Planning

Jussi Rintanen

NICTA, Canberra

August 13, 2008
What is planning?

- **Domain-independent** language for describing search problems.
- **Domain-independent** algorithms for solving search problems.
What is planning?

Definition:

- Choose actions and their ordering, to achieve a pre-defined goal.

Application areas:

- high-level planning for intelligent robots
- autonomous systems: NASA Deep Space One, ...
- production planning
- problem-solving (games like Rubik’s cube)
Why is planning difficult?

- Solutions to simplest planning problems are paths from an initial state to a goal state in the transition graph. Efficiently solvable e.g. by Dijkstra’s algorithm in $O(n \log n)$ time.

  Q: Why don’t we solve all planning problems this way?
  A: State spaces are often huge: $10^9, 10^{12}, 10^{15}, \ldots$ states. Constructing the transition graph explicitly is not feasible!!

- Planning algorithms often are – but are not guaranteed to be – more efficient than the obvious solution method of constructing the transition graph + running e.g. Dijkstra’s algorithm.
Solutions to simplest planning problems are paths from an initial state to a goal state in the transition graph. Efficiently solvable e.g. by Dijkstra’s algorithm in $O(n \log n)$ time.

Q: Why don’t we solve all planning problems this way?

A: State spaces are often huge: $10^9, 10^{12}, 10^{15}, \ldots$ states. Constructing the transition graph explicitly is not feasible!!

Planning algorithms often are – but are not guaranteed to be – more efficient than the obvious solution method of constructing the transition graph + running e.g. Dijkstra’s algorithm.
Why is planning difficult?

- Solutions to simplest planning problems are paths from an initial state to a goal state in the transition graph. Efficiently solvable e.g. by Dijkstra’s algorithm in $O(n \log n)$ time.
- Q: Why don’t we solve all planning problems this way?
- A: State spaces are often huge: $10^9, 10^{12}, 10^{15}, \ldots$ states. Constructing the transition graph explicitly is not feasible!!
- Planning algorithms often are – but are not guaranteed to be – more efficient than the obvious solution method of constructing the transition graph + running e.g. Dijkstra’s algorithm.
Generic search algorithms (A∗, ...) and **automatically derived** heuristics.

**Constraint-based** methods (planning as CSP, SAT)

**Symbolic** methods based on **Binary Decision Diagrams** and similar data structures (symbolic breadth-first search, symbolic heuristic search)
Representation of transition systems

- state = valuation of a finite set of state variables

**Example**

- HOUR : \{0, \ldots, 23\} = 13
- MINUTE : \{0, \ldots, 59\} = 55
- LOCATION : \{51, 52, 82, 101, 102\} = 101
- WEATHER : \{sunny, cloudy, rainy\} = cloudy
- HOLIDAY : \{T, F\} = F

- Any \(n\)-valued state variable can be represented by \(\lceil \log_2 n \rceil\) Boolean (2-valued) state variables.
- Actions change the values of the state variables.
Blocks world
The transition graph for three blocks
Blocks world with Boolean state variables

Example

s(clearA) = 0  s(clearB) = 1  s(clearC) = 1
s(AonB) = 0  s(AonC) = 0  s(AonTABLE) = 1
s(BonA) = 1  s(BonC) = 0  s(BonTABLE) = 0
s(ConA) = 0  s(ConB) = 0  s(ConTABLE) = 1

Not all valuations correspond to an intended state, e.g. if $s(AonB) = 1$ and $s(BonA) = 1$. 
Actions

An action $\langle p, e \rangle$ consists of

1. a **precondition** $v_1 = b_1, \ldots, v_n = b_n$ where $v_i$ are state variables and $b_i$ are 0 or 1,

2. an **effect** $v_1 := b_1, \ldots, v_m := b_m$ where $v_i$ are state variables and $b_i$ are 0 or 1.

Short-hands for conditions:
- $a$ for $a = 1$
- $\neg a$ for $a = 0$

Short-hands for assignments:
- $a$ for $a := 1$
- $\neg a$ for $a := 0$
Example

Action that moves $B$ from $A$ onto $C$:
$\langle\{\text{BonA, clearB, clearC}\}, \{\text{BonC, clearA, } \neg\text{BonA, } \neg\text{clearC}\}\rangle$. 

![Diagram showing the movement of B from A to C]
Actions
The successor state of a state

**Executability**

An action $\langle p, e \rangle$ is **executable in a state** $s$ iff the precondition $p$ is true in $s$.

**Successor states**

The **successor state** $\text{exec}_o(s)$ of $s$ with respect to $o = \langle p, e \rangle$ is obtained from $s$ by making the assignments in $e$.

**Example**

$\langle \{a\}, \{\neg a, b\} \rangle$ is executable in state $s$ that satisfies $a = 1, b = 1, c = 1$ because $s(a) = 1$.

Hence $a = 0, b = 1, c = 1$ hold in $\text{exec}_{\langle \{a\}, \{\neg a, b\} \rangle}(s)$.
Planning problem

Transition system $\langle V, I, A, G \rangle$

- $V$ is a finite set of state variables.
- $I$ is an initial state (a valuation of $V$).
- $A$ is a set of actions over $V$.
- $G$ is a set of conditions over $V$, the goals.
A plan for $\langle V, I, A, G \rangle$ is a sequence $\pi = o_1, \ldots, o_n$ of actions such that $o_1, \ldots, o_n \in A$ and there is a sequence of states $s_0, \ldots, s_n$ (the execution of $\pi$) so that

1. $s_0 = I$,
2. $s_i = \text{exec}_{o_i}(s_{i-1})$ for every $i \in \{1, \ldots, n\}$, and
3. $s_n$ satisfies $G$.

This can be equivalently expressed as

$$\text{exec}_{o_n}(\text{exec}_{o_{n-1}}(\cdots \text{exec}_{o_1}(I) \cdots)) \text{ satisfies } G.$$
Search algorithms: A* Example
Search algorithms: A*

Example
Search algorithms: $A^*$

Example

Diagram showing a search algorithm with states and actions, where $I$ is the initial state and $G$ is the goal state.
Search algorithms: A* 
Example
Search algorithms: $A^*$

Example
Search algorithms: A* Example
Methods for constructing heuristics

Different forms of relaxation:
1. ignore dependencies between variables
2. project to a subset of variables and solve (optimally)
Basic insight: compute a “distance” for each state variable, not individual states.
This can be done in low-polynomial time in the size of the problem instance.
Distance estimation
Tractor example

1. Tractor moves:
   - from 1 to 2: \( T_{12} = \langle \{T1\}, \{T2, \neg T1\} \rangle \)
   - from 2 to 1: \( T_{21} = \langle \{T2\}, \{T1, \neg T2\} \rangle \)
   - from 2 to 3: \( T_{23} = \langle \{T2\}, \{T3, \neg T2\} \rangle \)
   - from 3 to 2: \( T_{32} = \langle \{T3\}, \{T2, \neg T3\} \rangle \)

2. Tractor pushes A:
   - from 2 to 1: \( A_{21} = \langle \{T2, A2\}, \{T1, A1, \neg T2, \neg A2\} \rangle \)
   - from 3 to 2: \( A_{32} = \langle \{T3, A3\}, \{T2, A2, \neg T3, \neg A3\} \rangle \)

3. Tractor pushes B:
   - from 2 to 1: \( B_{21} = \langle \{T2, B2\}, \{T1, B1, \neg T2, \neg B2\} \rangle \)
   - from 3 to 2: \( B_{32} = \langle \{T3, B3\}, \{T2, B2, \neg T3, \neg B3\} \rangle \)
Distance estimation
Tractor example

<table>
<thead>
<tr>
<th>$t$</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Execute $T_{12} = \langle \{T_1\}, \{T_2, \neg T_1\} \rangle$
Distance estimation

Tractor example

<table>
<thead>
<tr>
<th>$t$</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Execute $T_{23} = \langle \{T2\}, \{T3, \neg T2\} \rangle$
Distance estimation

Tractor example

<table>
<thead>
<tr>
<th>$t$</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Execute $A_{32} = \langle \{T_3, A_3\}, \{T_2, A_2, \neg T_3, \neg A_3\} \rangle$
Distance estimation
Tractor example

<table>
<thead>
<tr>
<th>$t$</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Execute $B_{32} = \langle \{T3, B3\}, \{T2, B2, \neg T3, \neg B3\}\rangle$
Distance estimation

Tractor example

<table>
<thead>
<tr>
<th>$t$</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Execute $A_{21} = \langle \{T^2, A_2\}, \{T_1, A_1, \neg T^2, \neg A_2\}\rangle$
### Distance estimation

**Tractor example**

<table>
<thead>
<tr>
<th>$t$</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

**Execute** $B_{21} = \langle \{T2, B2\}, \{T1, B1, \neg T2, \neg B2\} \rangle$
Heuristic estimate of distance of $A_1, B_1$ is 4. Actual distance of $A_1, B_1$ is 8.
Abstraction Heuristics

Key observation

Eliminating any state variable can only reduce the length of the shortest plan.

- Any abstraction, with some variables eliminated, yields a smaller state space.
- Distances in the abstract state space are lower bounds for the distances in the state space itself.
The tractor example, abstracted to \{A_1, A_2, A_3, B_1, B_2, B_3\} (eliminating the tractor) yields actions

1. **Tractor moves:**
   - from 1 to 2: \( T_{12} = \langle \{\}, \{\} \rangle \)
   - from 2 to 1: \( T_{21} = \langle \{\}, \{\} \rangle \)
   - from 2 to 3: \( T_{23} = \langle \{\}, \{\} \rangle \)
   - from 3 to 2: \( T_{32} = \langle \{\}, \{\} \rangle \)

2. **Tractor pushes A:**
   - from 2 to 1: \( A_{21} = \langle \{A_2\}, \{A_1, \neg A_2\} \rangle \)
   - from 3 to 2: \( A_{32} = \langle \{A_3\}, \{A_2, \neg A_3\} \rangle \)

3. **Tractor pushes B:**
   - from 2 to 1: \( B_{21} = \langle \{B_2\}, \{B_1, \neg B_2\} \rangle \)
   - from 3 to 2: \( B_{32} = \langle \{B_3\}, \{B_2, \neg B_3\} \rangle \)

The abstract state space has 9 states (as opposed to 27). Reaching \( A_1, B_1 \) from the abstract initial state \( A_3, B_3 \) takes 4 abstract actions.
Abstraction Heuristics
Aggregation of several abstractions

In practice it is only possible to use abstractions that retain only very few state variables. These typically yield very weak lower bounds.

Useful strategy: aggregate several abstractions.

1. **Maximum** of lower bounds from different abstractions
2. **Sum** of lower bounds from different abstractions, provided that no action gets counted twice.

Central problem: Which abstractions to aggregate?
Conclusions

- Planning: problem-independent solution methods for search problems
- We considered one solution method: state-space search, with problem-independent methods for deriving admissible heuristics.
- Other solution methods exist (CSP, SAT, ...)
- Planning is more general than state-space search:
  - uncertainty (nondeterminism),
  - different types of objectives (goals vs. rewards),
  - rational, real time

leading to infinite state spaces, infinite executions, ...