A Framework for Simulation-Based Online Planning

Martin Wirsing
In Kooperation mit Lenz Belzner und Rolf Hennicker,
Eingeladener Vortrag auf FACS 2015, Niteroi

Modellierung Dynamischer und Adaptiver Systeme  WS 2016/17
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)

\[\text{while true do}
\text{observe state}
\text{plan}
\text{execute action w.r.t. plan}
\text{end while}\]
Online Planning (Informally, Concurrent)

\[
\text{while true do}
\begin{align*}
& \text{observe state} \\
& \text{execute } \parallel \text{ plan}
\end{align*}
\text{end while}
\]
Online Planning: Parameters

- State space $S$
- Action space $A$
- Operation $\textit{observe} : Agent \rightarrow S$
- Attribute $\textit{actionRequired} : Agent \rightarrow \text{Bool}$

- Operation $\textit{execute} : \text{RealAction} \rightarrow ()$

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

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

Agent || Planner

where

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

Planner

while true do
  plan()
end while
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

Agent || Planner

where

domain specific
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
A Framework for Online Planning
A Framework for Online Planning

Operates w.r.t. state and strategy

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

Changes strategy w.r.t. reward function

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

Real Situation

State Model

Simulation
3.1 The Framework for Simulation-Based Planning

![Diagram of the framework for simulation-based planning]

- **State**
  - Action: `execute()`

- **Agent**
  - Action Required: `Boolean`
  - Observe: `State`

- **Planner**
  - Plan: `Plan()`

- **SimPlanner**
  - Plan: `Plan()`
  - Update Strategy: `UpdateStrategy(Episode, Real)`

- **RealAction**
  - Execute: `State`

- **VirtualAction**
  - Execute: `State` to `State`

- **Strategy**
  - Sample Action: `VirtualAction`

- **RewardFct**
  - Get Reward: `Real`

- **Simulation**
  - Sample Successor: `State` to `State`
The Framework for Simulation-Based Planning

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

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

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

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

The diagram illustrates the process of simulating with respect to strategy and domain dynamics. It involves the following components:

- **State**: Represents the current state of the system.
- **Agent**: Includes the actionRequired and observe methods.
- **Planner**: Responsible for planning with the plan method.
- **SimPlanner**: Used to plan and update the strategy.
- **RealAction**: Represents the real action with an execute method.
- **VirtualAction**: Represents the virtual action with an execute method.
- **Strategy**: Defined by the sampleAction method.
- **Reward**: Calculated by the getReward method.

Mathematically, the framework involves:

- \( P_{Action}(A|S) \): The probability of action given strategy
- \( P_{Sim}(S|S \times A) \): The probability of simulation given state and action combination
The Framework for Simulation-Based Planning

Simulation result refines strategy

Weighted by episode reward

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

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

```plaintext
op plan()

vars s, r, episode, a

s ← state
r ← rewardFct.getReward(s)
episode ← nil

for 0 .. h_max do
    a ← strategy.sampleAction(s)
    s ← simulation.sampleSuccessor(s, a)
    episode ← episode::(s, a)
    r ← r + rewardFct.getReward(s)

end for

strategy ← updateStrategy(episode, r)

end op
```
Simulation-Based Planning: Plug Points

\begin{verbatim}
\textbf{op} plan()
  \textbf{vars} s, r, episode, a
  s \leftarrow \text{state}
  r \leftarrow \text{rewardFct.getReward}(s)
  episode \leftarrow \text{nil}
  \textbf{for} 0 .. h_{\text{max}} \textbf{do}
    a \leftarrow \text{strategy.sampleAction}(s)
    s \leftarrow \text{simulation.sampleSuccessor}(s, a)
    episode \leftarrow \text{episode}::(s, a)
    r \leftarrow r + \text{rewardFct.getReward}(s)
  \textbf{end for}
  \text{strategy} \leftarrow \text{updateStrategy}(episode, r)
\textbf{end op}
\end{verbatim}
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

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

Exploit observations  Explore solution space

- \( X_j \): 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 $h_{\text{max}}$
- Observe result
  - E.g. reward observed
  - Here: 0 or 1

Initial $P_{\text{Action}}(A|S)$
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 $P_{\text{Action}}(A|S)$
- 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
  - $P_{Sim}(S \mid S \times A)$ 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
Cross Entropy Planning Experiments

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

- h: Planning depth
- d: Action duration

![Graph showing performance comparison between CE, TACE, and C3](image)

- CE $h = 50, d = 1$
- TACE $h \in [2, 50], d \in [0.5, 4]$
- C3 $h \in [2, 50], d \in [0, 2]$

Interleaving

Parallel
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


