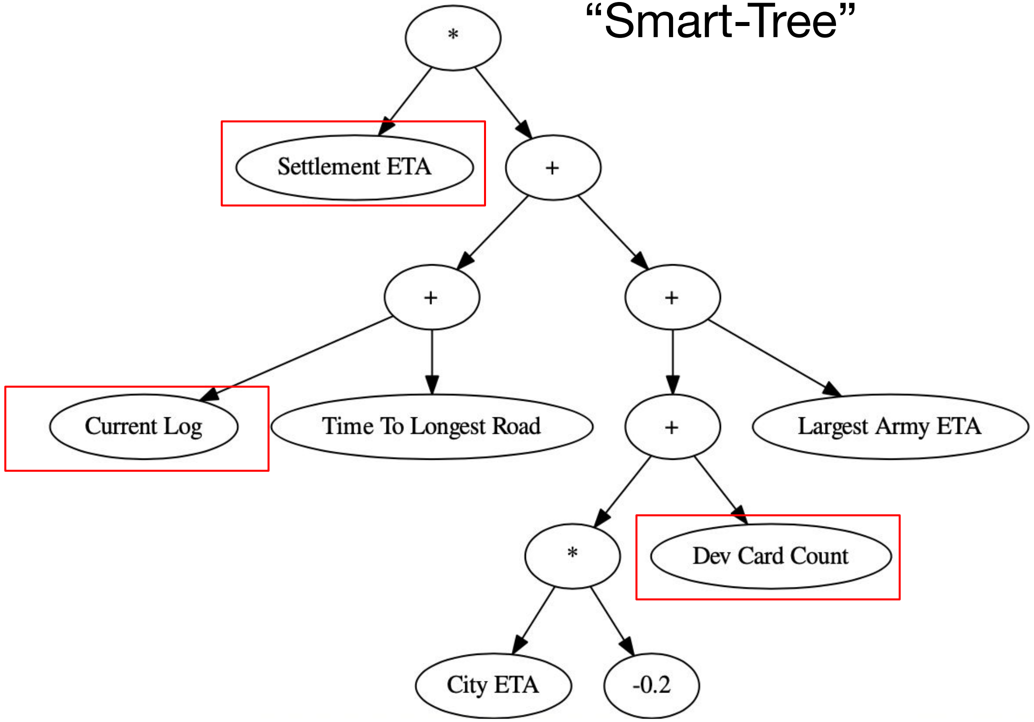
- Messy trees
  - Added fitness penalty for each node

Results

Best Training Strategy for Playing Smart Bots

- Only used mutation
- 60 Generations with max mutation parameters
- 40 generations of input training
- Trained only against smart bots

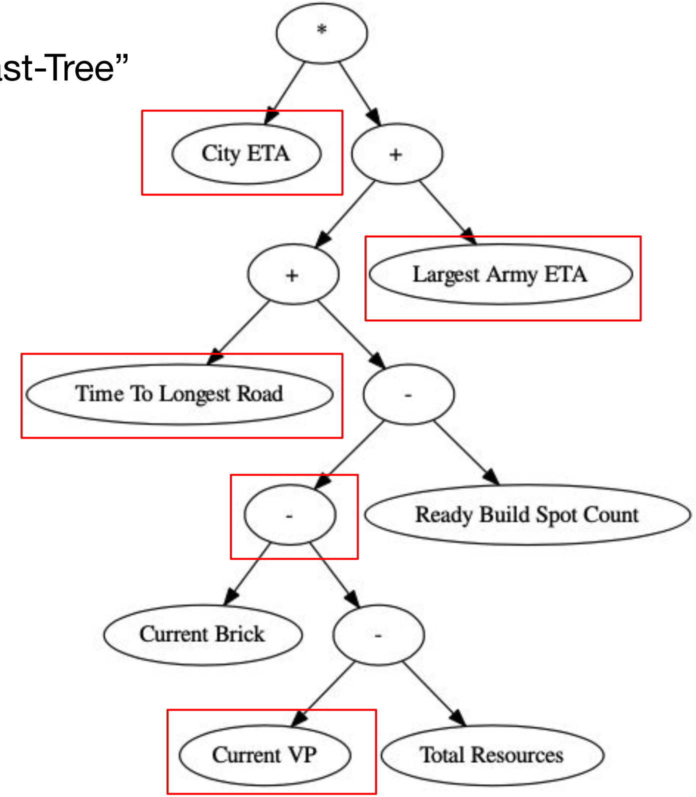
“Smart-Tree”



Strategy for Training Fast Bots is Similar

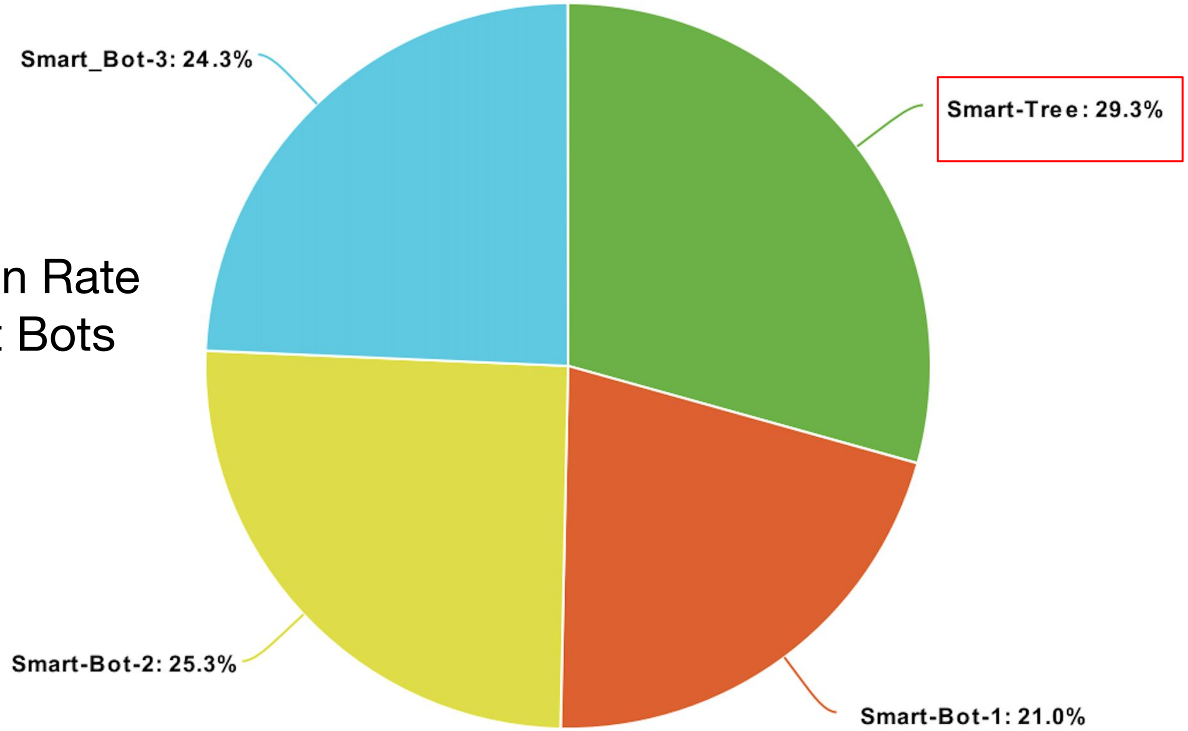
- Initialized fast bot trees from a collection of the best trees of other experiments
- Trained only on fast bots

“Fast-Tree”



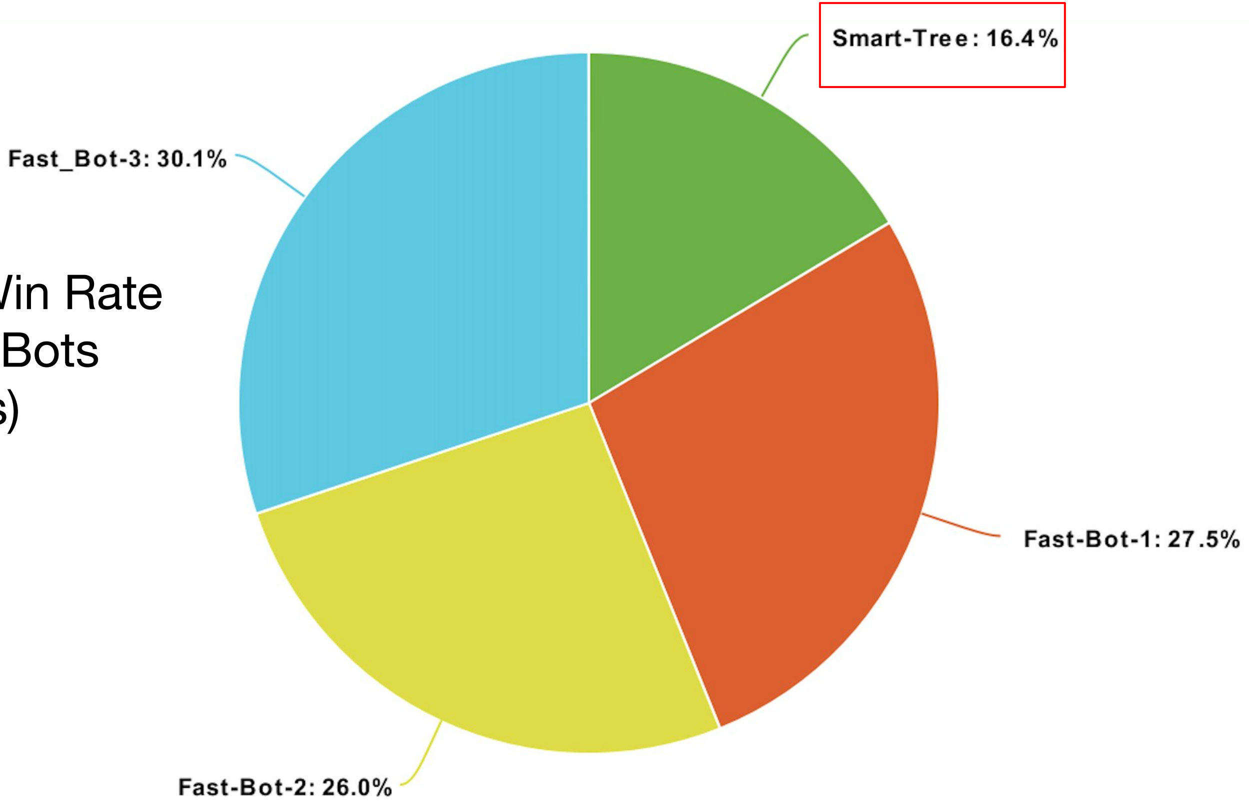
The Smart-Tree does well against smart bots

Smart-Tree Win Rate
Against Smart Bots
(300 Games)

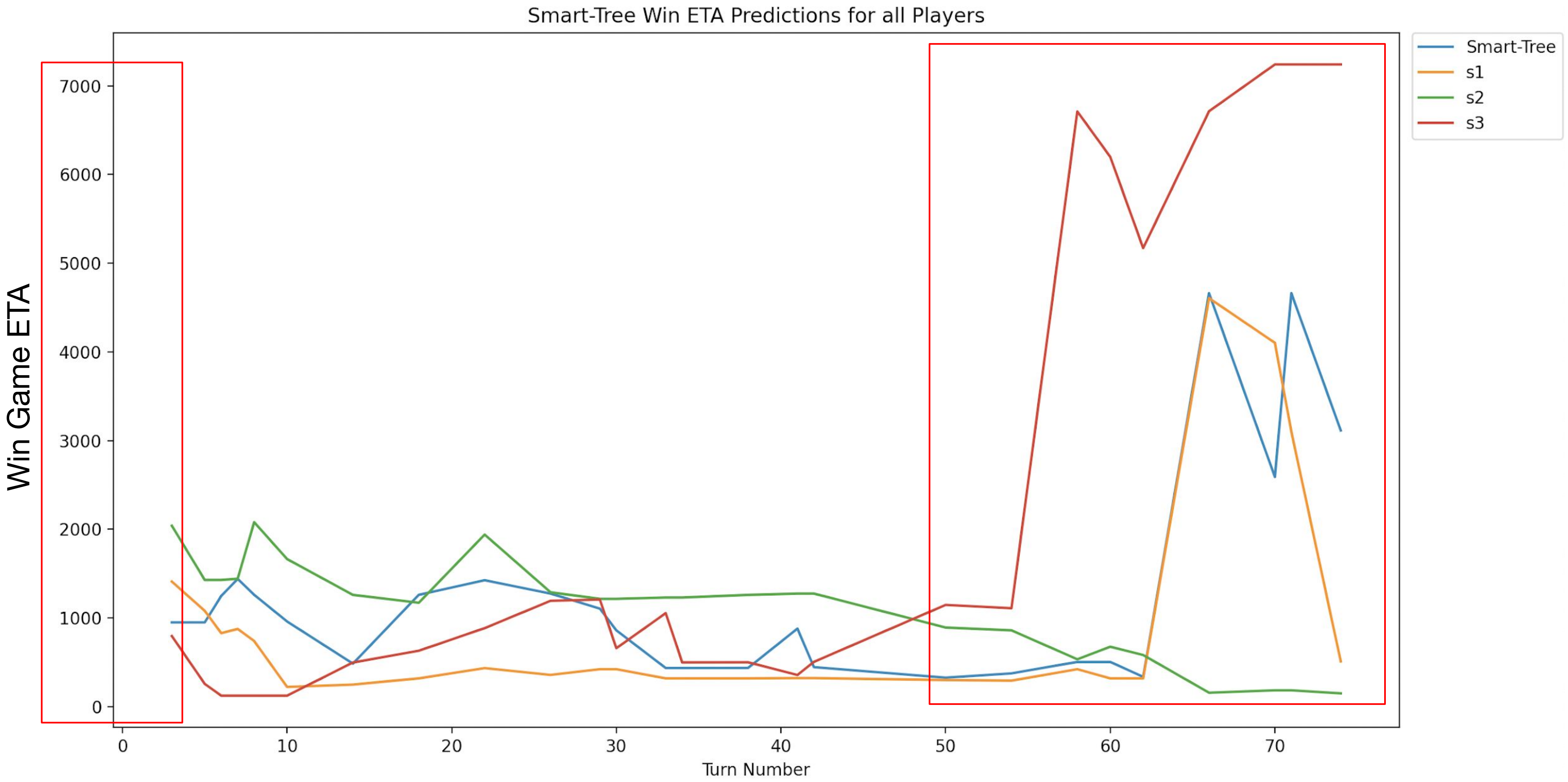


But the Smart-Tree struggles against fast bots

Smart-Tree Win Rate
Against Fast Bots
(1000 Games)



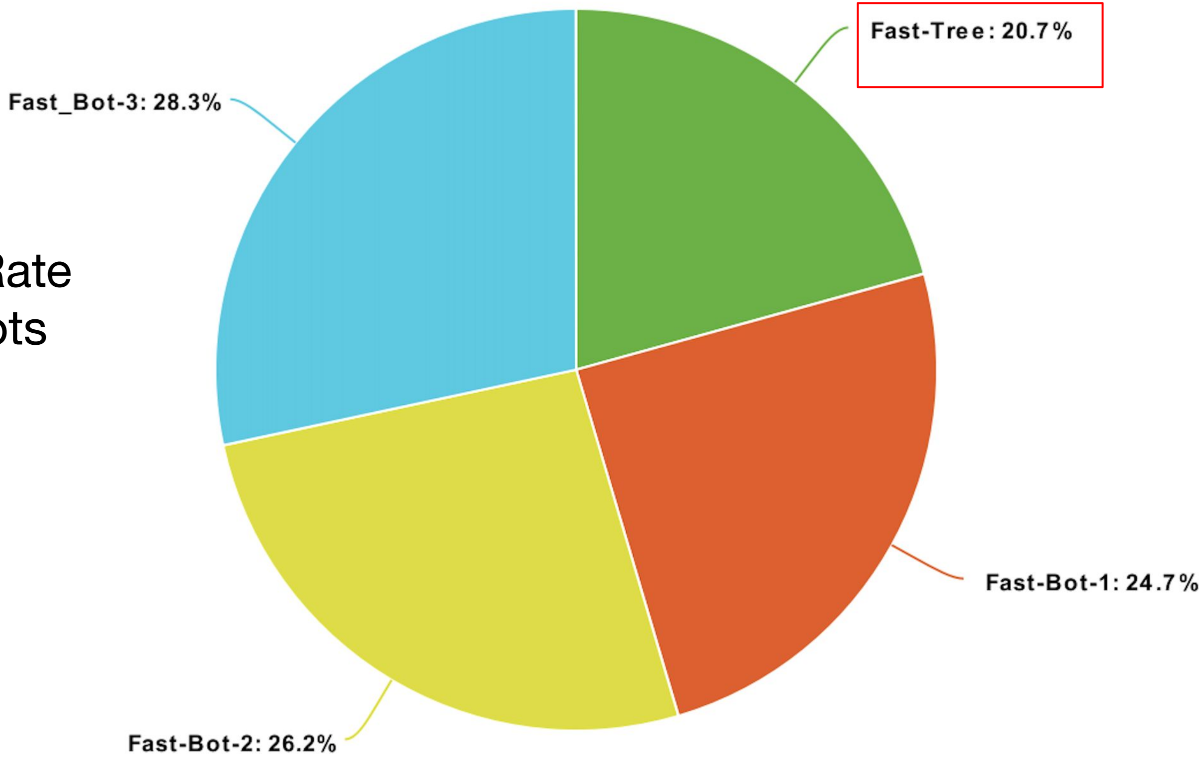
Smart-Tree Win-Game-ETA Predictions for All Players



Carleton

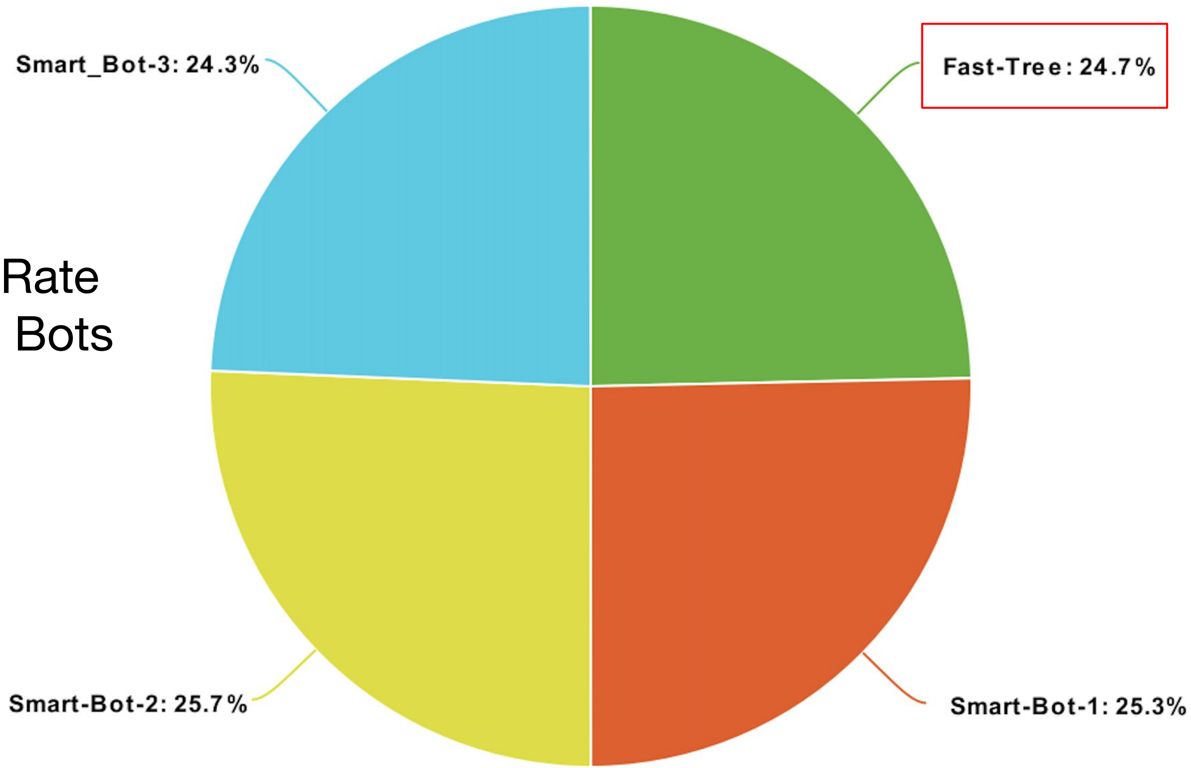
The Fast-Tree performs poorly against fast bots

Fast-Tree Win Rate Against Fast Bots (1000 Games)



But the Fast-Tree is slightly better against smart bots

Fast-Tree Win Rate
Against Smart Bots
(300 Games)



To conclude

Plan:

- Improve Smart Bot’s WinGameETA estimations

To conclude

Plan:

- Improve Smart Bot’s WinGameETA estimations

Expectations:

- Faster than tree simulation
- New approach
- Easier to interpret

To conclude

Plan:

- Improve Smart Bot’s WinGameETA estimations

Expectations:

- Faster than tree simulation
- New approach
- Easier to interpret

Results:

- As good as or slightly better than smart bot
- Not good as fast bot’s performance

To conclude

Plan:

- Improve Smart Bot's WinGameETA estimations

Expectations:

- Faster than tree simulation
- New approach
- Easier to interpret

Results:

- As good as or slightly better than smart bot
- Not good as fast bot's performance

Future:

- Build full scale decision maker

Citations

de Mesentier Silva, F., Lee, S., Togelius, J., & Nealen, A. (2017). AI-based playtesting of contemporary board games. In Proceedings of the 12th International Conference on the Foundations of Digital Games (pp. 1-10). [pdf]

Eger, M., & Martens, C. (2019, October). A Study of AI Agent Commitment in One Night Ultimate Werewolf with Human Players. In Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (Vol. 15, No. 1, pp. 139-145). [pdf]

Szita, I., Chaslot, G., & Spronck, P. (2009, May). Monte-carlo tree search in settlers of catan. In Advances in Computer Games (pp. 21-32). Springer, Berlin, Heidelberg. [pdf]

Woolford, M., & Watson, I. (2017, June). SCOUT: a case-based reasoning agent for playing race for the galaxy. In International Conference on Case-Based Reasoning (pp. 390-402). Springer, Cham. [link]

Thank you!