Seminar on Automated Planning

Bernhard Bliem & Matti Järvisalo

Practical arrangements, introduction, choosing topics
September 19, 2018
Practical Arrangements
Instructors: Dr. Bernhard Bliem, Prof. Matti Järvisalo
{bernhard.bliem,matti.jarvisalo}@helsinki.fi

Credit units: 5 ECTS

Language: english

WWW: https://courses.helsinki.fi/en/csm12113/124922697

Announcements: seminar webpage, email

Reception: Contact instructor(s) by email for an appointment.
or during seminar meetings
Course Requirements

- Choose a topic (scientific chapter/article) to study
- Write a 10-15 page (plus references) report on the topic
- Give a 40-min presentation on the topic
- Peer-review of the reports of two other students (draft and final versions)
  - or hands-on project work
- Act as the opponent of another student’s presentation
- Actively attend the seminar

Grading:

- On scale 0–5
- Report 40%, presentation 40%, peer-review/project work 20%
- Activity (incl. being an opponent) ±1 grade
Deadlines

- All deadlines are strict; you will fail the course if you do not meet a deadline
  - Need proof of illness to postpone deadlines
  - ... or let us know well in advance to make rearrangements
- Today: choose topic and presentation date
- Presentations: during period II, Nov 14 – Dec 5
- One week before your presentation: preliminary report & slides
  (send to instructors by email)
- At presentation day:
  Presenters: arrive early to set up slides etc.
  Opponents: actively ask questions during & after presentation
- December 12: final reports
- December 18: final peer reviews
Choosing a Topic

Topic = 1-2 articles from the list on the course webpage

- Choose 1-2 articles from the list
- Can suggest a topic+articles outside the list!
- You will likely need to read additional articles for necessary background
- Reserve topic no later than Sep 26 (within one week)
  - Preferably already TODAY!

Note: more background material listed on the course webpage
A seminar report is a short review paper: you explain some interesting results in your own words.

A typical seminar report will consist of the following parts:

- an informal introduction,
- a formally precise definition of the problem that is studied,
- a brief overview of very closely related work: here you might cite approx. 10 papers and explain their main contributions,
- a more detailed explanation of one or two interesting results, with examples
- conclusions.

Superficially, your report should look like a typical scientific article.

However, it will not contain any new scientific results, just a survey of previously published work.
The presentation is an overview of the report
- You should understand what you are saying
- Everyone should understand you
- The abstraction level should be right
- Examples are always good to communicate ideas
Use of Latex especially for the seminar report is strongly encouraged

Latex template for the report available via the seminar webpage

For the presentation, use software of your choice
  ▶ If you use latex, look into the `beamer` package
Some Words of Advice

▶ Start working on your topic early!
▶ Depending on your background, you will very likely need to read additional papers for background
▶ Aim at understanding the key aspects of your topic – do not get side-tracked
▶ You are responsible for figuring out the details
  ▶ The instructors will not teach you all necessary background
  ▶ In case you get completely stuck, contact the instructors
  ▶ You will need to show that you have made a serious attempt to understand the topic by yourself
Introduction
Solving hard problems

- Many practically relevant problems are NP-hard.
- There has been great progress in solving some of them.
- But domain-specific algorithms are only useful for one particular problem.
- Insights cannot easily be transferred to other problems.

Generic solvers

- Capable of solving many problems.
- Provide a declarative language for problem specification.
- Better solvers benefit all modeled problems.

Programming vs. Modeling + Solving
Planning

What is planning?

▶ “Planning is the reasoning side of acting.” [Ghallab et al., 2004]
▶ Which actions does an agent have to execute to reach certain goals?
▶ Which actions allow it to do so most efficiently?

Some Applications

▶ Robots
▶ Autonomous vehicles
▶ Controlling operations in spacecraft
▶ Scheduling of observations at the Hubble Space Telescope
▶ Controlling NPCs in computer games
▶ …
Planning is concerned with finding a sequence of actions that lead from an initial state of a system to a goal state.

Generally a very hard problem

Many algorithms have been proposed.

One of the core disciplines of AI

In real systems (e.g., robots), there is usually an interplay between planning and acting.
Formalisms

We focus on classical planning.

Simplifying assumptions

- Finite number of states
- Fully observable system
  - We know which state we are in.
- Static system
  - Changes are only due to actions.
- No extended conditions on goals
  - No special conditions on, e.g., which trajectory we took.
- Sequential plans
- Implicit time
  - Actions have no duration.

Research on classical planning is important to improve the techniques in more realistic settings.
We can formalize planning tasks with transition systems.

**Definition**

A transition system is a tuple \((S, A, \gamma, i, g)\), where

- \(S\) is the set of **states**, 
- \(A\) is the set of **actions**, 
- \(\gamma : S \times A \rightarrow S\) is a partial function, the **transition function**, 
- \(i \in S\) is the **initial state** and 
- \(g \in S\) is the **goal state**.

Many variants are in place – use depends on application.
Transition systems

We can formalize planning tasks with transition systems.

Definition

A transition system is a tuple \((S, A, \gamma, i, g)\), where

- \(S\) is the set of states,
- \(A\) is the set of actions,
- \(\gamma: S \times A \rightarrow S\) is a partial function, the transition function,
- \(i \in S\) is the initial state and
- \(g \in S\) is the goal state.

Many variants are in place – use depends on application.

We write \(s_0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_n\) if \(s_i = \gamma(s_{i-1}, a_i)\) holds for all \(i\).
Transition systems

We can formalize planning tasks with transition systems.

Definition

A transition system is a tuple \((S, A, \gamma, i, g)\), where

- \(S\) is the set of states,
- \(A\) is the set of actions,
- \(\gamma : S \times A \rightarrow S\) is a partial function, the transition function,
- \(i \in S\) is the initial state and
- \(g \in S\) is the goal state.

Many variants are in place – use depends on application.

We write \(s_0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_n\) if \(s_i = \gamma(s_{i-1}, a_i)\) holds for all \(i\).

A sequence \(\langle a_1, \ldots, a_n \rangle\) of actions is a plan if there are states \(s_0, \ldots, s_n\) such that \(s_0 = i, s_n = g\) and \(s_0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_n\).
Example: Blocks World

Our domain

- We have a table and wooden blocks A, B, C.
- A block can be on another block or on the table.
  - Each possible configuration of blocks is a state.
- We can move single blocks that have no blocks on top.
  - E.g., possible transitions for an action $\alpha$, which lifts A from the table onto B:

\[
\begin{array}{ccc}
A & B & C \\
\end{array}
\xrightarrow{\alpha}
\begin{array}{ccc}
B & C \\
A & \\
\end{array}
\quad \text{and} \quad
\begin{array}{ccc}
A & C \\
B & \\
\end{array}
\xrightarrow{\alpha}
\begin{array}{ccc}
C \\
A & B \\
\end{array}
\]

A possible planning task

How to get from initial state

\[
\begin{array}{cc}
A & B \\
\end{array}
\]

to goal state

\[
\begin{array}{c}
A \\
B \\
C \\
\end{array}
\]
?

Seminar on Automated Planning Autumn 2018
Why not use a path finding algorithm for finding plans?
Why not use a path finding algorithm for finding plans?
Why not use a path finding algorithm for finding plans?

As the instance grows, the number of states explodes.
State Explosion

Why path finding algorithms are not feasible:

<table>
<thead>
<tr>
<th>Blocks</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>73</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>$\sim 6 \cdot 10^7$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>$\sim 7 \cdot 10^{13}$</td>
</tr>
</tbody>
</table>

We need to reason over a compact representation of the transition system.
STRIPS Planning

STRIPS: A formalism for classical planning

We can use it to represent planning tasks in a compact way:

► A state is represented as a set of ground atoms.
   ► Example: \{\text{on\_table}(A), \text{on\_block}(B, A), \text{clear}(B)\}

► Actions are represented by means of operators.
  ► An action is any ground instance of an operator.
  ► An operator is defined by the following statements:
    - **Precondition** When is the action applicable?
    - **Delete list** Which atoms are no longer true afterwards?
    - **Add list** Which atoms additionally become true?

► We just need to store the initial state.

► Other states can then be generated using operators.
STRIPS Example: Blocks World

One possible model:

- **Atoms**: on_table(x), on_block(x, y), clear(x), for blocks x
  - Initial state: {on_table(A), clear(A), on_table(B), on_block(C, A), clear(C)}
  - Goal state: {on_table(C), on_block(B, C), on_block(A, B), clear(A)}

Seminar on Automated Planning Autumn 2018
STRIPS Example: Blocks World

One possible model:

- **Atoms:** on_table(x), on_block(x, y), clear(x), for blocks x
  - Initial state:
    \{on_table(A), clear(A), on_table(B), on_block(C, A), clear(C)\}
  - Goal state:
    \{on_table(C), on_block(B, C), on_block(A, B), clear(A)\}

- **Operator down_from(x, y):**
  - **Precondition** on_block(x, y) ∧ clear(x)
  - **Delete list** on_block(x, y)
  - **Add list** on_table(x), clear(y)
STRIPS Example: Blocks World

One possible model:

- Atoms: on_table(x), on_block(x, y), clear(x), for blocks x
  - Initial state:
    \{on_table(A), clear(A), on_table(B), on_block(C, A), clear(C)\}
  - Goal state:
    \{on_table(C), on_block(B, C), on_block(A, B), clear(A)\}
- Operator down_from(x, y):
  - Precondition: on_block(x, y) ∧ clear(x)
  - Delete list: on_block(x, y)
  - Add list: on_table(x), clear(y)
- Operator up_to(x, y):
  - Precondition: on_table(x) ∧ clear(x) ∧ clear(y)
  - Delete list: on_table(x), clear(y)
  - Add list: on_block(x, y)
Variants of Classical Planning

We introduced classical planning, which is quite restrictive. Sometimes one may want to use a more general formalism:

- not fully observable
- more than one initial state
- more than one goal state
- non-deterministic actions
- actions have costs / durations
- multiple actors
- parallel actions
- ...
Complexity of Planning

We can study the following decision problems for a specification language \( L \) (e.g., STRIPS):

**Plan Existence**

**Input:** A planning problem \( P \in L \)

**Question:** Is there a plan for \( P \)?

Also interesting because we often want optimal plans:

**Short Plan**

**Input:** A planning problem \( P \in L \) and an integer \( k \)

**Question:** Is there a plan with at most \( k \) actions for \( P \)?

For STRIPS, both problems are PSPACE-complete.
Algorithms

- State-space planning
  - Try to reach goal state from initial state
  - State-of-the-art (but only with good heuristics)
  - Main techniques: Forward search and backward search

- Plan-space planning
  - Here, plans are not linear, only a partial order of actions.
  - Graph: Nodes are partial plans, edges are refinements.
  - Plans are easier to understand and check for humans.

- Planning by translation to SAT
- Planning by translation to constraint satisfaction
- Planning-graph techniques
- Situation calculus: Planning in first-order logic
International Planning Competition (IPC)

- Purpose: Find benchmarks, determine state of the art
- Organized every few years since 1998
- Results presented at the International Conference on Planning and Scheduling (ICAPS)
- Planning Domain Definition Language (PDDL) introduced at first IPC

2018: Various tracks in three groups:
- Classical: Optimal, bounded-cost, satisficing, agile
- Probabilistic
- Temporal

There are several related, more specialized competitions.
Choosing Topics & Dates

Algorithms — Heuristics — Representations — Complexity — Competitions — Applications

Or suggest your own topic.
Topic List

Algorithms

1. Comparison of forward-search planners: FF, Fast Downward, LAMA
2. Planning-graph based techniques
3. SAT-based planning and beyond (QBF-based planning)
Topic List

**Algorithms**
1. Comparison of forward-search planners: FF, Fast Downward, LAMA
2. Planning-graph based techniques
3. SAT-based planning and beyond (QBF-based planning)

**Heuristics**
1. Heuristics for state-space planners
2. Choosing among different heuristics
3. Planning by refining solutions of relaxations
**Topic List**

**Algorithms**
1. Comparison of forward-search planners: FF, Fast Downward, LAMA
2. Planning-graph based techniques
3. SAT-based planning and beyond (QBF-based planning)

**Heuristics**
1. Heuristics for state-space planners
2. Choosing among different heuristics
3. Planning by refining solutions of relaxations

**Representations**
1. Extensions of classical planning (e.g., time, uncertainties)
2. Transformation to a non-propositional representation (PDDL $\rightarrow$ FDR)
Complexity

1. Overview of complexity of planning
2. Expressive power of planning formalisms
3. Bounds on time and space
4. Parameterized complexity
Topic List (contd.)

Complexity
1. Overview of complexity of planning
2. Expressive power of planning formalisms
3. Bounds on time and space
4. Parameterized complexity

Competitions
Overview of a planning competition: PDDL language, benchmarks, results
Topic List (contd.)

**Complexity**
1. Overview of complexity of planning
2. Expressive power of planning formalisms
3. Bounds on time and space
4. Parameterized complexity

**Competitions**
Overview of a planning competition: PDDL language, benchmarks, results

**Applications**
Present some interesting applications