Multi-Objective Monte Carlo Tree Search for Real-Time Games

Diego Perez, Sanaz Mostaghim, Spyridon Samothrakis, and Simon M. Lucas
IEEE Transactions on Computational Intelligence and AI in Games (2014)

Presented by Drew Buck

9/22/2015

Outline

- Games Research
  - Reinforcement Learning (RL)
- Monte Carlo Tree Search (MCTS)
  - Example
- Multi-Objective Optimization
- Multi-Objective Monte Carlo Tree Search (MO-MCTS)
- Examples
  - Deep Sea Treasure (DST)
  - Multi-Objective Physical Traveling Salesman Problem (MO-PTSP)
- Conclusions

Games Research

Why study games?

- Games provide a flexible, abstract domain to test decision-making strategies.
- Games can be made to model real-world problems.
- Most games are too hard to solve with brute-force search.

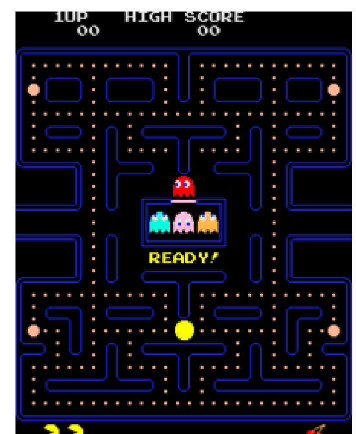


Image credit: http://www.wikipedia.org

Defining a Game

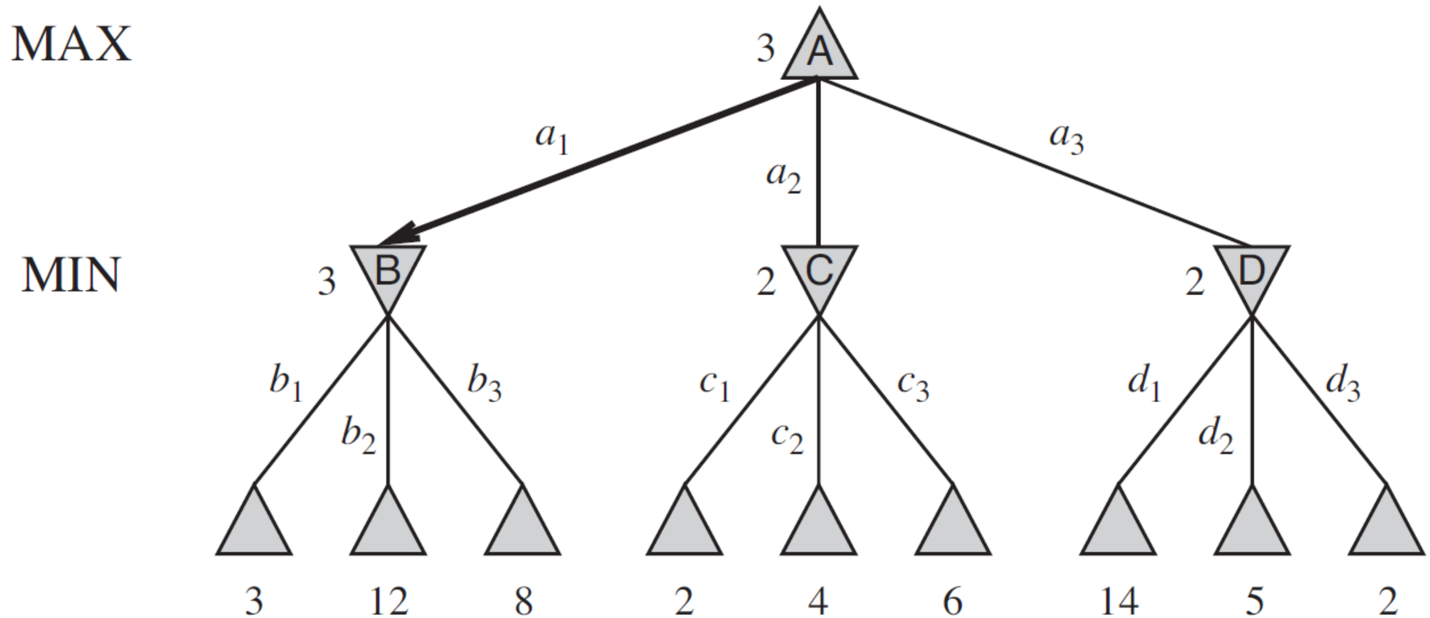
The general form of a game is a Markov decision process (MDP).

Define:

- S_0 : The initial state of the game
- $PLAYER(s)$: Which player has the move in a state
- $ACTIONS(s)$: The set of legal moves in a state
- $RESULT(s, a)$: Returns the outcome of a move
- $TERMINAL{-}TEST(s)$: True if the game is over
- $UTILITY(s, p)$: Gives the value of a state for a player

Minimax Strategy

Example:



The optimal move for "MAX" is a_1 because it maximizes the worst-case outcome.

Image credit: S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach, 3rd ed."

Tic-Tac-Toe Example

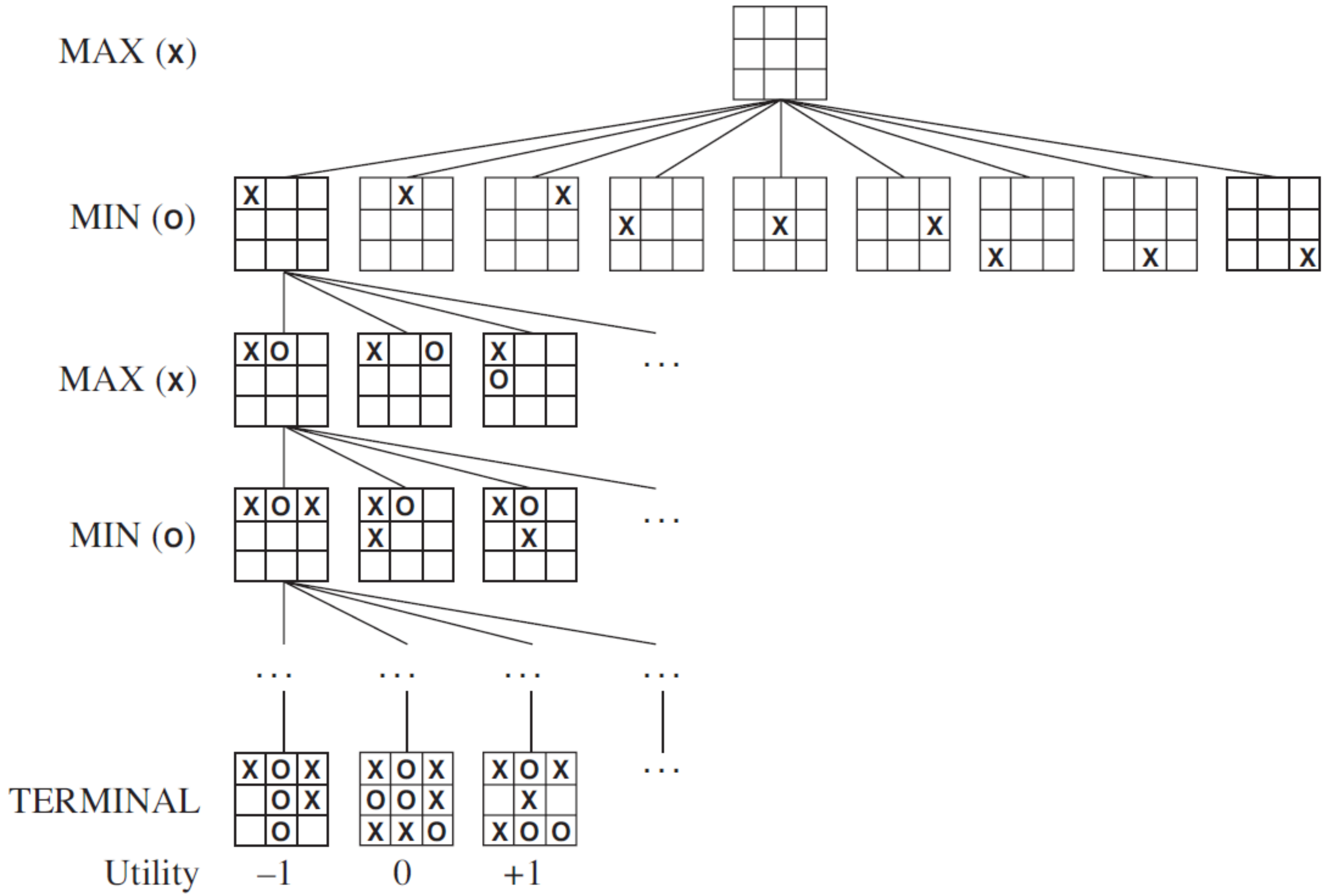


Image credit: S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach, 3rd ed."

Tic-Tac-Toe Example

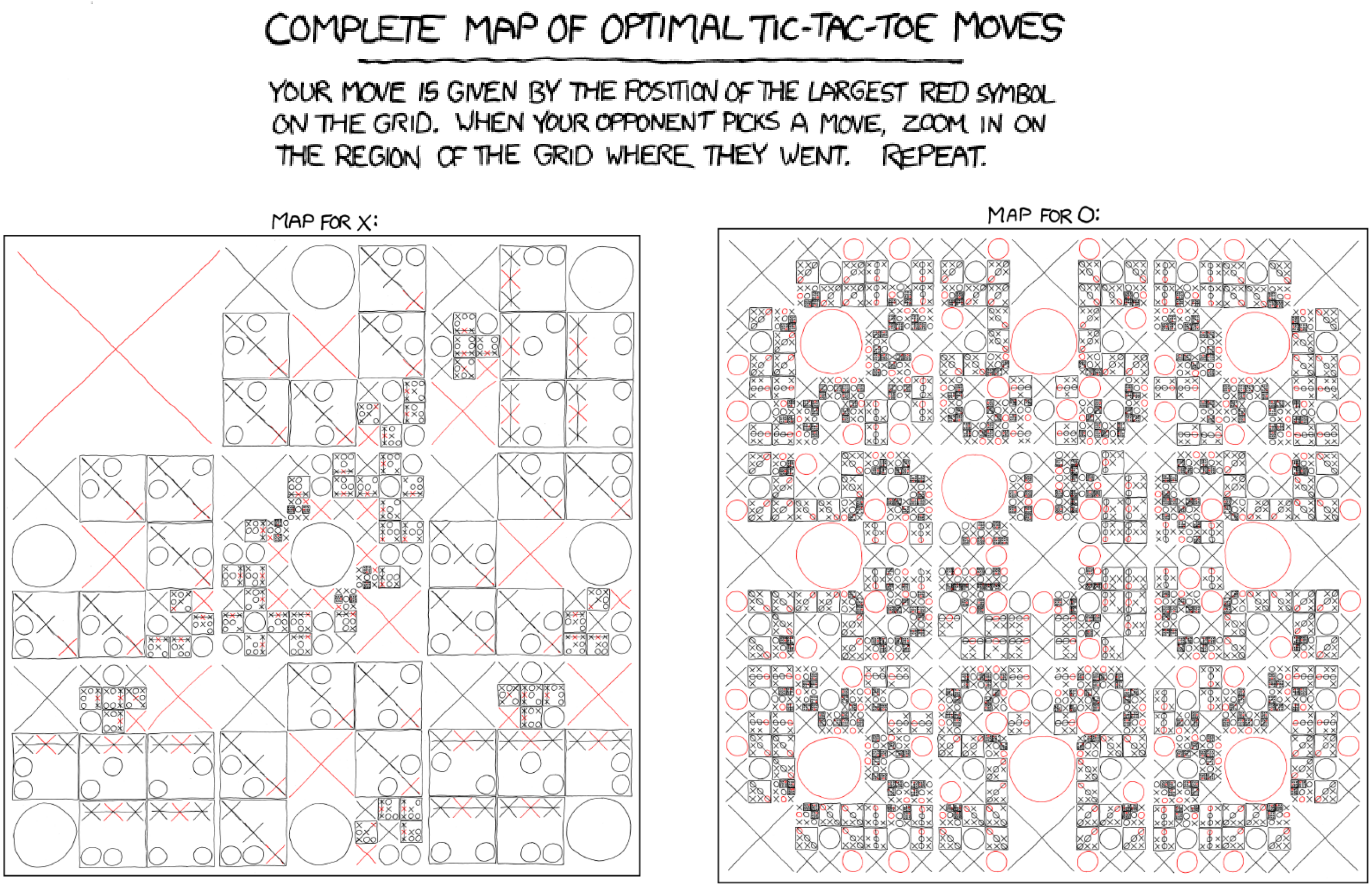


Image credit: https://xkcd.com/

Optimal Policy

A policy defines what action to take for any given game state.

- The optimal policy guarantees the best possible outcome regardless of how the opponent plays.

It's not always easy to define the optimal policy!

Proving that a policy is optimal requires examining all possible game states.

- Tic-Tac-Toe has about 9! = 362,880 states.
- Chess has over 10^{40} game states.
- Real-time games may have an infinite number of states!

Optimal Policy

How can we ensure that we make good decisions, even when we cannot consider all possible outcomes?

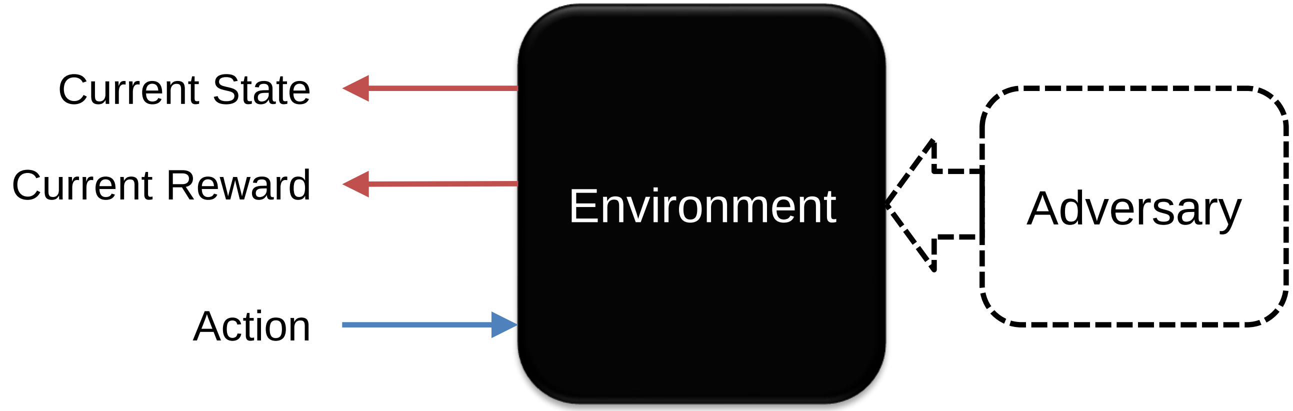
Strategies:

- Branch and bound
  - $\alpha - \beta$ pruning ignores moves that cannot influence the final decision.
- State value estimation
  - In chess, the value of a board state can be estimated by the number of remaining pieces for each player.
- Monte Carlo methods
  - Build the game tree and estimate state values from simulated games.

Reinforcement Learning

Reinforcement learning (RL) is a machine learning paradigm that governs how agents ought to make decisions in environments so as to maximize their reward.

The agent tries an action and the environment (or simulated opponent) provides a new state and reward.



Multi-Armed Bandits

How do we decide which action to try?

Without knowing the optimal policy, we must decide which move to take in each state so as to build the search tree effectively.

- Exploitation: If a move leads to a good reward, we should continue to take that move.
- Exploration: Sometimes we should try a sub-optimal move to expand the search space.

The multi-armed bandit problem:
Given several actions to choose from, how should they be sampled so as to balance exploitation and exploration?

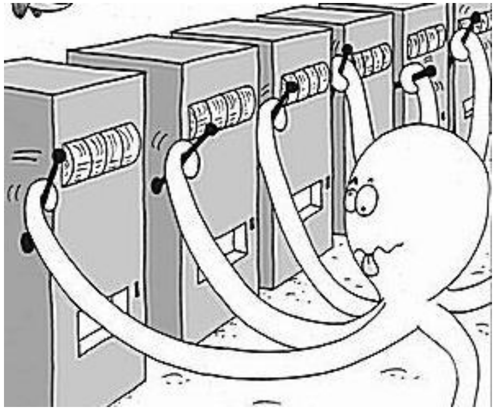


Image credit: http://research.microsoft.com

Multi-Armed Bandits

Upper Confidence Bound (UCB)

Select the action j maximizing $$UCB1 = \underbrace{\bar{X}_j}_{\text{Exploitation}} + \underbrace{C\sqrt{\frac{\ln n}{n_j}}}_{\text{Exploration}}$$

- $\bar{X}_j$ is the average reward obtained when action j is chosen
- n is the total number of plays
- n_j is the number of times action j was chosen
- C is a constant that balances exploration and exploitation
  - If the reward is bounded by $[0, 1]$, $C = \sqrt{2}$ is optimal

P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” Mach. Learn., vol. 47, no. 2, pp. 235–256, 2002.

Monte Carlo Tree Search

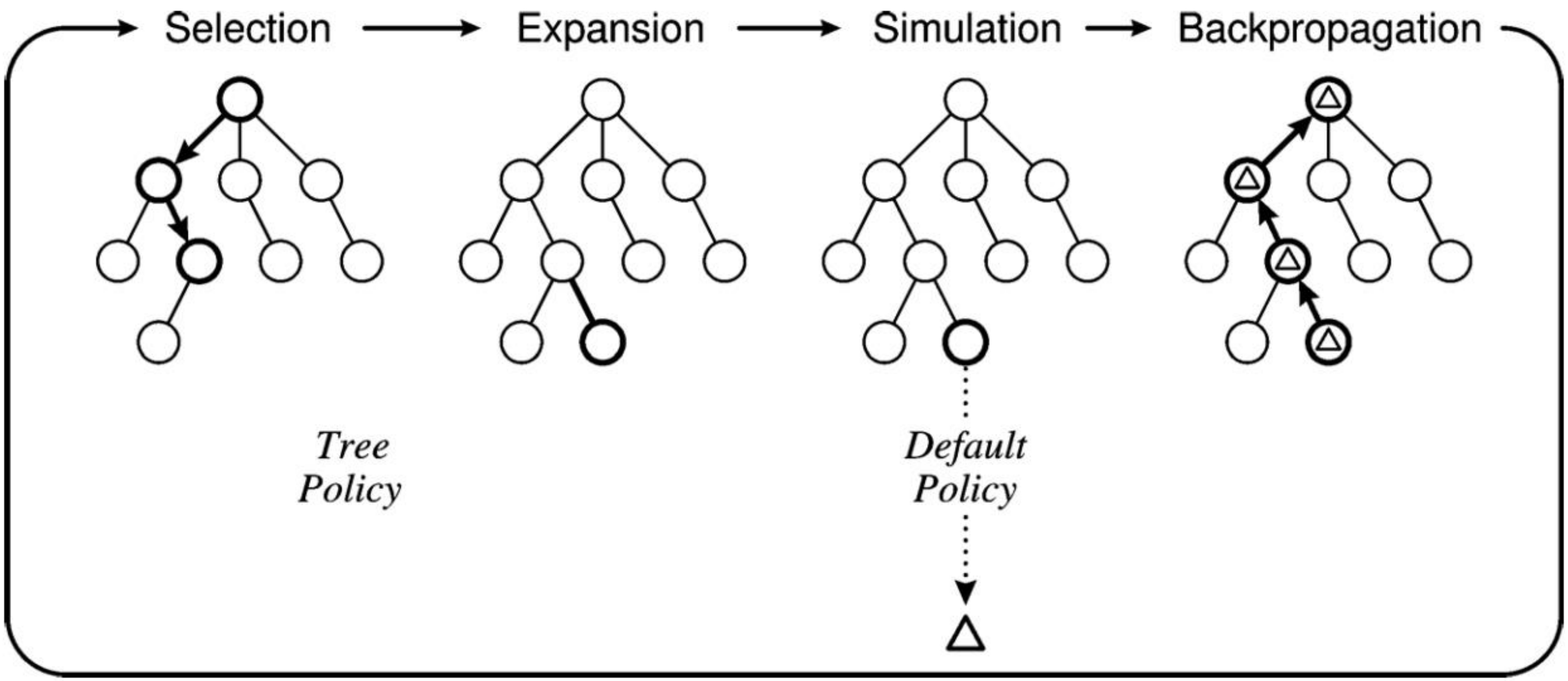
- Monte Carlo Tree Search (MCTS) is an online, anytime algorithm used by an agent to pick the next action
  - Suitable for real-time games (can always return the best result found so far)
  - Good for large search spaces with a high branching factor
    - Leading method for the game of Go
- Builds and searches an asymmetric tree
  - Requires access to a “black-box” environment simulator
  - Uses a tree variant of UCB called UCT (upper confidence trees)
    - Can use other multi-armed bandit selection techniques
  - Requires many simulations to provide accurate results (typically 1,000 – 1,000,000)

Monte Carlo Tree Search

Each iteration consists of four phases:

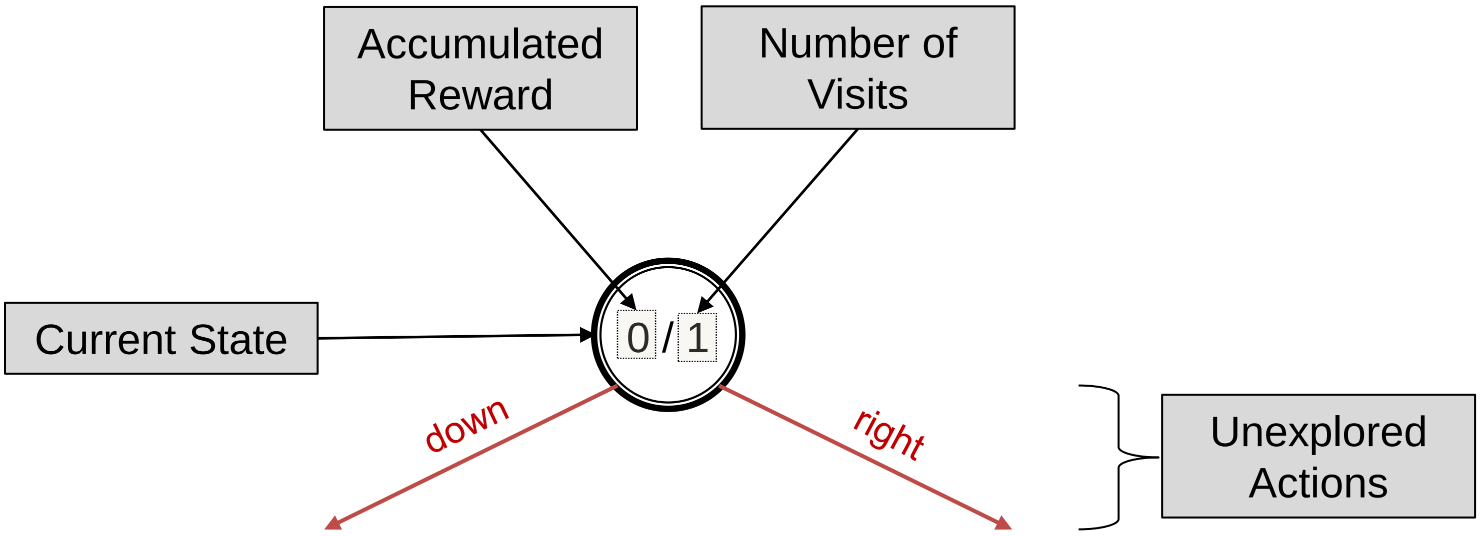
- Selection (tree policy)
  - Starting at the root, descend through the tree until finding a node with unexplored actions
- Expansion
  - Select an unexplored action and add a new child node to the tree
- Simulation (default policy)
  - Use the “black-box” simulator to run a random playout from the current state and get a reward
- Backpropagation
  - Update the statistics for each visited node

Monte Carlo Tree Search



C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, “A Survey of Monte Carlo Tree Search Methods,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1-43, 2012.

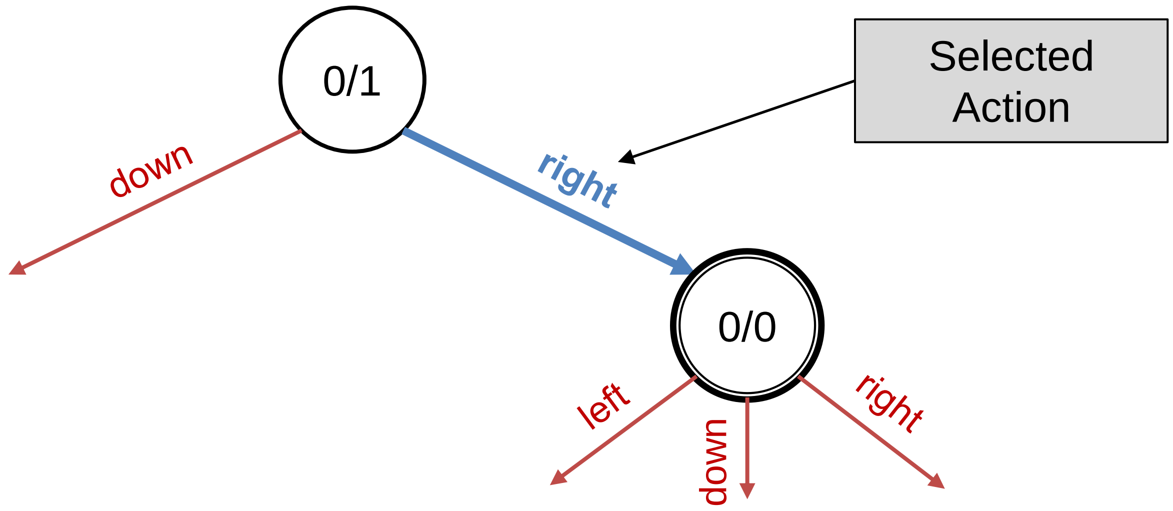
MCTS Example



<u>Step 1</u>: *Selection (Tree policy)*

- Descend through the tree until reaching a node with unexplored actions
    - *This is the root node for the first iteration*
- Use UCB1 to iteratively select children from the root node until a node with unexplored actions is found

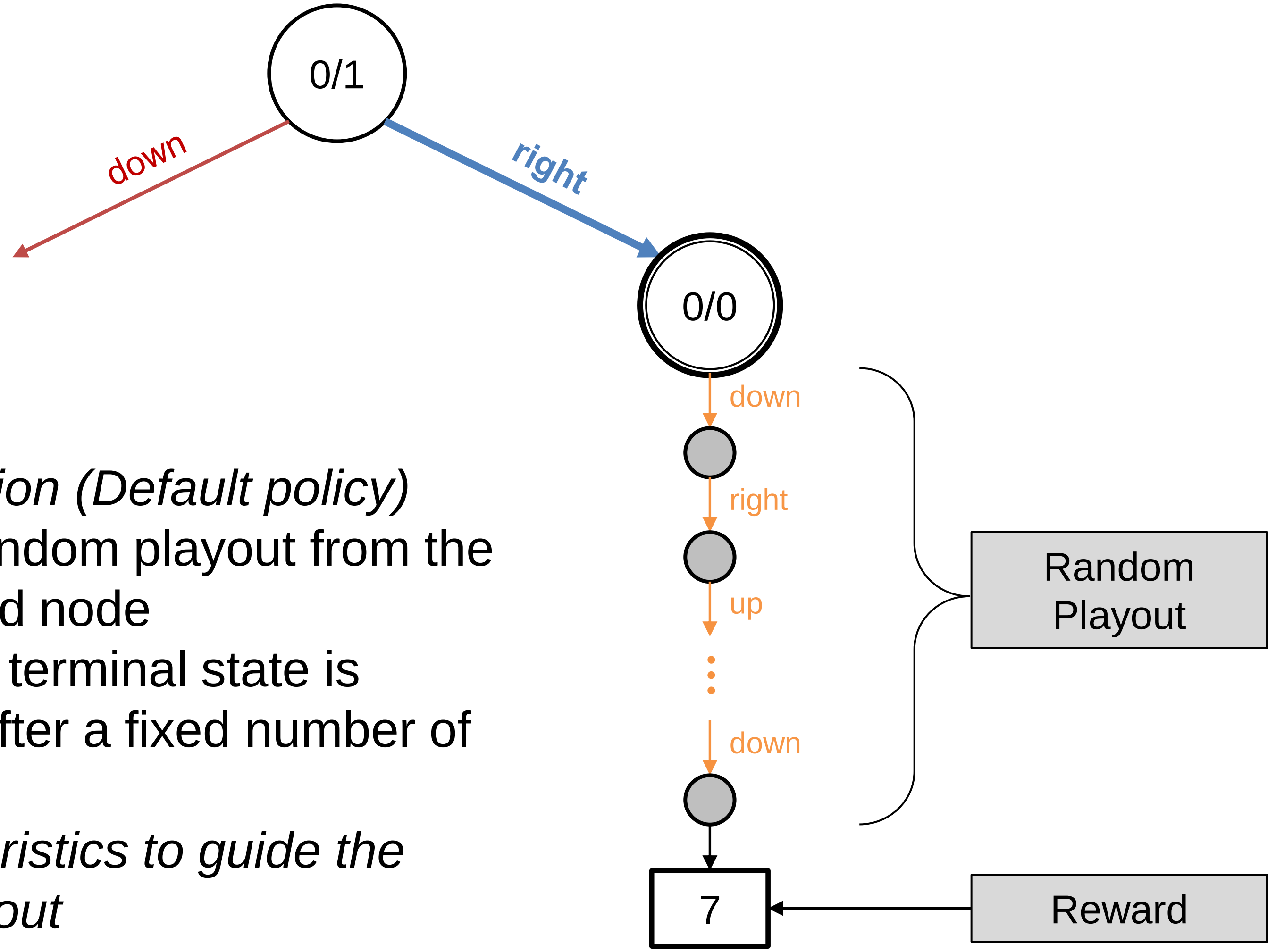
MCTS Example



Step 2: *Expansion*

- Randomly select an unexplored action and create a new child node
- *Alternative:*
  - *Use heuristics to select an action (e.g. RAVE)*

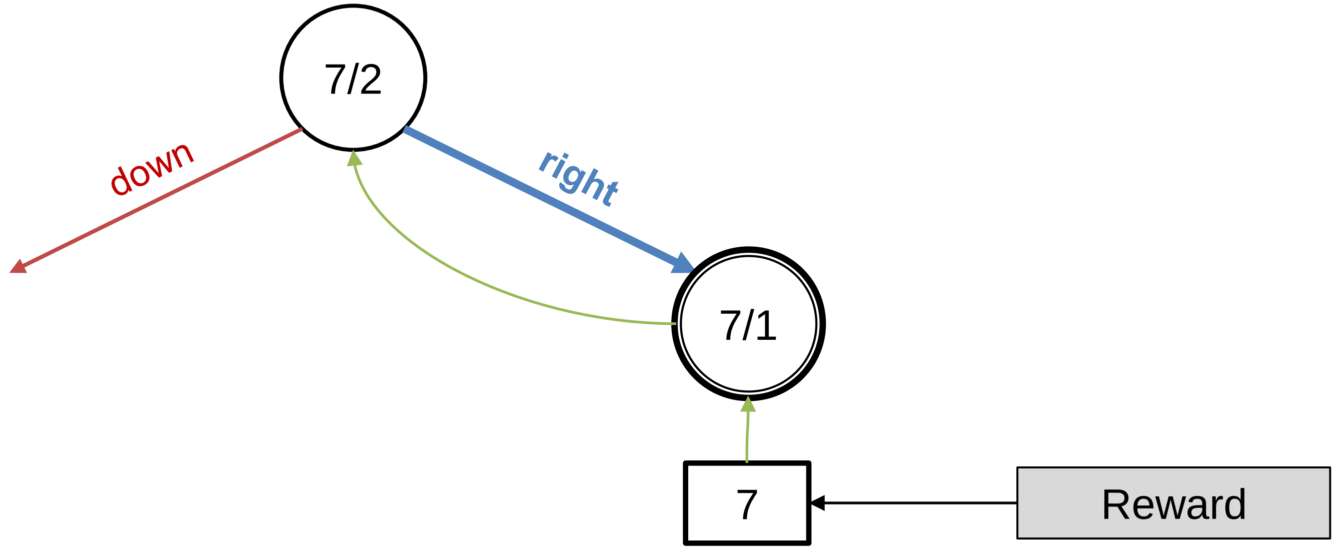
MCTS Example



<u>Step 3</u>: *Simulation (Default policy)*

- Perform a random playout from the newly created node
- Stop when a terminal state is reached or after a fixed number of steps
- *Can use heuristics to guide the random playout*

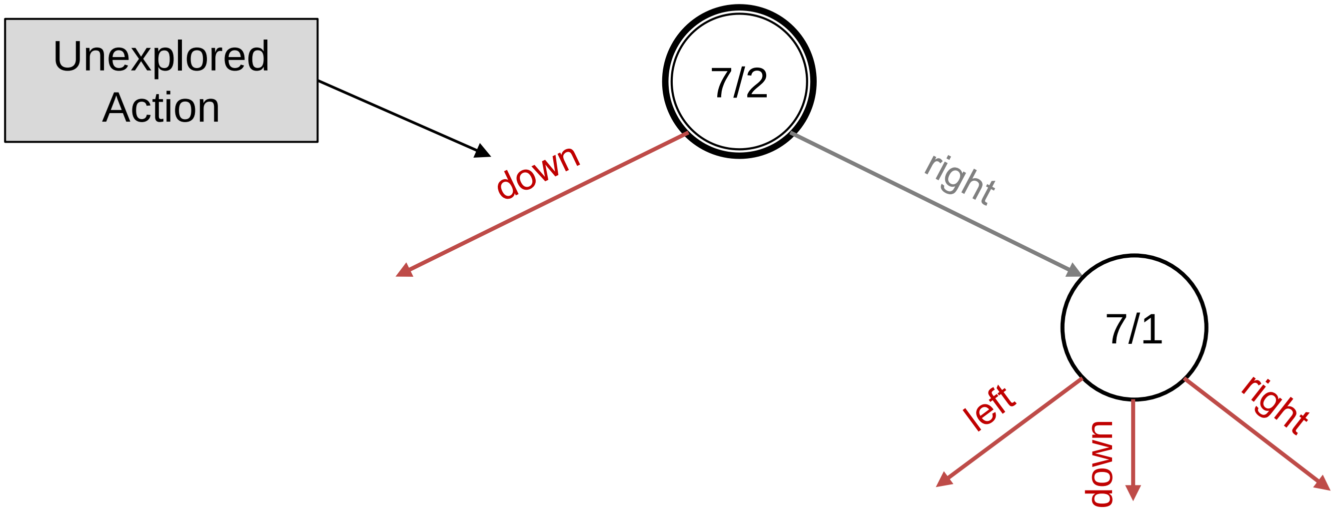
MCTS Example



Step 4: *Backpropagation*

- Revisit the parent of each node until reaching the root
- Accumulate the total reward for each node

MCTS Example

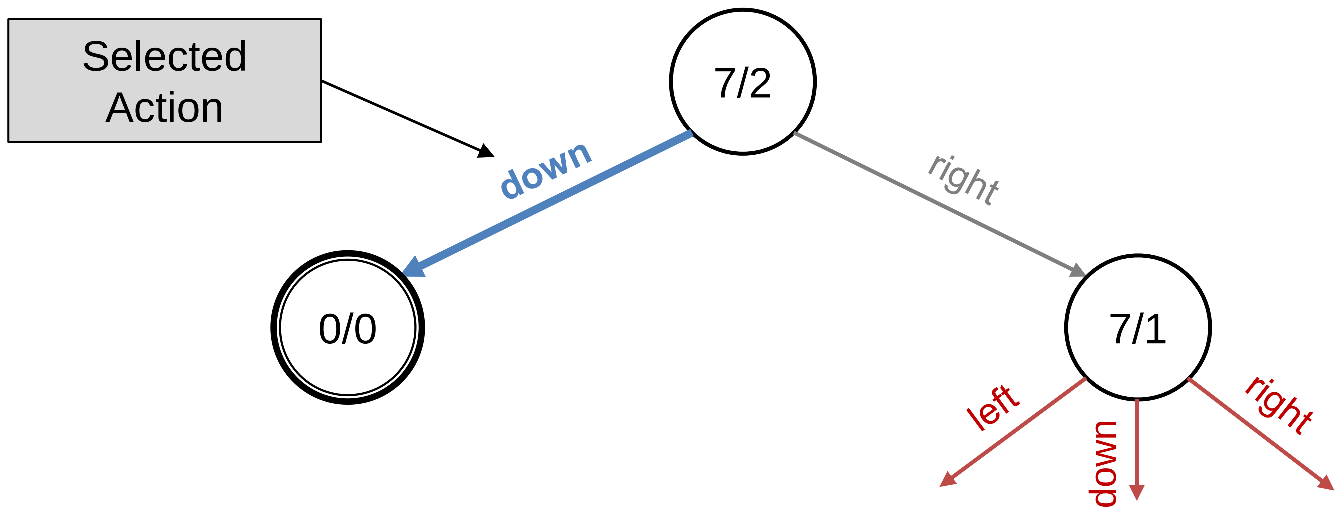


Next Iteration …

<u>Step 1</u>: *Selection (Tree policy)*

- Root node has an unexplored action, so stop at the root

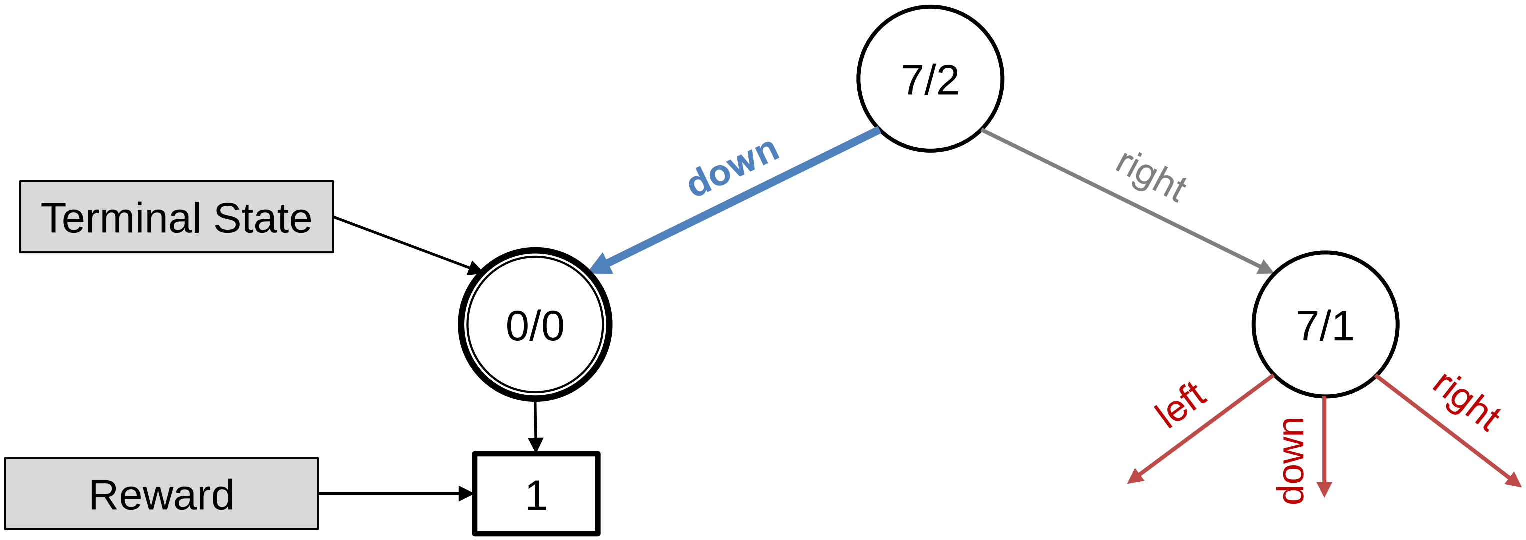
MCTS Example



Step 2: *Expansion*

- Select the unexplored action and create a new child node

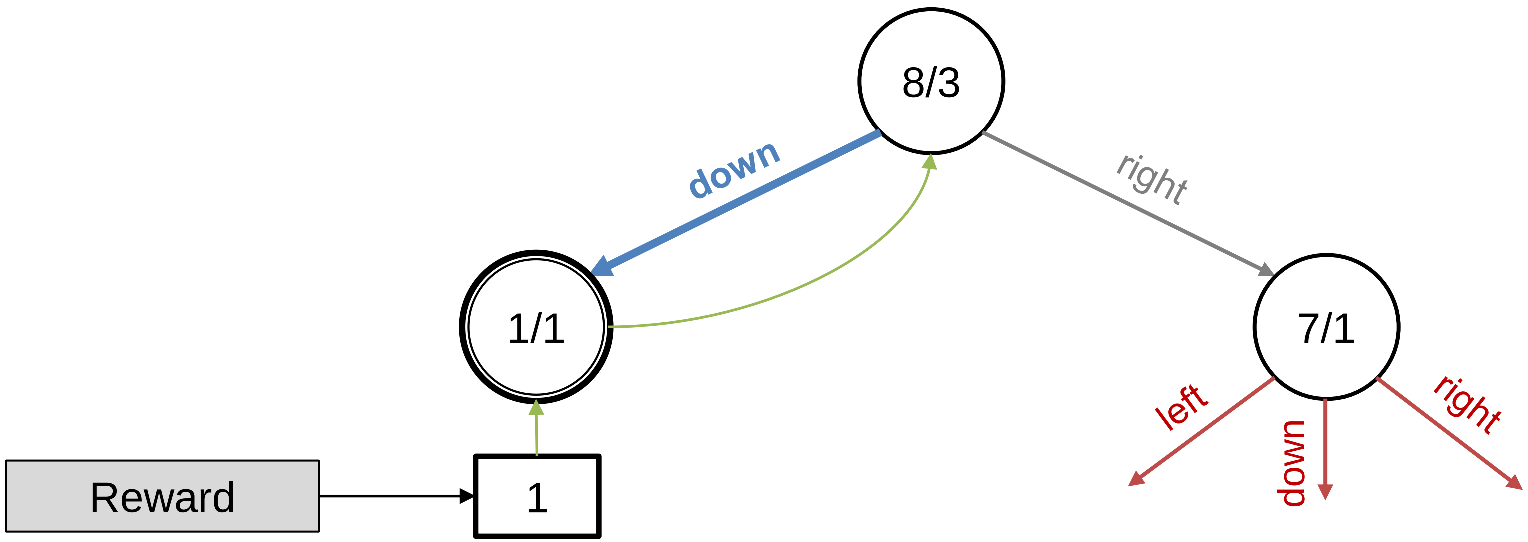
MCTS Example



Step 3: *Simulation (Default policy)*

- The new child may already be a terminal state, in which case we return the immediate reward

MCTS Example

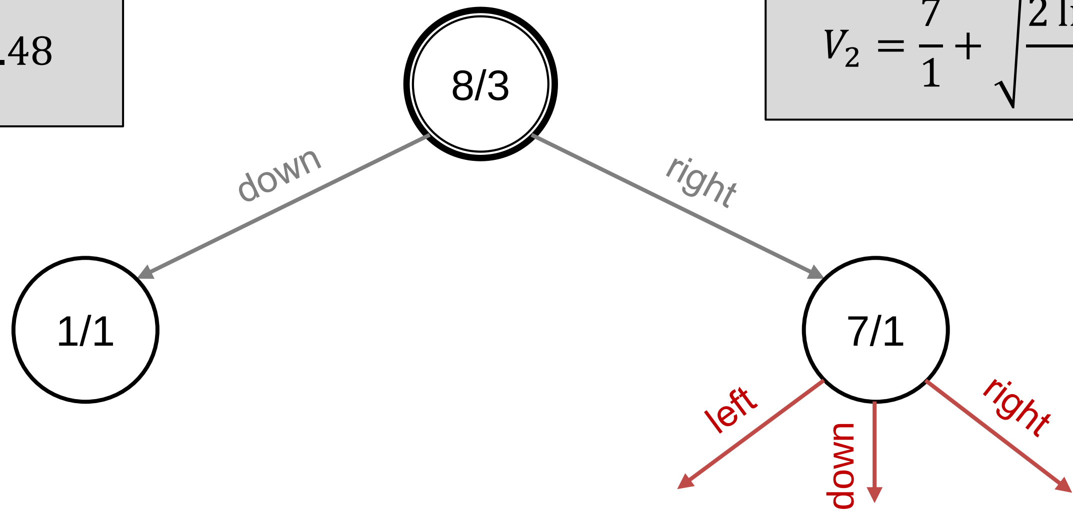


Step 4: *Backpropagation*

- Revisit the parent of each node until reaching the root
- Accumulate the total reward for each node

MCTS Example

$$V_1 = \frac{1}{1} + \sqrt{\frac{2\ln(3)}{1}} \approx 2.48$$



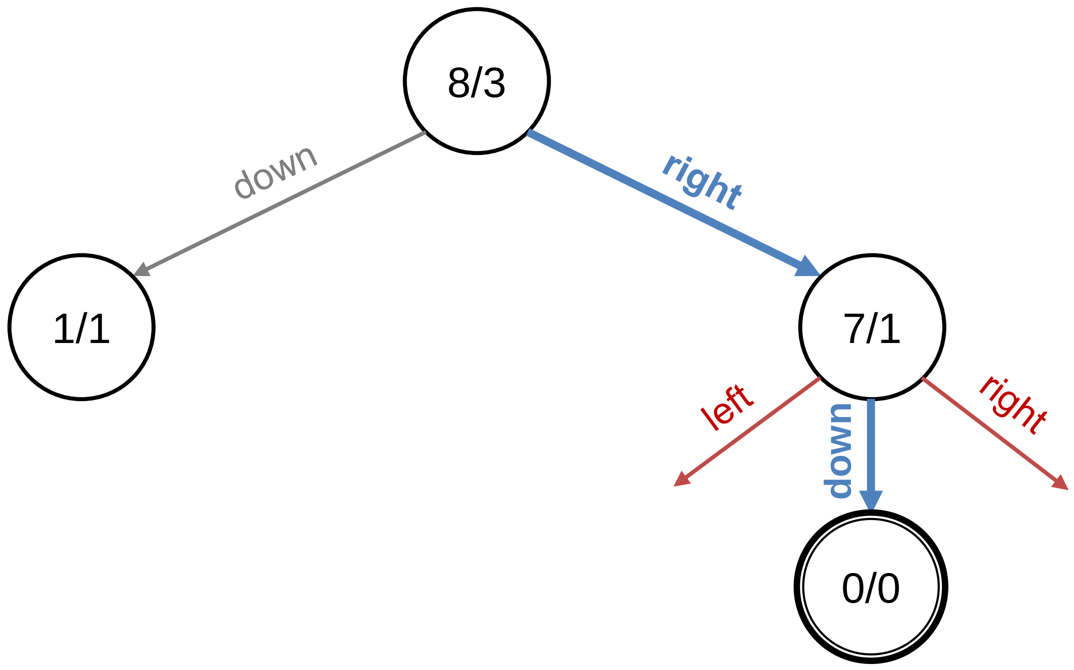
$$V_2 = \frac{7}{1} + \sqrt{\frac{2\ln(3)}{1}} \approx 8.48$$

Next Iteration …

<u>**Step 1**</u>: *Selection (Tree policy)*

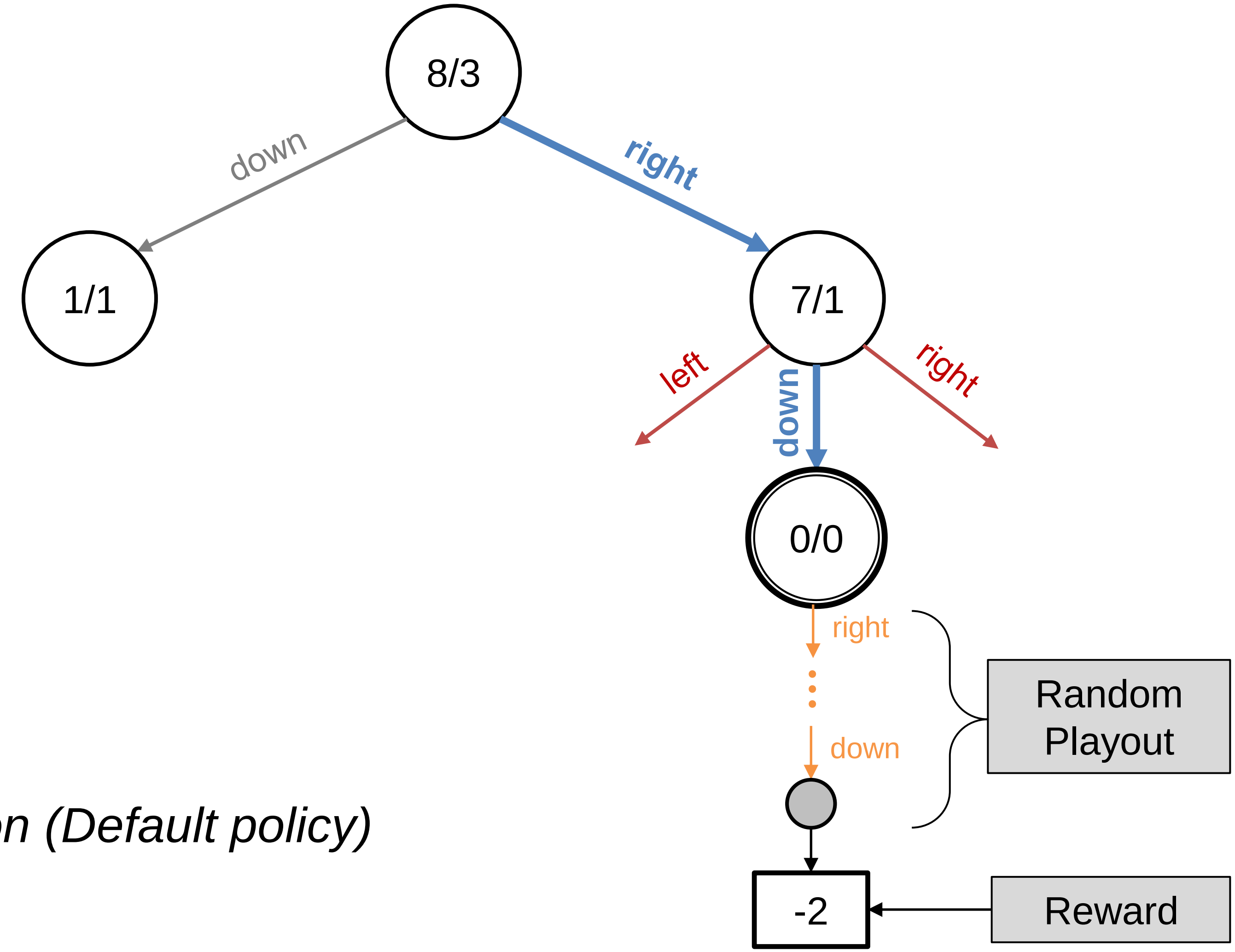
- All actions have been explored
- Compute the value of each child using the UCB1 equation
- "Right" action is biggest, which leads to a node with unexplored actions

MCTS Example



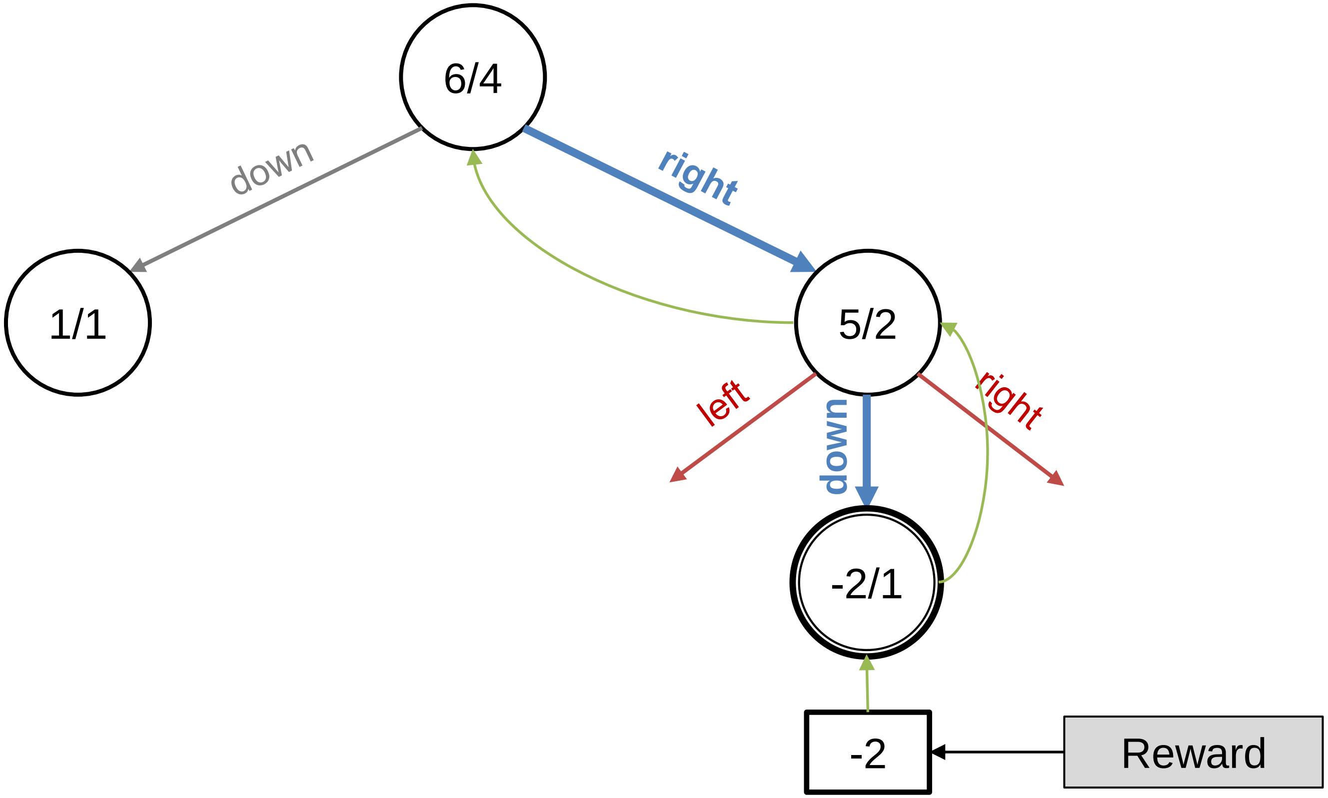
Step 2: *Expansion*

MCTS Example



<u>Step 3</u>: *Simulation (Default policy)*

MCTS Example



Step 4: *Backpropagation*

Monte Carlo Tree Search

When the computational budget is expended, return the action from the root that has been selected the most often.

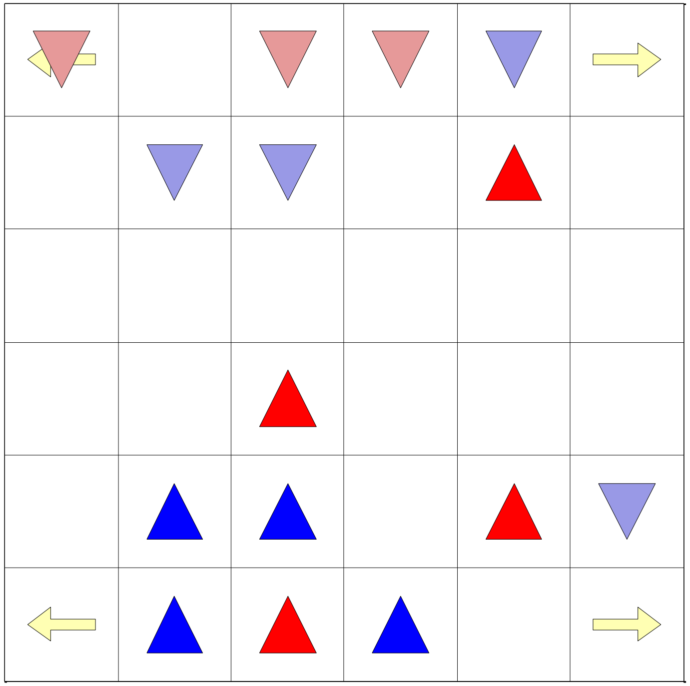
- This has been shown to outperform selecting the action with the largest average reward value.

Improvements

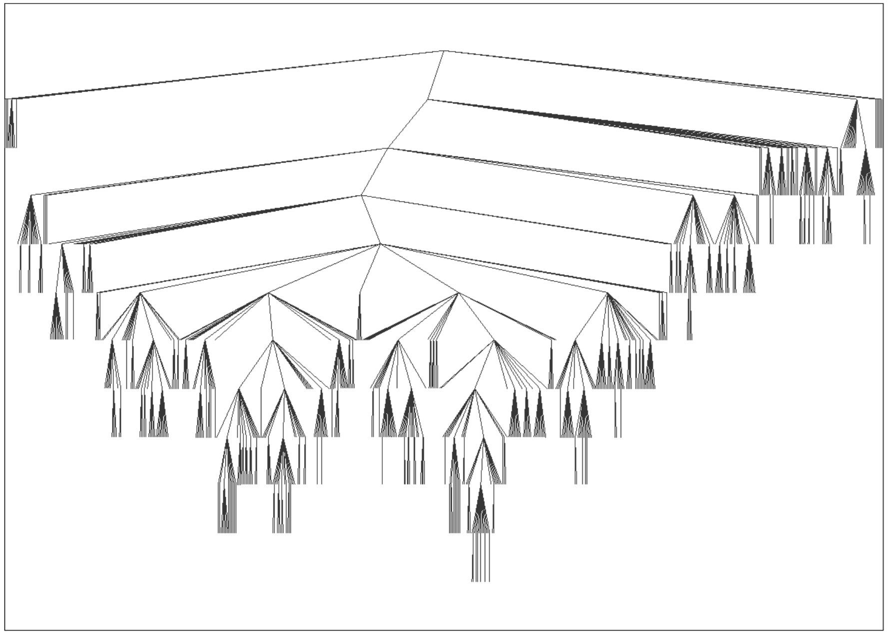
- Use domain-specific heuristics
  - Simulation playouts
  - Action selection
- Update multiple nodes each iteration
  - Use transposition tables to find similar or identical states
  - Update the statistics for actions globally

MCTS is best for problems that are too big or complicated to solve using exact methods quickly.

Example: Geister

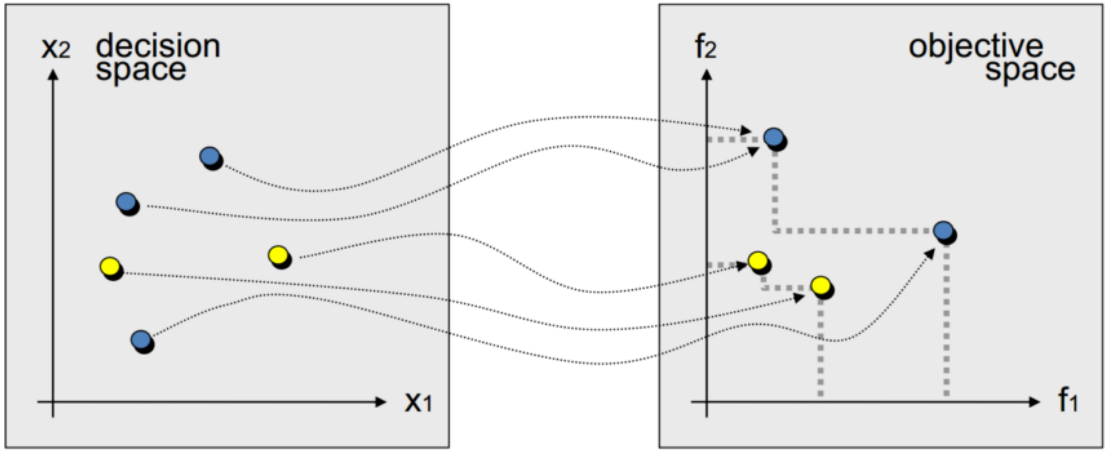


State of the board



Monte Carlo Search Tree

Multi-Objective Optimization



Multiple objective functions $f_i(x_i)$ to be maximized (or minimized)

A solution $\boldsymbol{x}$ *dominates* another solution $\boldsymbol{y}$ if and only if:

- $f_i(\boldsymbol{x})$ is not worse than $f_i(\boldsymbol{y}), \forall\, i = 1, 2, \ldots, m$
- $f_j(\boldsymbol{x})$ is better than $f_j(\boldsymbol{y})$ for at least one $j = 1, 2, \ldots, m$

The set of non-dominated solutions in decision space forms the Pareto front P in objective space.

NSGA-II

Evolutionary multi-objective optimization (EMO) algorithms are a popular choice for solving multi-objective optimization problems (MOPs).

The NSGA-II algorithm works well for 2 objectives, but alternatives exist for problems with more objectives.

The algorithm accounts for Pareto rank and crowding to provide a distribution of solutions along the Pareto front.

Algorithm 1 NSGA-II Algorithm

Input: MOP, N
Output: Non-dominated Set F_0
$t = 0$
$Pop(t) = NewRandomPopulation$
$Q(t) = breed(Pop(t))$ % Generate offspring
while Termination criterion not met **do**
    $U(t) = Pop(t) \cup Q(t)$
    $F = \textsc{FastNonDominatedSort}(U(t))$
    $Pop(t+1) = \emptyset,\ i = 0$
    while $|Pop(t+1)| + |F_i| \leq N$ **do**
        $\textsc{CrowdingDistanceAssignment}(F_i)$
        $Pop(t+1) = Pop(t+1) \cup F_i$
        $i = i + 1$
    $\textsc{Sort}(Pop(t+1))$
    $Pop(t+1) = Pop(t+1) \cup F_i[1 : (N - |Pop(t+1)|)]$
    $Q(t+1) = breed(Pop(t+1))$
    $t = t + 1$
return F_0

Multi-Objective RL

A reinforcement learning problem may have a reward vector instead of a scalar reward.

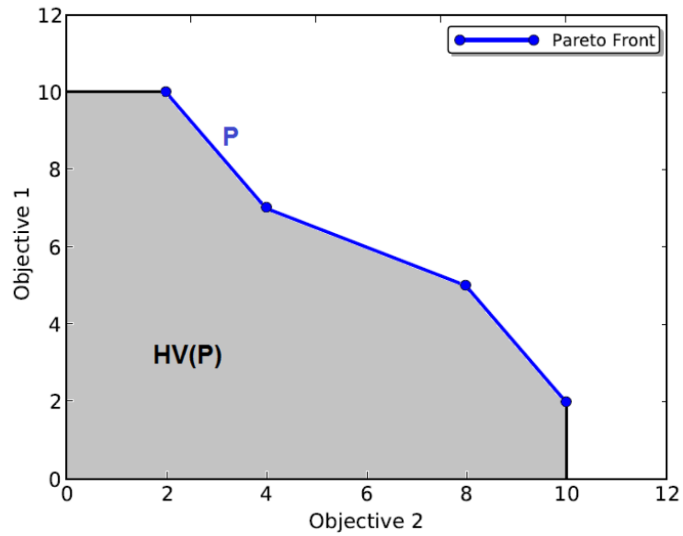
Single-policy algorithms simplify the reward into a conventional scalar reward

- Scalarization
- Objective preference ordering

Multiple-policy algorithms work to find the optimal Pareto front

- Convex Hull Iteration
- Multiple-Objective Monte Carlo Tree Search

Multi-Objective Metric



The Hypervolume Indicator (HV) is a popular metric for measuring the quality of a Pareto front P.

It is defined as the volume of the objective space dominated by P.

Computing $HV(P)$ is exponential in the number of objective dimensions.

- In high dimensions, the value can be approximated with Monte Carlo sampling

Multi-Objective MCTS

In Multi-Objective Monte Carlo Tree Search, each node stores a local Pareto front approximation.

- This allows each node to have an estimate of the quality of the solutions reachable from there.

Rewards are backpropagated only if they would expand the local Pareto front.

- A child's Pareto front can never dominate its parent.
- The root node has the best non-dominated front found during the search.

The average reward in the UCB1 equation is replaced with the average value of HV:

$$a^* = \underset{a \in A(s)}{\arg\max} \left\{ \frac{HV(P)}{N(s,a)} + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right\}$$

Multi-Objective MCTS

How to select a solution from the Pareto front?

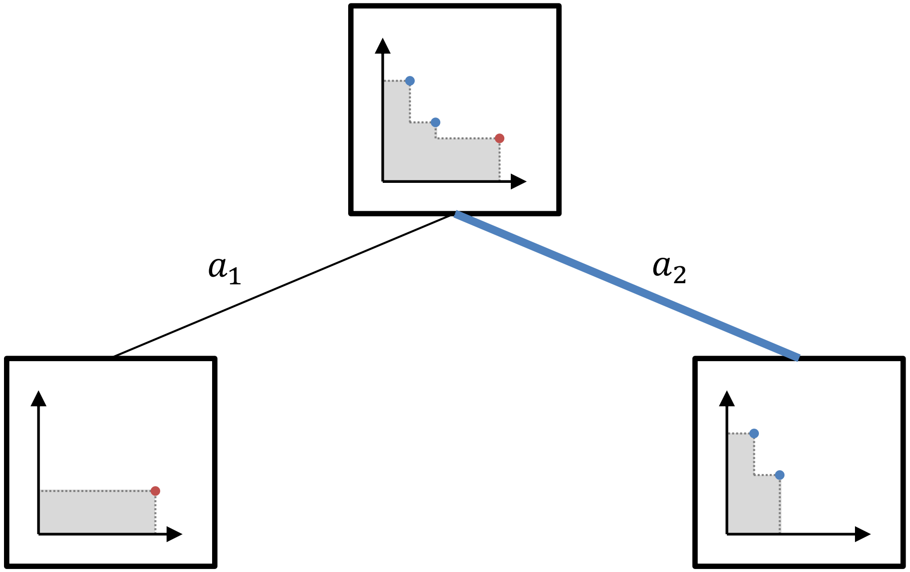
A weight vector can be defined $W = (w_1, \dots, w_m)$; $\sum_i^m w_i = 1$

- <u>Weighted sum</u>: Choose the action that maximizes the weighted sum of the reward vector multiplied by W

- <u>Euclidean distance</u>: Normalize the points in the Pareto front into the range $[0, 1]$ and choose the action that minimizes the distance to the weight vector W

Other methods can be used to select an action from the Pareto front.

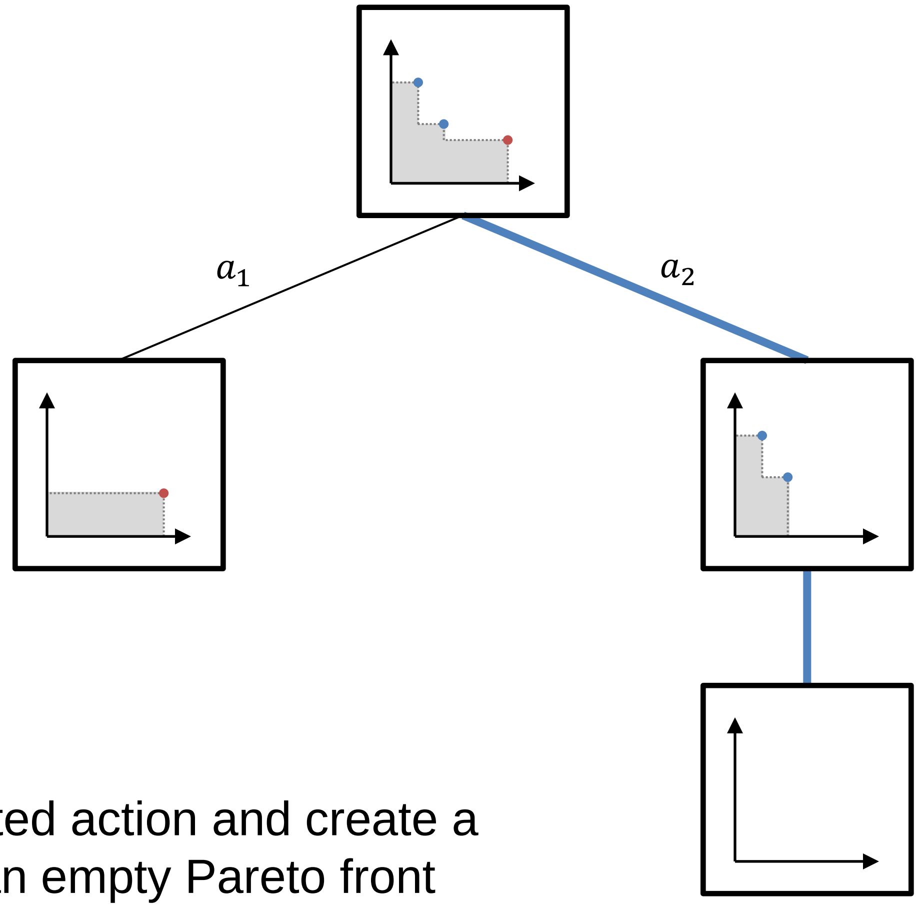
MO-MCTS Example



<u>Step 1</u>: *Selection (Tree policy)*

- Descend through the tree using UCB1 as before
- Node value is given by the hypervolume indicator

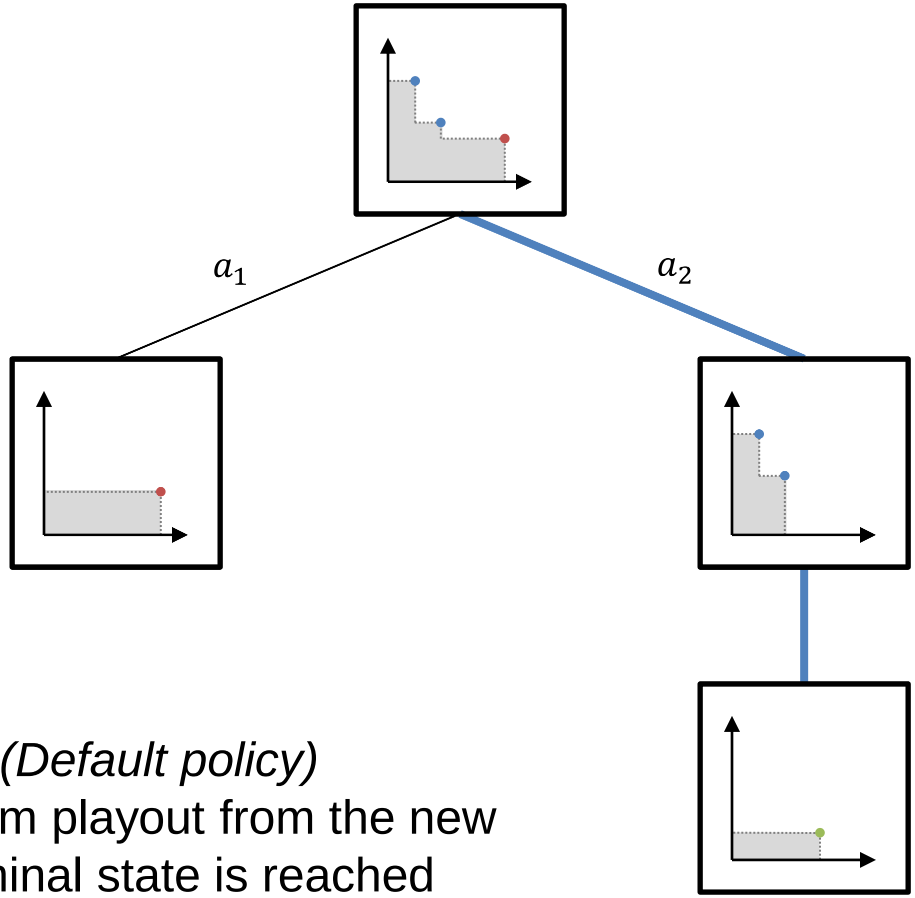
MO-MCTS Example



Step 2: *Expansion*

- Select an unvisited action and create a new node with an empty Pareto front

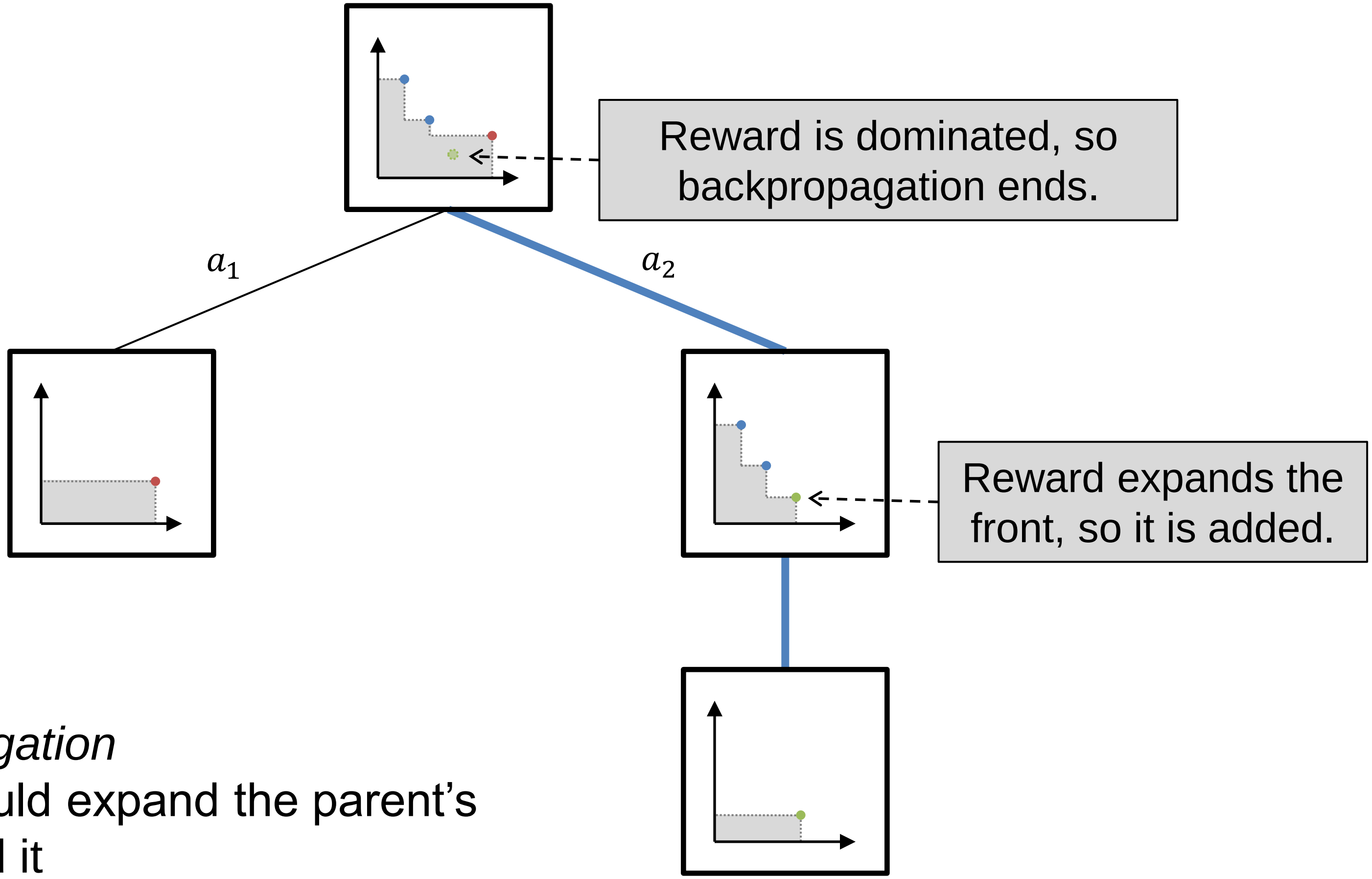
MO-MCTS Example



Step 3: *Simulation (Default policy)*

- Perform a random playout from the new node until a terminal state is reached
- Add the reward vector to the local Pareto front

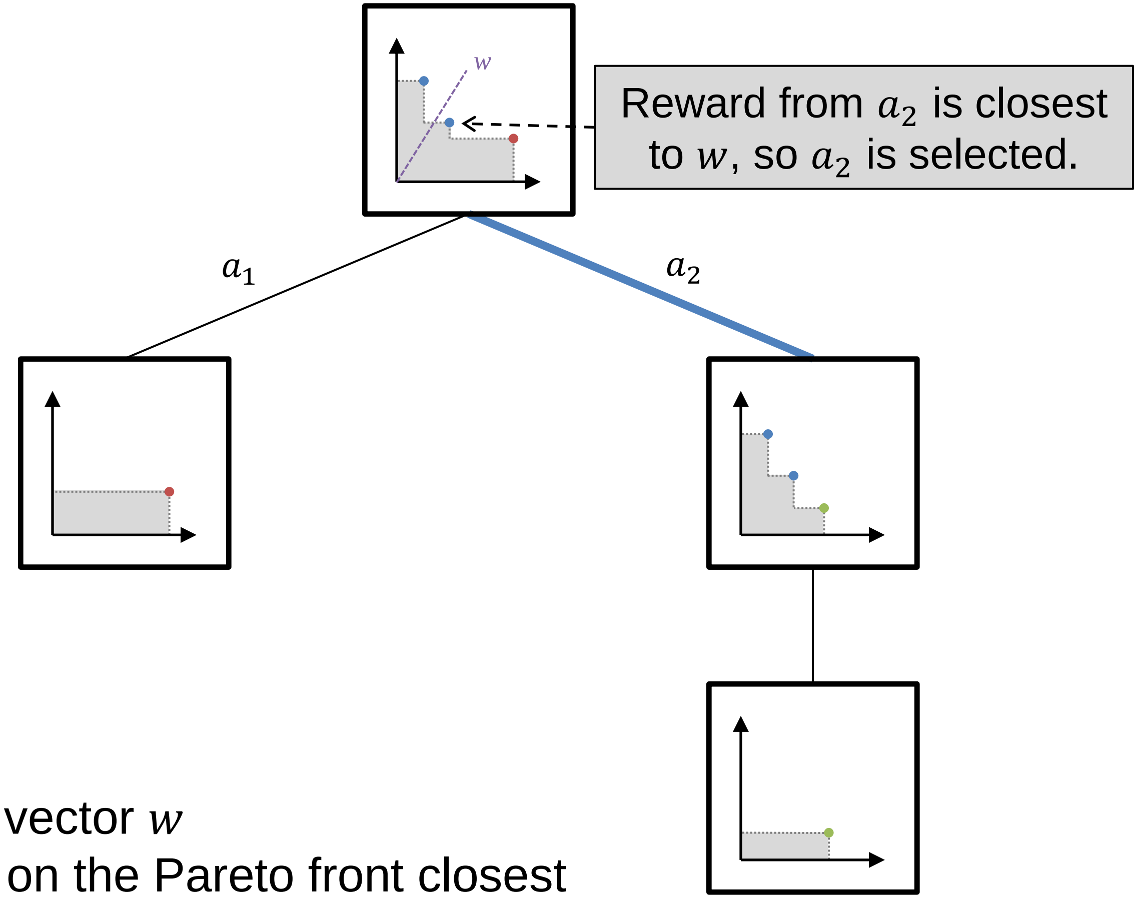
MO-MCTS Example



<u>Step 4</u>: *Backpropagation*

- If the reward would expand the parent's Pareto front, add it
- Otherwise, stop backpropagation

MO-MCTS Example

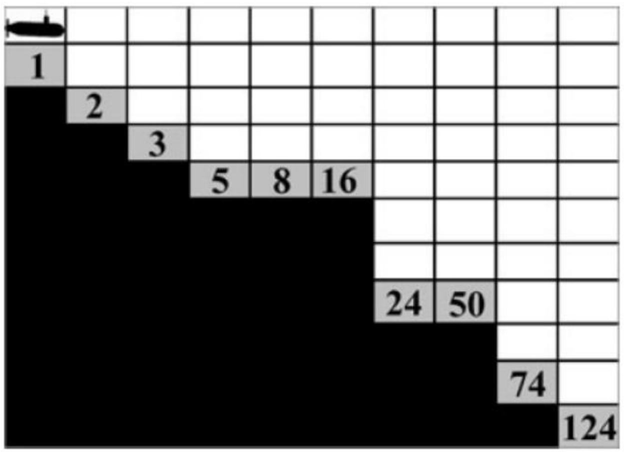


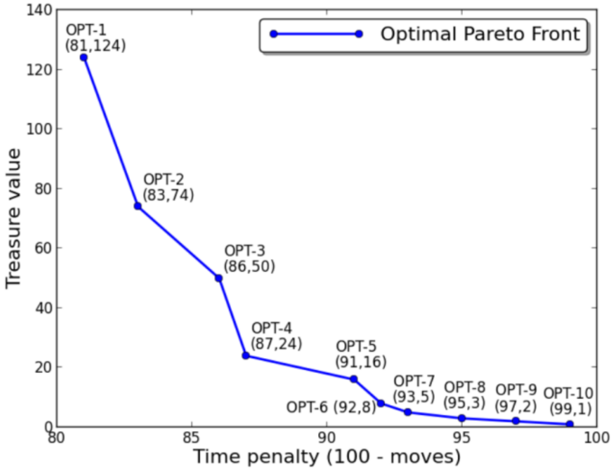
Action Selection

- Define a weight vector w
- Find the reward on the Pareto front closest to the weight vector
- Select the action associated with that reward

MO-MCTS Experiments

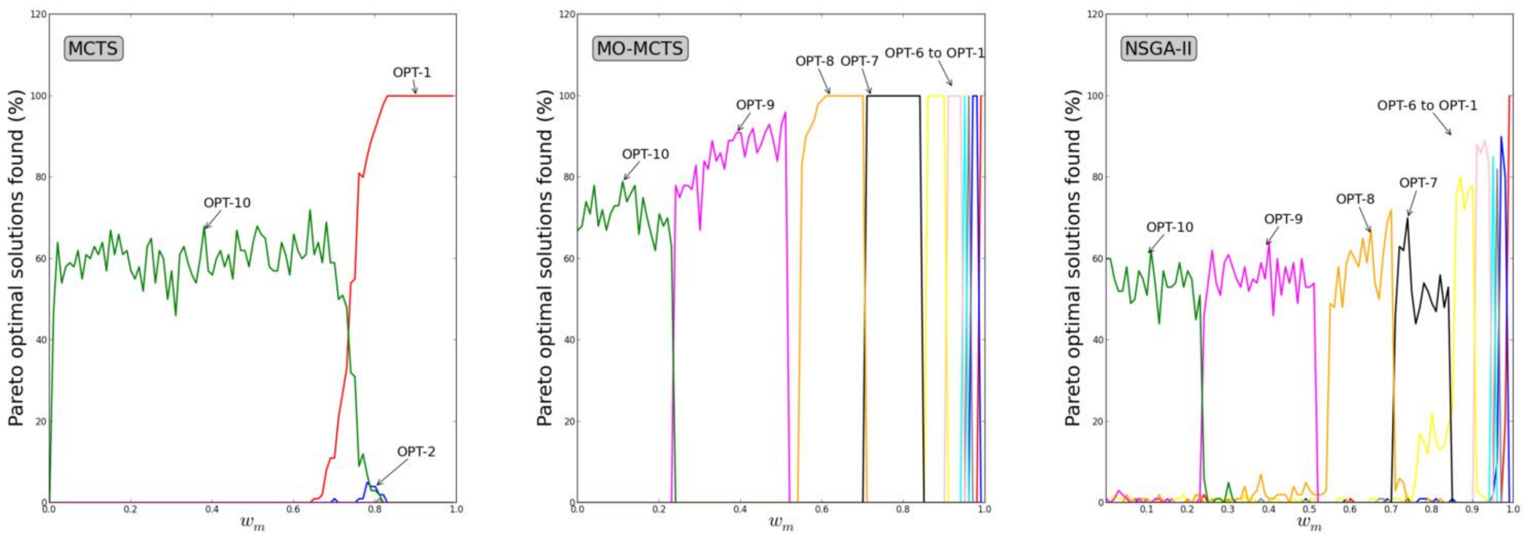
Deep Sea Treasure





The agent can move *up*, *down*, *left*, or *right* and ends the game upon reaching a treasure (or a 100 move limit). The goal is to maximize treasure value and minimize distance.

MO-MCTS Experiments



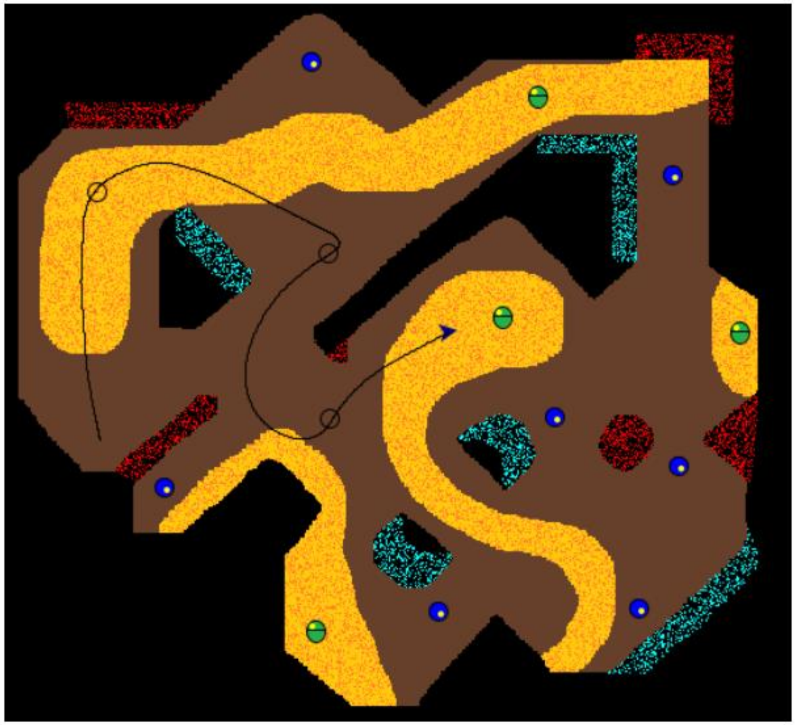
The weight vector $W = (w_m, 1 - w_m)$ is varied in 0.01 step increments and 100 runs are performed for each setting. Euclidean distance is used for MO-MCTS.

The graphs show the percentage of the runs that converged to each of the 10 true optimal points for this problem.

Note that MO-MCTS found more optima than regular MCTS and converged to them more often.

MO-MCTS Experiments

Physical Traveling Salesman Problem



Unvisited waypoints are blue circles
Visited waypoints are empty circles
Fuel canisters are green circles

- Agent must visit all 10 waypoints.
- Provide an action every 40ms
  - Throttle (on, off)
  - Steering (straight, left, right)
- Objectives are
  - Minimize distance
  - Minimize fuel use
  - Minimize damage
- Start with 5000 units of fuel
  - Waypoints provide 50 fuel units
  - Fuel canisters provide 250 fuel units
- Start with 5000 damage points
  - Hitting black walls causes 10 damage
  - Hitting red walls causes 30 damage
  - Driving through lava causes 1 damage

MO-MCTS Experiments

Reward vector to be maximized: $\bar{r} = \{\rho_t, \rho_f, \rho_d\}$

Time: $\rho_t = 1 - d_t/d_M$

- d_t : Minimum distance remaining through all waypoints
- d_M : Distance of the whole route from the start position

Fuel: $\rho_f = (1 - \lambda_t/\lambda_0) \times \alpha + \rho_t \times (1 - \alpha)$

- λ_t : Fuel consumed so far
- $\lambda_0 = 5000$: Initial fuel at the start of the game
- $\alpha = 0.66$: Balance fuel and time (to prevent standing still)

Damage: $\rho_d = \begin{cases} (1 - g_t/g_M) \times \beta_1 + \rho_t \times (1 - \beta_1), & sp > \gamma \\ (1 - g_t/g_M) \times \beta_2 + \rho_t \times (1 - \beta_2), & sp \leq \gamma \end{cases}$

- g_t : Damage suffered so far
- $g_M = 5000$: Max possible damage
- $\beta_1 = 0.75$: Balance damage and time at high speeds
- $\beta_2 = 0.25$: Balance damage and time at low speeds
- $\gamma = 0.8$: Speed threshold

MO-MCTS Experiments

The algorithm was run 30 times each on 10 maps with 4 predefined weight vectors.

By adjusting the weights, different solutions could be found.

Map	$W : (w_t, w_f, w_d)$	Time	Fuel	Damage
Map 1	$(0.33, 0.33, 0.33)$	1654 (7)	131 (2)	846 (13)
	$(0.1, 0.3, 0.6)$	1657 (8)	130 (2)	**773(11)**
	$(0.1, 0.6, 0.3)$	1681 (11)	131 (2)	837 (15)
	$(0.6, 0.1, 0.3)$	1649 (8)	132 (2)	833 (13)
Map 2	$(0.33, 0.33, 0.33)$	1409 (7)	235 (4)	364 (3)
	$(0.1, 0.3, 0.6)$	1402 (6)	236 (5)	**354(2)**
	$(0.1, 0.6, 0.3)$	1416 (8)	**219(4)**	360 (3)
	$(0.6, 0.1, 0.3)$	1396 (8)	245 (5)	361 (2)
Map 3	$(0.33, 0.33, 0.33)$	1373 (6)	221 (3)	301 (7)
	$(0.1, 0.3, 0.6)$	1378 (5)	211 (3)	**268(5)**
	$(0.1, 0.6, 0.3)$	1385 (6)	**203(4)**	291 (7)
	$(0.6, 0.1, 0.3)$	1363 (4)	229 (4)	285 (7)
Map 4	$(0.33, 0.33, 0.33)$	1383 (6)	291 (5)	565 (5)
	$(0.1, 0.3, 0.6)$	1385 (7)	304 (4)	**542(4)**
	$(0.1, 0.6, 0.3)$	1423 (6)	**273(4)**	583 (5)
	$(0.6, 0.1, 0.3)$	1388 (6)	309 (4)	559 (5)
Map 5	$(0.33, 0.33, 0.33)$	1405 (7)	467 (4)	559 (4)
	$(0.1, 0.3, 0.6)$	1431 (9)	447 (4)	**541(4)**
	$(0.1, 0.6, 0.3)$	1467 (9)	**411(5)**	567 (5)
	$(0.6, 0.1, 0.3)$	1399 (9)	469 (4)	547 (3)
Map 6	$(0.33, 0.33, 0.33)$	1575 (7)	549 (5)	303 (4)
	$(0.1, 0.3, 0.6)$	1626 (9)	540 (6)	286 (5)
	$(0.1, 0.6, 0.3)$	1703 (11)	**499(4)**	316 (7)
	$(0.6, 0.1, 0.3)$	1571 (7)	559 (5)	294 (4)
Map 7	$(0.33, 0.33, 0.33)$	1434 (5)	599 (6)	284 (6)
	$(0.1, 0.3, 0.6)$	1475 (10)	602 (5)	243 (6)
	$(0.1, 0.6, 0.3)$	1489 (12)	**549(3)**	264 (6)
	$(0.6, 0.1, 0.3)$	**1407(8)**	618 (5)	270 (6)
Map 8	$(0.33, 0.33, 0.33)$	1761 (9)	254 (5)	382 (3)
	$(0.1, 0.3, 0.6)$	1804 (10)	269 (4)	**357(4)**
	$(0.1, 0.6, 0.3)$	1826 (10)	230 (3)	392 (7)
	$(0.6, 0.1, 0.3)$	1732 (9)	311 (8)	379 (6)
Map 9	$(0.33, 0.33, 0.33)$	2501 (14)	926 (6)	574 (9)
	$(0.1, 0.3, 0.6)$	2503 (10)	921 (10)	**524(8)**
	$(0.1, 0.6, 0.3)$	2641 (14)	**833(5)**	574 (14)
	$(0.6, 0.1, 0.3)$	2470 (9)	956 (5)	573 (8)
Map 10	$(0.33, 0.33, 0.33)$	1430 (8)	630 (4)	205 (2)
	$(0.1, 0.3, 0.6)$	1493 (13)	615 (4)	209 (2)
	$(0.1, 0.6, 0.3)$	1542 (10)	**554(4)**	229 (5)
	$(0.6, 0.1, 0.3)$	**1378(5)**	663 (6)	202 (4)

TABLE I: MO-PTSP averages (plus standard error) with different weight vectors. Results in bold obtained an independent t-test p-value < 0.01.

MO-MCTS Experiments

All algorithms were compared in terms of solution dominance across the maps.

MO-MCTS dominates MCTS and NSGA-II more frequently and is less frequently dominated by PurofMovio, the winning solution from the game competition.

	$W : (w_t, w_f, w_d)$	**MO-MCTS** $(D, \emptyset, d)$	**MCTS** $(D, \emptyset, d)$	**NSGA-II** $(D, \emptyset, d)$	**PurofMovio** $(D, \emptyset, d)$
MO-MCTS	$W_1 : (0.33, 0.33, 0.33)$	–	$(8, 2, 0)$	$(8, 2, 0)$	$(0, 5, 5)$
	$W_2 : (0.1, 0.3, 0.6)$		$(10, 0, 0)$	$(4, 6, 0)$	$(0, 6, 4)$
	$W_3 : (0.1, 0.6, 0.3)$		$(8, 2, 0)$	$(7, 3, 0)$	$(0, 5, 5)$
	$W_4 : (0.6, 0.1, 0.3)$		$(10, 0, 0)$	$(3, 7, 0)$	$(0, 3, 7)$
MCTS	$W_1 : (0.33, 0.33, 0.33)$	$(0, 8, 2)$	–	$(0, 2, 8)$	$(0, 2, 8)$
	$W_2 : (0.1, 0.3, 0.6)$	$(0, 0, 10)$		$(4, 2, 4)$	$(0, 3, 7)$
	$W_3 : (0.1, 0.6, 0.3)$	$(0, 2, 8)$		$(0, 1, 9)$	$(0, 6, 4)$
	$W_4 : (0.6, 0.1, 0.3)$	$(0, 0, 10)$		$(3, 3, 4)$	$(0, 1, 9)$
NSGA-II	$W_1 : (0.33, 0.33, 0.33)$	$(0, 2, 8)$	$(8, 2, 0)$	–	$(0, 4, 6)$
	$W_2 : (0.1, 0.3, 0.6)$	$(0, 6, 4)$	$(4, 2, 4)$		$(0, 4, 6)$
	$W_3 : (0.1, 0.6, 0.3)$	$(0, 3, 7)$	$(9, 1, 0)$		$(0, 5, 5)$
	$W_4 : (0.6, 0.1, 0.3)$	$(0, 7, 3)$	$(4, 3, 3)$		$(0, 4, 6)$
PurofMovio	$W_1 : (0.33, 0.33, 0.33)$	$(5, 5, 0)$	$(8, 2, 0)$	$(6, 4, 0)$	–
	$W_2 : (0.1, 0.6, 0.3)$	$(4, 6, 0)$	$(7, 3, 0)$	$(6, 4, 0)$	
	$W_3 : (0.1, 0.3, 0.6)$	$(5, 5, 0)$	$(6, 4, 0)$	$(5, 5, 0)$	
	$W_4 : (0.6, 0.1, 0.3)$	$(7, 3, 0)$	$(9, 1, 0)$	$(6, 4, 0)$	

TABLE II: Results in MO-PTSP: Each cell indicates the triplet $(D, \emptyset, d)$, where D is the number of maps where the row algorithm dominates the column one, $\emptyset$ is the amount of maps where no dominance can be established, and d states the number of maps where the row algorithm is dominated by the column one. All the algorithms followed the same route (order of waypoints and fuel canisters) in every map tested.

MO-MCTS Experiments

In the final experiment, the weight vector is allowed to change between each waypoint.

$$1 = W_1 = (0.33, 0.33, 0.33)$$
$$2 = W_2 = (0.1, 0.3\ 0.6)$$
$$3 = W_3 = (0.1, 0.6, 0.3)$$

Computing the sequence with a hill-climbing algorithm gave improved performance over static weights.

Map	Weight genome	Time	Fuel	Damage	D
Map 1	11111111111111	1654 (7)	131 (2)	846 (13)	$\preceq$
	22222222222222	1657 (8)	130 (2)	773 (11)	$\preceq$
	33333333333333	1681 (11)	131 (2)	837 (15)	$\preceq$
	32312212331112	1619 (12)	130 (2)	744 (15)	
Map 2	11111111111111	1409 (7)	235 (4)	364 (3)	$\preceq$
	22222222222222	1402 (6)	236 (5)	354 (2)	$\preceq$
	33333333333333	1416 (8)	219 (4)	360 (3)	$\preceq$
	23131312323213	1390 (10)	210 (3)	353 (3)	
Map 3	11111111111111	1373 (6)	221 (3)	301 (7)	$\preceq$
	22222222222222	1378 (5)	211 (3)	268 (5)	$\preceq$
	33333333333333	1385 (6)	203 (4)	291 (7)	$\emptyset$
	11122222112231	1358 (9)	219 (7)	263 (12)	
Map 4	11111111111111	1383 (6)	291 (5)	565 (5)	$\preceq$
	22222222222222	1385 (7)	304 (4)	542 (4)	$\preceq$
	33333333333333	1423 (6)	273 (4)	583 (5)	$\emptyset$
	11121131212112	1360 (4)	282 (5)	540 (4)	
Map 5	11111111111111	1405 (7)	467 (4)	559 (4)	$\preceq$
	22222222222222	1431 (9)	447 (4)	541 (4)	$\preceq$
	33333333333333	1467 (9)	411 (5)	567 (5)	$\emptyset$
	21311213111211	1397 (11)	448 (10)	535 (5)	
Map 6	11111111111111	1575 (7)	549 (5)	303 (4)	$\preceq$
	22222222222222	1626 (9)	540 (6)	286 (5)	$\preceq$
	33333333333333	1703 (11)	499 (4)	316 (7)	$\emptyset$
	31121312111111	1570 (16)	535 (10)	266 (6)	
Map 7	11111111111111	1434 (5)	599 (6)	284 (6)	$\preceq$
	22222222222222	1475 (10)	602 (5)	243 (6)	$\preceq$
	33333333333333	1489 (12)	549 (3)	264 (6)	$\emptyset$
	11332211321332	1401 (12)	563 (4)	230 (12)	
Map 8	11111111111111	1761 (9)	254 (5)	382 (3)	$\preceq$
	22222222222222	1804 (10)	269 (4)	357 (4)	$\preceq$
	33333333333333	1826 (10)	230 (3)	392 (7)	$\emptyset$
	23221131313323	1747 (11)	247 (9)	363 (9)	
Map 9	11111111111111	2501 (14)	926 (6)	574 (9)	$\preceq$
	22222222222222	2503 (10)	921 (10)	524 (8)	$\preceq$
	33333333333333	2641 (14)	833 (5)	574 (14)	$\emptyset$
	21132331333223	2463 (19)	891 (8)	523 (9)	
Map 10	11111111111111	1430 (8)	630 (4)	205 (2)	$\preceq$
	22222222222222	1493 (13)	615 (4)	209 (2)	$\preceq$
	33333333333333	1542 (10)	554 (4)	229 (5)	$\emptyset$
	11311111322231	1418 (9)	623 (9)	197 (2)	

TABLE III: MO-PTSP Results with different weights. The last column indicates if the evolved individual dominates ($\preceq$) or not ($\emptyset$) each one of the base genomes for that particular map.

MO-MCTS Improvements

Transposition Tables

- Use hash tables to store representative nodes for equivalent locations.
- In DST, these are nodes with the same position and depth in the search tree.
- This avoids redundant computation on equivalent parts of the tree.

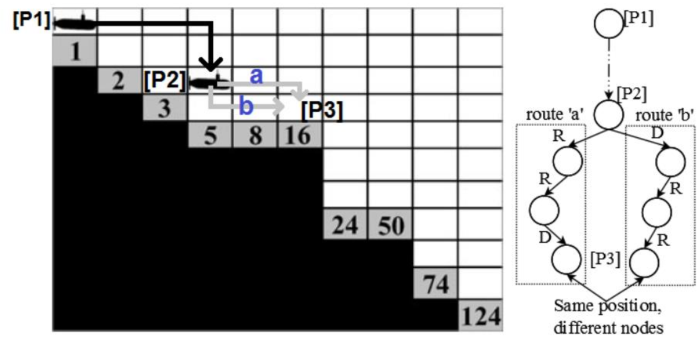


Fig. 7: Example of two different sequences of actions (R: Right, D: Down) that lie in the same position in the map, but at a different node in the tree.

Macro-Actions

- Repeat a given action during L consecutive time steps.
- This allows for additional computation time and lets the algorithm see farther into the future.
- In MO-PTSP, the macro-action size is $L = 15$.

Conclusions

Monte Carlo Tree Search is a leading anytime method for searching large sequential decision spaces.

Multiple-objective problems can have many different solutions. A weight vector can give flexibility in the decision-maker's preferences.

MO-MCTS is able to find more of the non-dominated solutions than single objective MCTS or NSGA-II.

The algorithm has not been tested on problems with many objectives. Computing the hypervolume indicator can be difficult in high dimensions.

Thank You