AI & Robotics: Research

Tree Search & POMDPs

Marc Toussaint
Technical University of Berlin
Summer 2020

Outline

- Tree Search Basics
- Stochastic Domains: MDPs
- Partial Observability: POMDPs
- Reparameterization Trick & DESPOT

Motivation

- Tree Search might sound boring, but it is essential to AI:
  - The concept of decision trees: What is the decision space?
  - State-of-the-art solvers for Go, POMDPs, Mixed-Integer Programming, CSPs, planning in logical domains, etc.
  - Tree Search is the scaffolding – the real smarts is in the heuristic

- **Heuristics & bounds** are fundamental in AI & Optimization!
  - Intelligent heuristis are a major part of classical AI research
  - What are good abstractions & simplifications?
  - Machine Learning is often used to learn heuristics or evaluation functions
  - Related to: Branch & Bound for Mixed-Integer Programming, Multi-Bound Tree Search in Logic Geometric Programming, Angelic Semantics, Optimism (=lower bound) in the face of Uncertainty (e.g., in bandits)

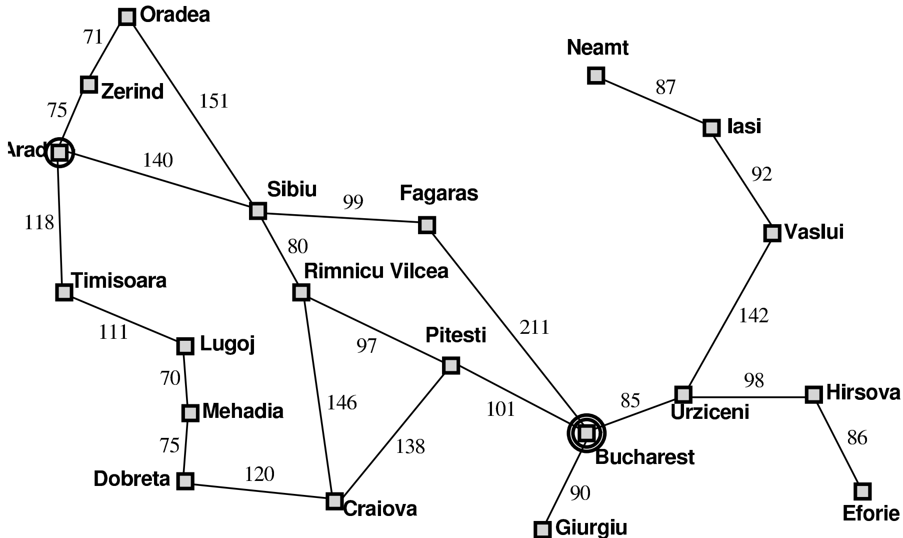
Basic Tree Search

Deterministic, fully observable search problem

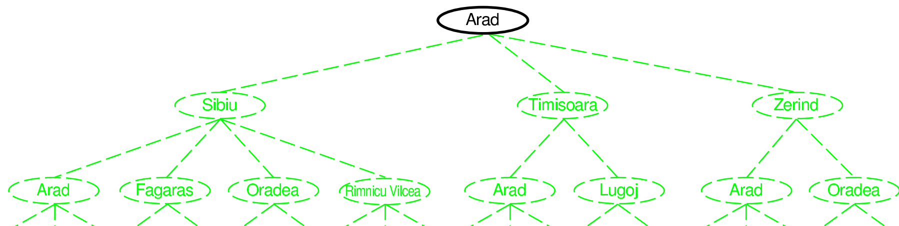
- A *deterministic, fully observable search problem* is defined by four items:
  - initial state $s_0 \in \mathcal{S}$
  - successor function $succ : \mathcal{S} \to \mathcal{S}^*$ ($\to$ set of decisions)
  - goal states $\mathcal{S}_{\text{goal}} \subseteq \mathcal{S}$
  - step cost function $cost(s, s')$, assumed to be ≥ 0

- A *solution* is a sequence of decisions leading from s_0 to a goal
- An *optimal solution* is a solution with minimal path costs

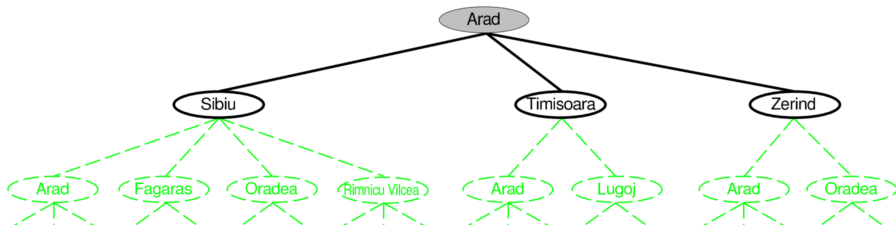
Example: Romania



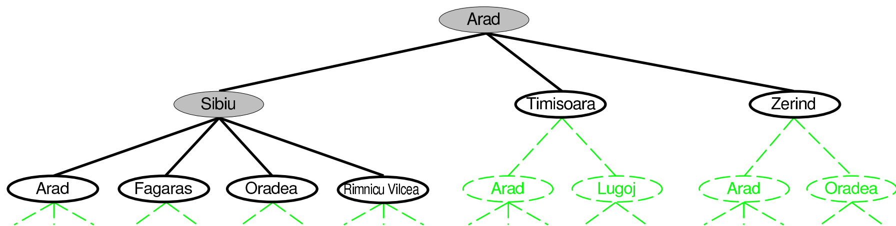
Tree search example



Tree search example



Tree search example



Generic tree search

- We build a decision tree:
  - A **node** stores: `(state, parent, children, cost-so-far, depth)`
  - A node is *not* 1-to-1 with a state. It is 1-to-1 with a *decision sequence* from start to state!

- Standard tree search algorithms differ in how they organize the **fringe**:

1: root $\leftarrow$ Node(state=s_0, cost=0) , fringe $\leftarrow$ $\langle$root$\rangle$
2: **while** fringe is not empty **do**
3: $n \leftarrow$ fringe.pop() *// pop first node from fringe*
4: if(n.state$\in \mathcal{S}_{\text{goal}}$) **return** n *// goal check*
5: fringe.insert(successors of n) *// expand*
6: if n.children=$\{\}$, destroy it and also check ancestors *// save space*
7: **end while**

Generic tree search

- We build a decision tree:
  - A **node** stores: `(state, parent, children, cost-so-far, depth)`
  - A node is *not* 1-to-1 with a state. It is 1-to-1 with a *decision sequence* from start to state!

- Standard tree search algorithms differ in how they organize the **fringe**:
  - Breadth-first search: fringe is FIFO
  - Depth-first search: fringe is LIFO
  - Depth-limited search: fringe does not insert nodes beyond depth limit
  - Best-first search: fringe is sorted with increasing $f(n)$
  - A*: $f(n) = g(n) + h(n)$, where $g(n) =$ cost-so-far, $h(n) =$ **admissible heuristic** (lower bound of cost-to-go)

Heuristics, lower-bounds, optimism, evaluation functions

- Tree Search is kind-of a fail-safe scaffold to organize decision making (Completeness and optimality guarantees, depending on method)
- For decades it's been understood in AI that the real smarts is in the heuristic, or evaluation function:
  - The better the heuristic or evaluation function, the less search is necessary. In the extreme case: the (learnt) heuristic guides you immediately to the goal – perfect prediction of the solution
  - Check google scholar: ”learning search heuristics” ”learning [search—game] evaluation functions”
  - Using Machine Learning to learn evaluation functions or heuristics is around since the 80ies
  - *Fast Downward* and its decendants have won numerous AI planning/solving competitions – in its default setting it just does A*. But with amazing theory and insight in deriving heuristics from the problem specifications. (Reading recommentations: ”Landmarks, critical paths and abstractions: What's the difference anyway?”)

Heuristics, lower-bounds, optimism, evaluation functions

- Heuristics & lower bounds *under estimate* costs!
  - This is essential to guarantee optimality!
  - Ensure that you checked any possibility that, optimistically, could lead to an optimal solution
  - This optimism can also be expressed probabilistically $\rightarrow$ UCT

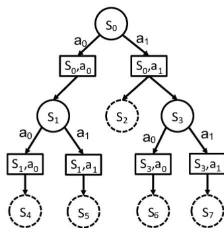
- Evaluation functions
  - is a general term to evaluate states – to estimate cost-to-go or chance-to-win
  - No guarantee of lower bound (non-admissible) $\rightarrow$ no guarantee of optimality
  - But good prediction $\rightarrow$ good (near optimal) decisions with little search

Stochastic Domains

Markov Decision Process

- We need to distinguish between actions and state transitions!

- **Markov Decision Process (MDP)**:
  - initial state distribution $P(s_0)$
  - transition probabilities $P(s' \mid s, a)$
  - reward probabilities $P(r \mid s, a)$
  - deterministic *policy* $a = \pi(s)$



- The tree has action branchings, and state-transition branchings
- cost-to-go(s) $\to$ **value**(s)
  (Value at state s: Expected discounted return, or chance-to-win.)

Value function, Q-function

- The *value function* is the *expected* discounted return when starting in state s and then following policy π:

$$V^\pi(s) = \mathsf{E}_\pi\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s\} ,$$

  with *discounting factor* $\gamma \in [0, 1]$

- The *state-action value function* (or Q**-function**) is the expected discounted return when starting in state s and taking first action a, and then following π:

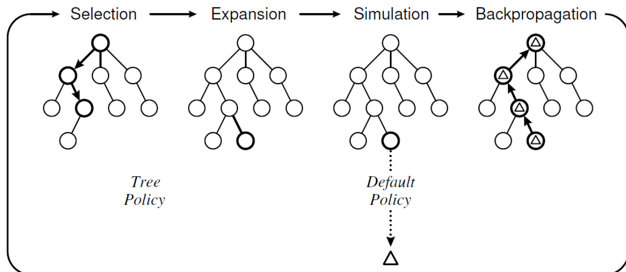
$$Q^\pi(s,a) = \mathsf{E}_\pi\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid s_0 = s, a_0 = a\}$$

  (Note: $V^\pi(s) = Q^\pi(s, \pi(s))$.)

- Optimal: $V^*(s) = V^{\pi^*}(s) , \quad Q^*(s,a) = Q^{\pi^*}(s,a)$

Monte-Carlo Tree Search (MCTS)

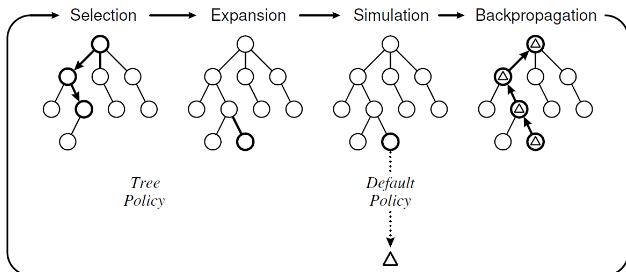
- "Monte-Carlo" $\leftrightarrow$ get value estimates from rollouts
  ($\sim$ learn a Q-function on the fly from sample rollouts)



from Browne et al.

- The **nodes** have different semantics in two consecutive layers:
  - In MDPs: action branching, state transition branching
  - In 2-player games: my-action branching, opponent-action branching
  - In POMDPs: action branching, observation branching

Monte-Carlo Tree Search



from Browne et al.

1: start tree $V = \{v_0\}$
2: **while** within computational budget **do**
3: $v_l \leftarrow$ TREEPOLICY(V) chooses and creates a new leaf of V
4: append v_l to V
5: $\Delta \leftarrow$ ROLLOUTPOLICY(V) rolls out a full simulation, with return Δ
6: BACKUP(v_l, Δ) updates the values of all parents of v_l
7: **end while**
8: return best child of v_0

Generic MCTS scheme

- MCTS typically computes full roll outs to a terminal state. A heuristic (evaluation function) to estimate the utility of a state is not needed, but can be incorporated.
- The tree grows unbalanced
- The TREEPOLICY decides where the tree is expanded – and needs to trade off exploration vs. exploitation
- The ROLLOUTPOLICY is necessary to simulate a roll out. It typically is a random policy; at least a randomized policy.

Upper Confidence Tree (UCT)

- UCT uses UCB (Upper Confidence Bound) to realize the $\textsc{TreePolicy}$, i.e. to decide where to expand the tree – it uses **Bandit** methods to explore the decision space

- BACKUP updates all parents of v_l as
  $n(v) \leftarrow n(v) + 1$ (count how often has it been played)
  $Q(v) \leftarrow Q(v) + \Delta$ (sum of rewards received)

- TREEPOLICY chooses child nodes based on UCB:

$$\operatorname*{argmax}_{v' \in \partial(v)} \frac{Q(v')}{n(v')} + \beta \sqrt{\frac{2 \ln n(v)}{n(v')}}$$

  or choose v' if $n(v') = 0$

Literature

- MCTS became a standard for games, rather simple to implement
- Can be extremely strong when combined with strong evaluation function estimates (NNs)

- Key paper:
  Kocsis & Szepesvári: *Bandit based Monte-Carlo Planning*, ECML 2006.
- Survey paper:
  Browne et al.: *A Survey of Monte Carlo Tree Search Methods*, 2012.
- Tutorial presentation:
  `http://web.engr.oregonstate.edu/~afern/icaps10-MCP-tutorial.ppt`

Partial Observability

Partially Observable Markov Decision Process

- A POMDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{Z}, \mathcal{T}, \mathcal{O}, \mathcal{R} \rangle$
  - $\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}$ are the same as in MDP
  - $\mathcal{Z}$ is observation space.
  - $\mathcal{O}$ is observation function: $\mathcal{O}(s', a, z) = P(z \mid s', a)$

- The actual state $s \in \mathcal{S}$ is unobservable to the agent
  $\Rightarrow$ the policy can only take the **history** as input

$$\pi : \ (a_{o:t\text{-}1}, z_{0:t\text{-}1}) \mapsto a_t$$

- The history uniquely determines the **belief**

$$\begin{aligned} b_t(s') = P(s'|b, a, z) &= \eta \ P(z|s', a, b) \ P(s'|b, a) \\ &= \eta \ \mathcal{O}(s', a, z) \ \sum_s P(s'|s, a) \ b_{t\text{-}1}(s) \\ &= \eta \ \mathcal{Z}(s', a, z) \ \sum_s \mathcal{T}(s, a, s') \ b_{t\text{-}1}(s) \end{aligned}$$

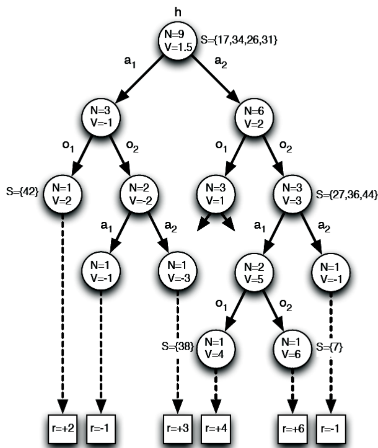
POMDPs

- In essence,
  - the agent can't know in which state s/he is,
  - computing a state-based policy $\pi : \ s \mapsto a$ does not make sense
  - we need to find a policy $\pi : \ (a_{o:t\text{-}1}, z_{0:t\text{-}1}) \mapsto a_t$
  - the history $(a_{o:t\text{-}1}, z_{0:t\text{-}1})$ uniquely identifies a **node** in the action-observation decision tree!
  - In this action-observation decision tree, we want to estimate values!

MCTS applied to POMDPs

Nodes

- store $n(v)$ and $Q(v)$
- represent a history $h(v)$
- have a *set* of states $\mathcal{S}(v)$ attached



from Silver & Veness

MCTS applied to POMDPs

- For each rollout:
  - Choose a *random* initial world state $s_0 \sim \mathcal{S}(v_0)$ and add it to $\mathcal{S}(v_0)$
  - Use a TREEPOLICY to traverse the current tree; during this, update the state sets $\mathcal{S}(v)$ to contain the world state simulated by the simulator
  - Use a ROLLOUTPOLICY to simulate a full rollout
  - Append a new leaf v_l with novel history $h(v_l)$ and a single state $\mathcal{S}(v_l)$ associated

Reparameterization & DESPOT

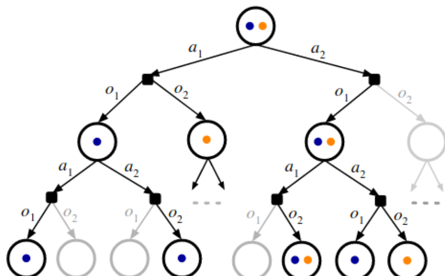
Reparameterization Trick

(The word is usually used in the context of auto-encoders. But the notion is more general.)

- Key idea:
  - The previous models define probability distributions over transitions or observations. Essentially Bayesian networks.
  - The usual semantics is that at each transition, some sampling happens.
  - But the exact same distribution over futures can be generated by just *once* sampling a random seed of a RNG, then using this RNG deterministically at each transition/observation.
  - Every BayesNet can be converted to a deterministic computation graph plus one RNG with initial random seed – they describe the exact same joint distribution.
- In our context:
  - Every rollout is modelled by once sampling a random seed, then deterministically going down the tree.

DESPOT

- Determinized Sparse Partially Observable Tree (DESPOT)
  - Initially sample a fixed set of K *scenarios* ($\sim$ random seeds)
  - All transitions are deterministic in each scenario
  - In one scenario, an action sequence $a_{1:t}$ implies a deterministic experience $(s_0,\ a_1, s_1, z_1,\ a_2, s_2, z_2,\ \dots)$
  - The DESPOT tree is formally defined to include all such experiences for all action sequences
  - Each node in the tree is visited by some scenarios – this set represents the belief
  - The “observation branching” has at most K branches



DESPOT

- Structurally, *Anytime Regularized DESPOT* (AR-DESPOT) works like MCTS
  - But instead of estimating values simply as averages, it maintains rigorous upper and lower bounds of the *optimal* value $\hat{V}_{\pi^*}(b)$, where $\hat{\cdot}$ means estimated under K scenarios.
  - ...and provides theory on the how $\hat{V}_{\pi^*}(b)$ bounds $V_{\pi^*}(b)$

DESPOT

Algorithm 1 AR-DESPOT

1: Set b_0 to the initial belief.
2: **loop**
3: $\quad T \leftarrow$ BUILDDESPOT(b_0).
4: $\quad$ Compute an optimal policy π^* for T using (4)
5: $\quad$ Execute the first action of a of π^*.
6: $\quad$ Receive observation z.
7: $\quad$ Update the belief $b_0 \leftarrow \tau(b_0, a, z)$.

BUILDDESPOT(b_0)

1: Sample a set $\mathbf{\Phi}_{b_0}$ of K random scenarios for b_0.
2: Insert b_0 into T as the root node.
3: **while** time permitting **do**
4: $\quad b \leftarrow$ RUNTRIAL(b_0, T).
5: $\quad$ Back up upper and lower bounds for every node on the path from b to b_0.
6: **return** T

RUNTRIAL(b, T)

1: **if** $\Delta(b) > D$ **then**
2: $\quad$ **return** b
3: **if** b is a leaf node **then**
4: $\quad$ Expand b one level deeper, and insert all new nodes into T as children of b.
5: $a^* \leftarrow \arg\max_{a \in A} U(b, a)$.
6: $z^* \leftarrow \arg\max_{z \in Z_{b,a^*}} \text{WEU}(\tau(b, a^*, z))$.
7: $b \leftarrow \tau(b, a^*, z^*)$.
8: **if** $\text{WEU}(b) \geq 0$ **then**
9: $\quad$ **return** RUNTRIAL(b, T)
10: **else**
11: $\quad$ **return** b

Somani, Ye, Hsu, Lee: *DESPOT: Online POMDP planning with regularization*. NIPS 2013

LeTS-Drive

- Essentially DESPOT (but highly parallelized) plus NN heuristic!
- modifies the TreePolicy to be more UCT-like:

$$a^* = \underset{a \in A}{\text{argmax}} \left\{ u(b,a) + c\pi_0(a|x_b)\sqrt{\frac{n(b)}{n(b,a)+1}} \right\}$$