Monte Carlo CoP

by

Michael Färber

22 September, 2016

Introduction

Monte Carlo Tree Search

- Combines tree search with random sampling
- Very successful since the introduction of UCT in 2006
- Applied to many games, most frequently to Go

leanCoP

- First-order ATP by Jens Otten
- Small and easily extensible
- Can we make a version of leanCoP that uses MCTS, i.e. *monteCoP*?

Monte Carlo Tree Search

Bandit problems

- K actions available, giving rewards in $[0, 1]$ with certain probability
- All rewards and probabilities are constant and initially unknown
- Which actions to try?

UCB

Choose action j that maximises: $\overline{X_j} + \sqrt{\frac{2 \ln n}{n_j}}$, where:

- $\overline{X_j}$ is average reward from action j,
- n_j is number of times action j was chosen,
- n is total number of actions chosen.

Monte Carlo Tree Search

Basic algorithm

1. Choose most urgent node in search tree
2. Run a simulation from node.
3. Add new child node for first simulation action
4. Calculate simulation reward
5. Update child node and its ancestors with reward
6. Repeat

UCT

- Use UCB to choose most urgent node
- My implementations (70 lines of Haskell):
  `https://github.com/01mf02/uct`

UCT problem characterisation

- State transition: What states are reachable from a state and with what probabilities should they be chosen?
- Reward: What is the quality of a final state? (Between 0 and 1.)

Example: Travelling salesman problem

- State: The sequence of cities visited
- State transition: The cities not yet visited, weighted by inverse distance to last visited city
- Reward: The distance between the visited cities

monteCoP

CoP family

Previous CoP extensions

- MaLeCoP: leanCoP + machine learning (Prolog)
- contiCoP: leanCoP port to OCaml in continuation-passing style
- FEMaLeCoP: contiCoP + extension clause biasing

New CoPs

- lazyCoP: contiCoP + lazy lists
- stateCoP: lazyCoP + execute single proof steps only
- monteCoP: stateCoP + without backtracking, use UCT to coordinate search

monteCoP as UCT problem

- State: Open subgoals (list of clauses)
- State transition: Valid steps to prove first subgoal
- Reward: Estimated likelihood that open subgoals are provable
- Final state: Reached if simulation reaches certain depth or first subgoal is not provable

Provability estimation

Naive heuristic

- Reward: $1 - \frac{|\text{subgoals left}|}{|\text{subgoals opened}|}$

Frequentist heuristic

- Provability of literal: ratio of successful and tried proofs of literal
- Provability of clause: product of literal provabilities
- Reward: product of clause provabilities

Problem: frequentist reward does not always converge to 1 as final proof approached

Future extensions

- Use clause statistics to bias state transition (similar to FEMaLeCoP)
- Update state transition probabilities based on rewards
- Use better classifiers to estimate reward
- Consider path features when estimating literal provability