A Framework for Simulation-Based Online Planning

Martin Wirsing
In Kooperation mit Lenz Belzner und Rolf Hennicker,
FACS 2015, LNCS 9539, 2015, 1-30

Modellierung Dynamischer und Adaptiver Systeme
WS 2018/19
Autonomous Systems

- Autonomous systems have to adapt to
  - environmental conditions and
  - new requirements at runtime even if they are defined at design time

- ASCENS project
  - 2010-2015, EU-funded Integrated Project
  - 15 partners from 7 countries
  - Developed systematic approach for engineering autonomous ensembles including
    - SW process, formal modeling, verification, monitoring, adaptation, awareness
  - Case studies on robotics, cloud computing, e-mobility
Decision Making under Uncertainty

- Very large state spaces ($|S| > 10^{10}$)
- Probabilistic effects
- Partially uncontrolled environment
- Incomplete design time knowledge
Contents

1. Online planning
2. A generic framework for online planning
3. Simulation-based online planning
   1. The framework
   2. Monte Carlo Tree Search for discrete domains
   3. Cross Entropy for continuous domains:
4. Concluding remarks
1. Online Planning
Online Planning

Real Situation

observe

build State Model and plan

execute

Image sources:
thegrid.soup.io/post/312159914
mobots.epfl.ch/marxbot.html
Online Planning

Real Situation

observe

build State Model and plan

execute

Image sources:
thegrid.soup.io/post/312159914
mobots.epfl.ch/marxbot.html
Online Planning (Informally, Sequential)

while true do
  observe state
  plan
  execute action w.r.t. plan
end while
Online Planning (Informally, Concurrent)

\[
\text{while true do}
\]
\[
\quad \text{observe state}
\]
\[
\quad \text{execute \mid\mid \text{plan}}
\]
\[
\text{end while}
\]
Online Planning: Parameters

- **State space** $S$
- **Action space** $A$
  - Attribute $\text{actionRequired} : \text{Agent} \rightarrow \text{Bool}$
  - Operation $\text{observe} : \text{Agent} \rightarrow S$
- **Operation** $\text{execute} : \text{RealAction} \rightarrow ()$

**Planning**
- Reward function $R : S \rightarrow \mathbb{R}$  =>  $\text{getReward}$
- Strategy $P_{Action}(A \mid S)$  =>  $\text{sampleAction}$
- Planning refines initial strategy according to $R$

**Online planning**
- Iterated execution and planning
Online Planning (Refined)

Agent || Planner

where

\[\text{while true do}\]
\[\quad \text{state} \leftarrow \text{observe}()\]
\[\quad \text{planner.state} \leftarrow \text{state}\]
\[\quad \text{when actionRequired do}\]
\[\qquad \text{actionRequired} \leftarrow \text{false}\]
\[\qquad \text{action} \leftarrow \text{planner.strategy.sampleAction(state)}\]
\[\qquad \text{end when}\]
\[\quad \text{action.real.execute}()\]
\[\text{end while}\]

while true do
\[\text{plan()}\]
end while
Agent || Planner

while true do
    state ← observe()
    planner.state ← state

    when actionRequired do
        actionRequired ← false
        action ← planner.strategy.sampleAction(state)
    end when

    action.real.execute()
end while

while true do
    plan()
end while

Agent

domain specific
Plug Points

Agent || Planner

while true do
  state ← observe()
  planner.state ← state
  when actionRequired do
    actionRequired ← false
    action ← planner.strategy.sampleAction(state)
  end when
  action.real.execute()
end while

while true do
  plan()
end while
A Framework for Online Planning

Observation

Real Situation

Interaction
A Framework for Online Planning

Operation w.r.t. state and strategy

Real Situation

Agent

1. actionRequired : Bool
2. <plug-pt> observe() : State

Planner

1. <plug-pt> plan()

RealAction

1. <plug-pt> execute()

VirtualAction

1. <plug-pt> sampleAction(State) : VirtualAction

Strategy

1. <plug-pt> sampleAction(State) : VirtualAction

RewardFct

1. <plug-pt> getReward(State) : Real

\( P_{Action}(A|S) \)
A Framework for Online Planning

Changes strategy w.r.t. reward function

\[ P_{Action}(A|S) \]
3. Simulation-Based Online Planning
Three Types of State

Real Situation

State Model

Simulation
Approach

- Refine strategy $P_{Action}(A|S)$ by Simulation-Based Planning
  - Provide agent with simulation of itself and domain
  - Generate simulations of future episodes
  - Evaluate simulation episodes wrt. reward function
  - Use estimates to refine simulations
  - Finally: Execute a real action that performed well in simulation
  - Repeat
3.1 The Framework for Simulation-Based Planning

- **State**
  - **Agent**
    - `actionRequired : Bool`
    - `<plug-pt> observe() : State`
  - **Planner**
    - `<plug-pt> plan()`
- **RealAction**
  - `<plug-pt> execute()`
- **VirtualAction**
  - `<plug-pt> execute(State) : State`
- **Strategy**
  - `<plug-pt> sampleAction(State) : VirtualAction`
- **RewardFct**
  - `<plug-pt> getReward(State) : Real`
- **Simulation**
  - `<plug-pt> sampleSuccessor(State, VirtualAction) : State`
The Framework for Simulation-Based Planning

\[ P_{\text{Action}}(A|S) \]

\[ P_{\text{Sim}}(S|S \times A) \]
The Framework for Simulation-Based Planning

\[ P_{\text{Sim}}(S | S \times A) \]

\[ P_{\text{Action}}(A | S) \]
The Framework for Simulation-Based Planning

The framework for simulation-based planning involves simulating with respect to strategy and domain dynamics. The key components include:

- **State**
  - State
  - planner
  - Agent
    - actionRequired : Bool
    - observe() : State
  - Planner
    - plan() 
    - updateStrategy(Episode, Real)
  - SimPlanner
    - plan() 
    - updateStrategy(Episode, Real)

- **RealAction**
  - execute()

- **VirtualAction**
  - execute(State) : State

- **Strategy**
  - sampleAction(State) : VirtualAction
  - getReward()
  - sampleSuccessor(State, VirtualAction) : State

- **P_{Action}(A|S)**

- **P_{Sim}(S|S \times A)**

The diagram illustrates the simulation process with arrows indicating the flow of information and the application of strategies and actions within the simulation framework.
The Framework for Simulation-Based Planning

The diagram illustrates the process of planning and simulation. It shows the interaction between the state, agent, planner, simulation, and actions.

- **State**: Represents the current state of the system.
- **Agent**: Takes actions based on the current state.
- **Planner**: Plans actions given the state and strategy.
- **Simulation**: Simulates the result of an action weighted by episode reward.
- **RealAction**: Represents real-world actions.
- **VirtualAction**: Simulates actions in a virtual environment.

Key equations:

- $P_{Action}(A|S)$: Probability of an action given the state.
- $P_{Sim}(S|S \times A)$: Probability of the simulation result given the state and action.
SBP Parameters

- **Simulation** $P_{Sim}(S \mid S \times A)$
  - Agent’s model/knowledge of domain dynamics
  - Can be changed at runtime
  - May differ from real domain dynamics
  - Can be learned/refined from observations

- **Maximum search depth** $h_{max}$
  - Impacts simulation effort
  - Less simulation steps: Fast but shallow planning
  - Can be dynamically adapted
Simulation-Based Planning Algorithm

\textbf{op} plan() \\
\hspace{1em} \textbf{vars} s, r, episode, a \\
\hspace{1em} s ← state \\
\hspace{1em} r ← rewardFct.getReward(s) \\
\hspace{1em} episode ← nil \\
\hspace{1em} \textbf{for} \ 0\. .. \ h_{max} \ \textbf{do} \\
\hspace{2em} a ← strategy.sampleAction(s) \\
\hspace{2em} s ← simulation.sampleSuccessor(s, a) \\
\hspace{2em} episode ← episode::(s, a) \\
\hspace{2em} r ← r + rewardFct.getReward(s) \\
\hspace{1em} \textbf{end for} \\
\hspace{1em} strategy ← updateStrategy(episode, r) \\
\hspace{1em} \textbf{end op}
Simulation-Based Planning: Plug Points

\[ \text{op} \text{ plan()} \]
\[ \text{vars } s, r, \text{ episode, a} \]
\[ s \leftarrow \text{state} \]
\[ r \leftarrow \text{rewardFct.getReward}(s) \]
\[ \text{episode } \leftarrow \text{nil} \]
\[ \text{for } \theta \ldots h_{\text{max}} \text{ do} \]
\[ \quad a \leftarrow \text{strategy.sampleAction}(s) \]
\[ \quad s \leftarrow \text{simulation}.\text{sampleSuccessor}(s, a) \]
\[ \quad \text{episode } \leftarrow \text{episode}::(s, a) \]
\[ \quad r \leftarrow r + \text{rewardFct.getReward}(s) \]
\[ \text{end for} \]
\[ \text{strategy } \leftarrow \text{updateStrategy}($\text{episode}$, $r$) \]
\[ \text{end op} \]
Simulation-Based Planning: Variants

- Variants define `updateStrategy(Episode, Real)`
  - Vanilla Monte Carlo
  - Genetic Algorithms
  - **Monte Carlo Tree Search**
    - for discrete domains
  - **Cross Entropy Planning**
    - for continuous domains
3.2 Monte Carlo Tree Search for Discrete Domains

- **Strategy as tree**
  - Nodes represent states and action choices
  - Add a node per simulation
  - Aggregate simulation data in nodes
    - Reward and frequency
  - Sample actions w.r.t. aggregated data

Strategy Inside the Tree

- E.g. Upper Confidence Bounds for Trees
- Treat action selection as multiarmed bandit
- Select actions that maximize

\[ UCT_j = X_j + 2C \sqrt{\frac{2 \ln n}{n_j}} \]

Strategy Inside the Tree

- E.g. Upper Confidence Bounds for Trees
- Treat action selection as multiarmed bandit
- Select actions that maximize

$$UCT_j = X_j + 2C \sqrt{\frac{2 \ln n}{n_j}}$$

- $X_j$: Average reward of child node $j$
- $n$: Nr. of episodes from current node
- $n_j$: Nr. of episodes from child node $j$
- $C$: UCT exploration constant

Expand the Tree

- Add a new node
  - When an episode leaves the tree

Strategy Outside the Tree

- Simulate episode to depth $h_{max}$
- Observe result
  - E.g. reward observed
  - Here: 0 or 1

Initial $P_{Action}(A|S)$

Reward: 1
Update Strategy

- Update the statistics
- This changes the strategy inside the tree
Trees Represent Strategies

- MCTS builds a skewed tree
- Tree can be interpreted as $P_{Action}(A|S)$
- Promising parts of the strategy space are preferred
Example Domain

- **Search and Rescue**
  - Victims, fires and ambulances
  - Unknown topology
  - Unknown initial situation

- **Agent actions**
  - Noop, Move
  - Load or drop a victim
  - Extinguish fire if adjacent

- **Noise**
  - Actions may fail
  - Fires ignite and cease

- **Experiment**
  - Monte Carlo Tree Search
  - Large state space (> $10^{12}$)
  - Large branching factor ($2^{18}$)
  - 0.2 seconds/decision
  - $P_{Sim}(S | S \times A)$ models domain perfectly
Experimental Results (I)

- Measured (in %)
  - Victims at ambulance (blue)
  - Victims in a fire (red)
  - Positions on fire (green)
- Provided reward
  - Victim at ambulance: +100
- System synthesized sensible behavior
- Results in 0.95 confidence interval
  - Checked with MultiVeStA


Autonomy
Experimental Results (II)

- Measured (in %)
  - Victims at ambulance (blue)
  - Victims in a fire (red)
  - Positions on fire (green)
- Expose system to **unexpected events**
  - At steps 20, 40, 60, 80
  - All carried victims are dropped
  - New fires break out
  - Events NOT simulated by planner
  - New situation incorporated by planner
- System showed sensible reactions
- Results in 0.95 confidence interval

**Robustness**
Experimental Results (III)

- Measured (in %)
  - Victims at ambulance (blue)
  - Victims in a fire (red)
  - Positions on fire (green)

- **Change system goals** while operating
  - Change of reward function
    - Steps 0-40: Reward for victims not in a fire
    - Steps 40-80: Reward for victims at ambulance
  - Change NOT simulated by planner
  - But planner incorporates new situation

- System adapted behavior wrt. goals
- Results in 0.95 confidence interval

---

**Flexibility**
From Discrete to Continuous Domains

- **Actions**
  - State and action space $= \mathbb{R}^n$
  - E.g. (speed, rotation, duration) for actions

- **Cross Entropy Planning**
  - Approximate (unknown) target distribution
    - Multivariate Gaussian distribution
    - Sample state space (locally) and choose „elite“ samples for updating the strategy („sharpen“ the Gaussian)

- Here: Gaussians over sequences of actions
  - Sequence length = planning depth
Cross Entropy Planning

- White circle represents agent
- Red boxes represent moving victims
- Black lines are simulation episodes
- Action parameters are speed, rotation and duration
- Images show iterations 1, 5 and 10
  - Simulation depth is adaptive here (reduced simulation cost)
  - Note the iterative “shaping” of a promising strategy
Video: Cross Entropy Planning
Video: Cross Entropy Planning

- The video showed *interleaving* planning and execution

- Illustrates iterative shaping of a probabilistic strategy

- When *parallelizing* planning and execution, this looks a little different...
Video: Continuous CE Planning
Cross Entropy Planning Experiments

- CE: Cross Entropy Planning
- TACE: Time Adaptive CE
- C3: Continuous CE Control

- $h$: Planning depth
- $d$: Action duration

![Graph showing time vs. parallel and interleaving execution times for different planning strategies. The graph includes lines for CE with $h = 50$, $d = 1$, TACE with $h \in [2, 50]$, $d \in [0.5, 4]$, and C3 with $h \in [2, 50]$, $d \in [0, 2]$. The graph highlights the differences in execution times between parallel and interleaved execution.]
Concluding Remarks

- **Motivation**
  - Complex dynamic domains
  - High degrees of non-determinism

- **Approach**
  - Model a space of solutions, instead of a single one
  - Online planning: Refine the solution space at runtime wrt. observations and knowledge to determine a currently viable action

- **This Talk**
  - Component framework for Online Planning
    - Parallelization of execution and planning
  - Instantiation: Simulation Based Planning
    - Two examples: MCTS, Cross Entropy Planning

- **Outlook**
  - Model learning of domain dynamics
  - Soft temporal logic for formal (statistical) verification
  - Learning and planning for ensembles
References


