Planning, STRIPS, and heuristics

Alexander Shleyfman

December 13, 2017
# Introduction
- What is planning?
- Transition system
- Example

## Deterministic planning tasks
- State variables
- Operators
- Deterministic planning tasks
- STRIPS operators

## STRIPS language
- Example

## Delete Relaxation and heuristics
- Motivation
- Definition by example
- The max heuristic $h_{\text{max}}$
What is planning?

- Intelligent decision making: What actions to take?
- General-purpose problem representation.
- Algorithms for solving any problem expressible in the representation.
- Application areas:
  - High-level planning for intelligent robots
  - Autonomous systems: NASA Deep Space One, ...
  - Problem solving (single-agent games like Rubik’s cube)
Why is planning difficult?

- Solutions to classical planning problems are paths from an initial state to a goal state in the transition graph.
  - Efficiently solvable by Dijkstra’s algorithm in $O(|V| \log |V| + |E|)$ time.
  - Why don’t we solve all planning problems this way?

- State spaces may be huge: $10^9, 10^{12}, 10^{15}, \ldots$ states.
  - Constructing the transition graph is infeasible!
  - Planning algorithms try to avoid constructing whole graph.

- Planning algorithms often are – but not guaranteed to be – more efficient than obvious solution methods constructing the transition graph and using e.g. Dijkstra’s algorithm.
Different classes of problems

- **dynamics**: deterministic, nondeterministic, or probabilistic
- **observability**: full, partial, or none
- **horizon**: finite or infinite
- . . .

We will talk about **classical planning**:

1. deterministic dynamics –
   action + current state uniquely determine successor state;
2. fully observable;
3. finite + compact representation.
Determistic dynamics example

Moving objects with a robotic hand: move the green block onto the blue block.
Transition systems

Goal states

Initial state

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z
Transition systems

Formalization of the dynamics of the world/application

Definition (transition system)

A transition system is \( \langle S, I, \{a_1, \ldots, a_n\}, G \rangle \) where

- \( S \) is a finite set of states (the state space),
- \( I \subseteq S \) is a finite set of initial states,
- every action \( a_i \subseteq S \times S \) is a binary relation on \( S \),
- \( G \subseteq S \) is a finite set of goal states.

Definition (applicable action)

An action \( a \) is applicable in a state \( s \) if \( sas' \) for at least one state \( s' \).
Transition systems

Deterministic transition systems

A transition system is deterministic if there is only one initial state and all actions are deterministic. Hence all future states of the world are completely predictable.

Definition (deterministic transition system)

A deterministic transition system is $\langle S, I, O, G \rangle$ where

- $S$ is a finite set of states (the state space),
- $I \in S$ is a state,
- actions $a \in O$ (with $a \subseteq S \times S$) are partial functions,
- $G \subseteq S$ is a finite set of goal states.

Successor state wrt. an action

Given a state $s$ and an action $a$ so that $a$ is applicable in $s$, the successor state of $s$ with respect to $a$ is $s'$ such that $s a s'$, denoted by $s' = app_a(s)$. 
Blocks world
The rules of the game

Location on the table does not matter.

Location on a block does not matter.
Blocks world

The rules of the game

At most one block may be below a block.

At most one block may be on top of a block.
Blocks world
The transition graph for three blocks
Blocks world

Properties

<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>5</td>
<td>501</td>
</tr>
<tr>
<td>6</td>
<td>4051</td>
</tr>
<tr>
<td>7</td>
<td>37633</td>
</tr>
<tr>
<td>8</td>
<td>394353</td>
</tr>
<tr>
<td>9</td>
<td>4596553</td>
</tr>
<tr>
<td>10</td>
<td>58941091</td>
</tr>
</tbody>
</table>

1. Finding a solution is polynomial time in the number of blocks (move everything onto the table and then construct the goal configuration).

2. Finding a shortest solution is NP-complete (for a compact description of the problem).
Succinct representation of transition systems

- More **compact** representation of actions than as relations is often
  - possible because of symmetries and other regularities,
  - unavoidable because the relations are too big.
- Represent different aspects of the world in terms of different **state variables.**  
  \[ \rightsquigarrow \text{A state is a valuation of state variables.} \]
- Represent actions in terms of changes to the state variables.
State variables

- The state of the world is described in terms of a finite set of finite-valued state variables.

Example

- **hour**: \( \{0, \ldots, 23\} = 15 \)
- **minute**: \( \{0, \ldots, 59\} = 10 \)
- **location**: \( \{T1, \ldots, T9, 301, 401\} = T1 \)
- **weather**: \( \{\text{sunny, cloudy, rainy}\} = \text{cloudy} \)
- **holiday**: \( \{T, F\} = F \)

- Any \( n \)-valued state variable can be replaced by \( \lceil \log_2 n \rceil \) Boolean (2-valued) state variables.
- Actions change the values of the state variables.
Blocks world with state variables

State variables:

- $\text{location-of-A}: \{B, C, \text{table}\}$
- $\text{location-of-B}: \{A, C, \text{table}\}$
- $\text{location-of-C}: \{A, B, \text{table}\}$

Example

\[
\begin{align*}
 s(\text{location-of-A}) &= \text{table} \\
 s(\text{location-of-B}) &= A \\
 s(\text{location-of-C}) &= \text{table}
\end{align*}
\]

Not all valuations correspond to an intended blocks world state, e.g. $s$ such that $s(\text{location-of-A}) = B$ and $s(\text{location-of-B}) = A$. 
Blocks world with Boolean state variables

Example

\[ s(A\text{-on-}B) = 0 \]
\[ s(A\text{-on-}C) = 0 \]
\[ s(A\text{-on-table}) = 1 \]
\[ s(B\text{-on-}A) = 1 \]
\[ s(B\text{-on-}C) = 0 \]
\[ s(B\text{-on-table}) = 0 \]
\[ s(C\text{-on-}A) = 0 \]
\[ s(C\text{-on-}B) = 0 \]
\[ s(C\text{-on-table}) = 1 \]
Operators

Actions for a state set with propositional state variables \( A \) can be concisely represented as operators \( \langle c, e \rangle \) where

- the **precondition** \( c \) is a propositional formula over \( A \) describing the set of states in which the action can be taken (states in which an arrow starts), and

- the **effect** \( e \) describes the successor states of states in which the action can be taken (where the arrows go). Effect descriptions are procedural: how do the values of the state variable change?
Effects (for deterministic operators)

Definition (effects)

(Deterministic) effects are recursively defined as follows:

1. If $a \in A$ is a state variable, then $a$ and $\neg a$ are effects (atomic effects).
2. If $e_1, \ldots, e_n$ are effects, then $e_1 \land \cdots \land e_n$ is an effect (conjunctive effects). The special case with $n = 0$ is the empty conjunction $\top$.

Atomic effects $a$ and $\neg a$ are best understood as assignments $a := 1$ and $a := 0$, respectively.
Blocks world operators

In addition to state variables like $A\text{-}on\text{-}T$ and $B\text{-}on\text{-}C$, for convenience we also use state variables $A\text{-}clear$, $B\text{-}clear$, and $C\text{-}clear$ to denote that there is nothing on the block in question.

\[\langle A\text{-}clear \land A\text{-}on\text{-}T \land B\text{-}clear, \ A\text{-}on\text{-}B \land \neg A\text{-}on\text{-}T \land \neg B\text{-}clear \rangle\]
\[\langle A\text{-}clear \land A\text{-}on\text{-}T \land C\text{-}clear, \ A\text{-}on\text{-}C \land \neg A\text{-}on\text{-}T \land \neg C\text{-}clear \rangle\]
\[\langle A\text{-}clear \land A\text{-}on\text{-}B, \ A\text{-}on\text{-}T \land \neg A\text{-}on\text{-}B \land B\text{-}clear \rangle\]
\[\langle A\text{-}clear \land A\text{-}on\text{-}C, \ A\text{-}on\text{-}T \land \neg A\text{-}on\text{-}C \land C\text{-}clear \rangle\]
\[\langle A\text{-}clear \land A\text{-}on\text{-}B \land C\text{-}clear, \ A\text{-}on\text{-}C \land \neg A\text{-}on\text{-}B \land B\text{-}clear \land \neg C\text{-}clear \rangle\]
\[\langle A\text{-}clear \land A\text{-}on\text{-}C \land B\text{-}clear, \ A\text{-}on\text{-}B \land \neg A\text{-}on\text{-}C \land C\text{-}clear \land \neg B\text{-}clear \rangle\]
\[\ldots\]
Operator semantics

Changes caused by an operator

For each effect $e$ and state $s$, we define the change set of $e$ in $s$, written $[e]_s$, as the following set of literals:

1. $[a]_s = \{a\}$ and $[\neg a]_s = \{\neg a\}$ for atomic effects $a$, $\neg a$
2. $[e_1 \land \cdots \land e_n]_s = [e_1]_s \cup \cdots \cup [e_n]_s$

Applicability of an operator

Operator $\langle c, e \rangle$ is applicable in a state $s$ iff $s \models c$ and $[e]_s$ is consistent.
Operator semantics (ctd.)

Definition (successor state)

The successor state \( \text{app}_o(s) \) of \( s \) with respect to operator \( o = \langle c, e \rangle \) is the state \( s' \) with \( s' \models [e]_s \) and \( s'(v) = s(v) \) for all state variables \( v \) not mentioned in \([e]_s\).
This is defined only if \( o \) is applicable in \( s \).

Example

Consider the operator \( \langle a, \neg a \wedge \neg b \wedge c \rangle \) and the state \( s = \{ a \mapsto 1, b \mapsto 1, c \mapsto 1, d \mapsto 1 \} \).
The operator is applicable because \( s \models a \).
Applying the operator results in the successor state \( \text{app}_{\langle a, \neg a \wedge \neg b \wedge c \rangle}(s) = \{ a \mapsto 0, b \mapsto 0, c \mapsto 1, d \mapsto 1 \} \).
Deterministic planning tasks

Definition (deterministic planning task)

A deterministic planning task is a 4-tuple \( \Pi = \langle A, I, O, G \rangle \) where

- \( A \) is a finite set of state variables,
- \( I \) is an initial state over \( A \),
- \( O \) is a finite set of operators over \( A \), and
- \( G \) is a formula over \( A \) describing the goal states.

Note: We will omit the word “deterministic” where it is clear from context.
Deterministic planning tasks

Mapping planning tasks to transition systems

From every deterministic planning task $\Pi = \langle A, I, O, G \rangle$ we can produce a corresponding transition system $T(\Pi) = \langle S, I, O', G' \rangle$:

1. $S$ is the set of all valuations of $A$,  
2. $O' = \{ R(o) \mid o \in O \}$ where $R(o) = \{ (s, s') \in S \times S \mid s' = \text{app}_o(s) \}$, and
3. $G' = \{ s \in S \mid s \models G \}$. 

Alexander Shleyfman
Planning, STRIPS, and heuristics
December 13, 2017 24 / 48
**Definition**

An operator \( \langle c, e \rangle \) is a **STRIPS operator** if

1. \( c \) is a conjunction of literals, and
2. \( e \) is a conjunction of atomic effects.

Hence every STRIPS operator is of the form

\[
\langle l_1 \land \cdots \land l_n, \ l'_1 \land \cdots \land l'_m \rangle
\]

where \( l_i \) are literals and \( l'_j \) are atomic effects.

**Note:** Many texts also require that all literals in \( c \) are positive.
Why STRIPS is interesting

- STRIPS operators are particularly simple, yet expressive enough to capture general planning problems.
- In particular, STRIPS planning is no easier than general planning problems.
- Most algorithms in the planning literature are only presented for STRIPS operators (generalization is often, but not always, obvious).
Transformation to STRIPS

- Not every operator is equivalent to a STRIPS operator.
- However, each operator can be transformed into a set of STRIPS operators whose “combination” is equivalent to the original operator. (How?)
- However, this transformation may exponentially increase the number of required operators. There are planning tasks for which such a blow-up is unavoidable.
- There are polynomial transformations of planning tasks to STRIPS, but these do not preserve the structure of the transition system (e.g., length of shortest plans may change).
STRIPS

STanford Research Institute Problem Solver – is an automated planner developed by Richard Fikes and Nils Nilsson in 1971 at SRI International. The same name was later used to refer to the formal language of the inputs to this planner.

This language includes:

- World description (State space)
- Legal moves (Actions)
- Initial position (Initial state)
- Desired outcome (Goal states)
Planning task in STRIPS is a quadruple $\langle P, A, I, G \rangle$:

- $P$: finite set of **predicates**
- $A$: finite set of **actions** of a form $\langle \text{pre}, \text{add}, \text{del} \rangle$
  - (preconditions/add effects/delete effects: subsets of predicates)
- $I$: **initial state**
  - (subset of predicates, others assumed to be false)
- $G$: **goal description**
  - (subset of predicates)
STRIPS Example

A package (P) is on a track (T) that goes from A to B. The goal (G) is to have the package at B.
$P$: set of \textit{binary predicates} that can be either \textit{True} or \textit{False}.
STRIPS Example

$P$: set of binary predicates that can be either True or False

Predicates for Package $P_A$, $P_B$, $P_T$
Predicates for Track $T_A$, $T_B$
A: set of actions:
STRIPS Example

A: set of actions:

- DriveAB
- DriveBA
- LoadA
- LoadB
- UnloadA
- UnloadB
STRIPS Example

A: set of actions:

- DriveAB
- LoadA
- UnloadA
- DriveBA
- LoadB
- UnloadB

Example of action presentation:

\[
\begin{align*}
\text{LoadA} &= \begin{cases} 
pre = \{P_A, T_A\} \\
add = \{P_T\} \\
del = \{P_A\}
\end{cases}, \\
\text{DriveAB} &= \begin{cases} 
pre = \{T_A\} \\
add = \{T_B\} \\
del = \{T_A\}
\end{cases}
\end{align*}
\]
STRIPS Example

$I$: initial state

\[ \begin{align*}
P_A & \\
P_B & \\
P_T & \\
T_A & \\
T_B & 
\end{align*} \]
I: initial state

Example of application of LoadA
STRIPS Example

$I$: initial state

Example of application of LoadA

$G$: goal state

– each state that contains the predicate $P_B$
Coming up with heuristics in a principled way

General procedure for obtaining a heuristic

Solve an easier version of the problem.

Two common methods:
- relaxation: consider less constrained version of the problem
- abstraction: consider smaller version of real problem

Both have been very successfully applied in planning. Here we consider relaxation.
Relaxing a problem

How do we relax a problem?

Example (Route planning for a road network)
The road network is formalized as a weighted graph over points in the Euclidean plane. The weight of an edge is the road distance between two locations.

A relaxation drops constraints of the original problem.

Example (Relaxation for route planning)
Use the Euclidean distance \( \sqrt{|x_1 - y_1|^2 + |x_2 - y_2|^2} \) as a heuristic for the road distance between \((x_1, x_2)\) and \((y_1, y_2)\).
This is a lower bound on the road distance (\(\sim\) admissible).

\(\sim\) We drop the constraint of having to travel on roads.
Relaxations for planning

- Relaxation is a general technique for heuristic design:
  - **Straight-line heuristic** (route planning): Ignore the fact that one must stay on roads.
  - **Manhattan heuristic** (15-puzzle): Ignore the fact that one cannot move through occupied tiles.
- We want to apply the idea of relaxations to planning.
- Informally, we want to ignore **bad side effects** of applying operators.

Example (FreeCell)

If we move a card $c$ to a free tableau position, the **good effect** is that the card formerly below $c$ is now available.
The **bad effect** is that we lose one free tableau position.
What is a good or bad effect?

**Question:** Which operator effects are good, and which are bad?

Difficult to answer in general, because it depends on context:

- Locking the entrance door is **good** if we want to keep burglars out.
- Locking the entrance door is **bad** if we want to enter.

We will now consider a reformulation of planning tasks that makes the distinction between good and bad effects obvious.
## Relaxed planning tasks

### Definition (relaxation of operators)
Let \( o = \langle pre, add, del \rangle \) be a STRIPS operator, we define a delete relaxed operator as \( o^+ := \langle pre, add, \emptyset \rangle \).
The set of all delete relaxed operators will be denoted by \( O^+ = \{ o^+ | o \in O \} \).

### Definition (relaxation of planning tasks)
The relaxation \( \Pi^+ \) of a planning task \( \Pi = \langle A, I, O, G \rangle \) in positive normal form is the planning task \( \Pi^+ := \langle A, I, O^+, G \rangle \).

### Definition (relaxation of operator sequences)
The relaxation of an operator sequence \( \pi = o_1 \ldots o_n \) is the operator sequence \( \pi^+ := o_1^+ \ldots o_n^+ \).
Example: delete relaxation intuition

Figure: Applying DriveAB\(^+\) to the initial state \(I\).
Example: delete relaxation intuition

Figure: Applying DriveAB$^+$ to the initial state I.
Relaxed planning tasks: terminology

- Planning tasks in positive normal form without delete effects are called **relaxed planning tasks**.
- Plans for relaxed planning tasks are called **relaxed plans**.
- If $\Pi$ is a planning task in positive normal form and $\pi^+$ is a plan for $\Pi^+$, then $\pi^+$ is called a **relaxed plan for $\Pi$**.
Short summary

Lemma

\textit{If } \pi \textit{ is a plan for } \Pi, \textit{ then } \pi^+ \textit{ is a plan for } \Pi^+.\]

\implies \text{Relaxations of plans are relaxed plans.}

\implies \text{Relaxations are no harder to solve than the original task.}

\implies \text{Optimal relaxed plans are never longer than optimal plans for original tasks.}
Hardness of optimal relaxed planning

Theorem (optimal relaxed planning is hard)

The problem of deciding whether a given relaxed planning task has a plan of length at most $K$ is NP-complete.

Proof.

For membership in NP, guess a plan and verify. It is sufficient to check plans of length at most $|A|$, so this can be done in nondeterministic polynomial time.

For hardness, we reduce from the set cover problem.
Sliding Tile Puzzle 2X2

\[ P = \begin{cases} 
1_{1,1}, 1_{12}, 1_{21}, 1_{22}, \\
2_{1,1}, 2_{12}, 2_{21}, 2_{22}, \\
3_{1,1}, 3_{12}, 3_{21}, 3_{22}, \\
b_{1,1}, b_{12}, b_{21}, b_{22}
\end{cases} \]

\[ O = \begin{cases} 
U(1_{21}), U(1_{22}), U(2_{21}), U(2_{22}), U(3_{21}), U(3_{22}), \\
D(1_{11}), D(1_{12}), D(2_{11}), D(2_{12}), D(3_{11}), D(3_{12}), \\
L(1_{12}), L(1_{22}), L(2_{12}), L(2_{22}), L(3_{12}), L(3_{22}), \\
R(1_{11}), R(1_{12}), R(2_{11}), R(2_{12}), R(3_{11}), R(3_{12}),
\end{cases} \]

\[ I = \{3_{1,1}, 2_{12}, 1_{22}, b_{21}\} \]

\[ G = \{1_{1,1}, 2_{12}, 3_{21}\} \]
Sliding Tile Puzzle: Operators

\[ U(1_{21}) = \langle \{ b_{1,1}, 1_{21} \}, \{ b_{21}, 1_{1,1} \}, \{ b_{1,1}, 1_{21} \} \rangle \]

\[ U^+(1_{21}) = \langle \{ b_{1,1}, 1_{21} \}, \{ b_{21}, 1_{1,1} \}, \emptyset \rangle \]
Sliding Tile Puzzle: Relaxed Planning Graph
Sliding Tile Puzzle: Relaxed Planning Graph
Sliding Tile Puzzle: Relaxed Planning Graph
Sliding Tile Puzzle: Relaxed Planning Graph
Sliding Tile Puzzle: Relaxed Planning Graph
Sliding Tile Puzzle: Relaxed Planning Graph
Sliding Tile Puzzle: Relaxed Planning Graph
Sliding Tile Puzzle: Relaxed Planning Graph
Sliding Tile Puzzle: Relaxed Planning Graph
Sliding Tile Puzzle: Relaxed Planning Graph
The max heuristic $h_{\text{max}}$

Forward cost heuristics: max heuristic $h_{\text{max}}$

Combination rule for action nodes (AND nodes):

- $\text{cost}(u) = \max\{\text{cost}(v_1), \ldots, \text{cost}(v_k)\} + \text{c}(u)$
  
  (with $\max(\emptyset) := 0$)

Combination rule for atom nodes (OR nodes):

- $\text{cost}(u) = \min\{\text{cost}(v_1), \ldots, \text{cost}(v_k)\}$

In both cases, $\{v_1, \ldots, v_k\}$ is the set of true successors of $u$.

Intuition:

- **action rule**: If we have to achieve several conditions, estimate this by the most expensive cost.
- **atom rule**: If we have a choice how to achieve a condition, pick the cheapest possibility.
- the node $G$ is an AND node of a zero cost.
Sliding Tile Puzzle: $h_{\text{max}}$
Sliding Tile Puzzle: $h_{\text{max}}$
Sliding Tile Puzzle: $h_{\text{max}}$
Sliding Tile Puzzle: $h_{\text{max}}$
Sliding Tile Puzzle: $h_{\text{max}}$
Sliding Tile Puzzle: $h_{max}$
Sliding Tile Puzzle: $h_{\text{max}} = 2$
Comparison of relaxation heuristics

Lemma (admissibility of $h_{\text{max}}$)

Let $s$ be a state of planning task $\langle A, I, O, G \rangle$. Then:

$$h_{\text{max}}(s) \leq h^+(s) \leq h^*(s)$$

Note: Inspect the heuristic $h_{\text{add}}$, which almost similar to $h_{\text{max}}$, but the max function is replaced with $\sum$ over the precondition atoms of an actions.
Some heuristics, such as $h_{\text{max}}$, can be also computed directly for the recursive definition.

Let $F$ be a set of atoms:

$$h_c(F) = C\{ \text{cost}(p) \mid p \in F \},$$

$$\text{cost}(o) = C(\text{pre}(o)) + c(o),$$

$$\text{cost}(p) = \begin{cases} 0 & \text{if } p \in s, \\ \min \{ \text{cost}(p) \mid p \in \text{add}(o) \} & \text{else.} \end{cases}$$

Where $C$ is a function on a set, such as max or $\sum$. This values can be computed via dynamic programming, Dijkstra, or other methods.