INTRODUCTION TO AI STRIPS PLANNING

.. and Applications to Video-games!
Course overview

- Lecture 1: Game-inspired competitions for AI research, AI decision making for non-player characters in games
- Lecture 2: STRIPS planning, state-space search
- Lecture 3: Planning Domain Definition Language (PDDL), using an award winning planner to solve Sokoban
- Lecture 4: Planning graphs, domain independent heuristics for STRIPS planning
- Lecture 5: Employing STRIPS planning in games: SimpleFPS, iThinkUnity3D, SmartWorkersRTS
- Lecture 6: Planning beyond STRIPS
STRIPS planning

- STRIPS! So why do we like this formalism?
STRIPS planning

- STRIPS! So why do we like this formalism?
  - Simple formalism for representing planning problems
  - Easy to compute applicable actions
    - Check whether the list of preconditions is a subset of the state description: \( \text{PRECONDITIONS} \subseteq S \)
  - Easy to compute the successor states
    - Add the list of positive effects to the state description and remove the list of negative effects:
      \[
      S' = (S \setminus \text{NEGATIVE-EFFECTS}) \cup \text{POSITIVE-EFFECTS}
      \]
  - Easy to check if the goal is satisfied
    - Check whether the goal is a subset of the state description:
      \( G \subseteq S \)
STRIPS planning

- STRIPS! So why do we like this formalism?
  - It can already describe difficult and complex problems (in more challenging domains than the example we saw)

- ...let’s see how we can solve this kind of problems
Finding a solution to the planning problem following a state-based search

- **Init**: \( \text{On}(A,\text{Table}) \land \text{On}(B,\text{Table}) \land \ldots \)

- **Goal**: \( \text{On}(A,B) \land \ldots \)

- **Action**: \( \text{Move}(b,x,y) \),
  - PRECONDITIONS: \( \text{On}(b,x) \land \ldots \)
  - EFFECTS: \( \text{On}(b,y) \land \ldots \)

- **Action**: \( \text{MoveToTable}(b,x) \),
  - PRECONDITIONS: \( \text{On}(b,x) \land \ldots \)
  - EFFECTS: \( \text{On}(b,\text{Table}) \land \ldots \)
STRIPS planning: state-based search

- Finding a solution to the planning problem following a state-based search
  - Init (where to start from)
  - Goal (when to stop searching)
  - Action (how to generate the “graph”)

- Progression planning: forward state-based search
- Regression planning: backward state-based search
Progression planning

- Start from the initial state
- Check if the current state satisfies the goal
- Compute applicable actions to the current state
- Compute the successor states
- Pick one of the successor states as the current state
- Repeat until a solution is found or the state space is exhausted
Progression planning

- Start from the initial state

On (A, Table)
On (B, Table)
On (C, Table)
Clear (A)
Clear (B)
Clear (C)
Check if the current state satisfies the goal

No
Progression planning

- Compute applicable actions to the current state
  - Action( Move(b,x,y),
    PRECONDITIONS: On(b,x) \land Clear(b) \land Clear(y) )
  - Action( MoveToTable(b,x),
    PRECONDITIONS: On(b,x) \land Clear(b) )
Progression planning

- Compute applicable actions to the current state
  - Action( Move(b,x,y),
    PRECONDITIONS: On(b,x) ∧ Clear(b) ∧ Clear(y))
  - Action( MoveToTable(b,x),
    PRECONDITIONS: On(b,x) ∧ Clear(b))
  - Move(B,Table,C)
    Preconditions:
    - On(B,Table)
    - Clear(B)
    - Clear(C)
  - Applicable action!
Progression planning

- Compute applicable actions to the current state
  - Action (Move(b,x,y),
    PRECONDITIONS: On(b,x) ∧ Clear(b) ∧ Clear(y))
  - Action (MoveToTable(b,x),
    PRECONDITIONS: On(b,x) ∧ Clear(b))
  - Move(A,Table,B)
  - Move(A,Table,C)
  - Move(B,Table,A)
  - Move(B,Table,C)
  - Move(C,Table,A)
  - Move(C,Table,B)

- All these are applicable actions!
Progression planning

- Compute applicable actions to the current state

- Action( Move(b,x,y),
  PRECONDITIONS: On(b,x) \land Clear(b) \land Clear(y) )

- Action( MoveToTable(b,x),
  PRECONDITIONS: On(b,x) \land Clear(b) )

- Move(B,Table,B)
  Preconditions:
  - On(B,Table)
  - Clear(B)

- This is also an applicable action!
Progression planning

- Compute applicable actions to the current state
  - Action( Move(b,x,y),
    PRECONDITIONS: On(b,x) \land Clear(b) \land Clear(y) )
  - Action( MoveToTable(b,x),
    PRECONDITIONS: On(b,x) \land Clear(b) )
  - MoveTT(B,Table)
    Preconditions:
    - On(B,Table)
    - Clear(B)
    This is also an applicable action!
Progression planning

- Compute applicable actions to the current state

  - **Action** (Move(b,x,y),
    
    PRECONDITIONS: On(b,x) \& Clear(b) \& Clear(y))

  - **Action** (MoveToTable(b,x),
    
    PRECONDITIONS: On(b,x) \& Clear(b))

- All these are applicable also actions!
Progression planning

- Compute the successor states
  - **Action**: Move(b,x,y),
    
    EFFECTS: On(b,y) ∧
    
    Clear(x) ∧
    
    ¬On(b,x) ∧
    
    ¬Clear(y)
  
  - **Move(B,Table,C)**
    
    Effects:
    
    - On(B,C)
    
    - Clear(Table)
    
    - ¬On(B,Table)
    
    - ¬Clear(C)
Progression planning

- Compute the successor states

Move(A, Table, C) → Move(B, Table, C)
Move(A, Table, B) → Move(B, Table, A)
Move(C, Table, B) → Move(C, Table, A)

Move(A, Table, C) → Move(A, Table, B)
Move(A, Table, C) → Move(C, Table, B)
Move(B, Table, C) → Move(B, Table, A)
Move(B, Table, C) → Move(C, Table, A)
Move(B, Table, A) → Move(C, Table, A)
Move(C, Table, B) → Move(C, Table, A)
Progression planning

- Pick one of the successor states as current..
Progression planning

..and repeat!
Progression planning

- Pick one of the successor states as current
Progression planning

- Is it guaranteed that progressive planning will find a solution if one exists?
Progression planning

- Is it guaranteed that progressive planning will find a solution if one exists?

  - Given that the state-space is finite (ground atoms, no function symbols, finite number of constants)...

  - ..Yes! As long as we visit each state only once..
Progression planning

- Pick one of the **not-visited** successor states as current
Progression planning

- Is it guaranteed that progressive planning will find a solution if one exists?

  - Given that the state-space is finite (ground atoms, no function symbols, finite number of constants).

  - Yes! As long as we visit each state only once.

- But it may have to explore the whole state-space
Progression planning
Progression planning

- Unlike this simple example, in many problems the state space is actually huge.
- Even this simple example becomes challenging if we consider 100 boxes and 1000s of applicable actions of the form Move(b,x,y) in each state.

- Similar to search problems we can make use of heuristics that help progression planning pick the most promising states to investigate first.
Heuristics for progression planning

- A* search
- Evaluation function \( f(s) = g(s) + h(s) \)
  - \( g(s) \): the number of actions needed to reach state \( s \) from the initial state (accurate)
  - \( h(s) \): the number of actions needed to reach a goal state from state \( s \) (estimated)

- Use \( f(s) \) to order the successor states and pick the most promising one.
Consider a heuristic $h(s)$ with the following behavior:

- $Move(A, Table, C)$
  - $g(s)=1$
  - $h(s)=2$

- $Move(A, Table, B)$
  - $g(s)=1$
  - $h(s)=1$

- $Move(C, Table, B)$
  - $g(s)=1$
  - $h(s)=2$

- $Move(B, Table, C)$
  - $g(s)=1$
  - $h(s)=1$

- $Move(B, Table, A)$
  - $g(s)=1$
  - $h(s)=2$

- $Move(C, Table, A)$
  - $g(s)=1$
  - $h(s)=2$
So why is it different than the usual search problems?

E.g., in grid-based problems we could define $h(s)$ using a “relaxed” distance metric such as Manhattan distance.
Heuristics for progression planning

- So why is it different than the usual search problems?
  - E.g., in grid-based problems we could define $h(s)$ using a “relaxed” distance metric such as Manhattan distance.

- The action schemas provide valuable information that can be used to specify domain independent heuristic functions!

- Moreover they provide a logical specification of the problem that allows approaches for finding a solution that are different than search
Heuristics for progression planning

- Empty list of preconditions
  - $h(s) =$ number of actions needed to achieve the goal if we assume that all actions are always applicable

- Empty list of negative effects
  - $h(s) =$ number of actions needed to achieve the goal if we disregard the negative effects of actions

- Planning graphs

- Simple example: $h(s) =$ number of literals in the goal that are missing from $s$
Heuristics for progression planning

- Start from the initial state
- Check if the current state satisfies the goal
- Compute applicable actions to the current state
- Compute the successor states
- Pick one the most promising of the successor states as the current state
- Repeat until a solution is found or the state space is exhausted
Heuristics for progression planning

- Compute the successor states

- Move(A, Table, C)
  - \( g(s) = 1 \)
  - \( h(s) = 2 \)

- Move(A, Table, B)
  - \( g(s) = 1 \)
  - \( h(s) = 1 \)

- Move(A, Table, C)
  - \( g(s) = 1 \)
  - \( h(s) = 1 \)

- Move(C, Table, B)
  - \( g(s) = 1 \)
  - \( h(s) = 2 \)

- Move(C, Table, A)
  - \( g(s) = 1 \)
  - \( h(s) = 1 \)
Heuristics for progression planning

- Compute the successor states

- Move(A,Table,C)
  - $g(s)=1$
  - $h(s)=2$

- Move(A,Table,B)
  - $g(s)=1$
  - $h(s)=1$

- Move(C,Table,B)
  - $g(s)=1$
  - $h(s)=2$

- Move(B,Table,C)
  - $g(s)=1$
  - $h(s)=1$

- Move(B,Table,A)
  - $g(s)=1$
  - $h(s)=2$

- Move(C,Table,A)
  - $g(s)=1$
  - $h(s)=2$
Heuristics for progression planning

- Pick the most promising successor state \(\text{wrt } f(s)\)

- \(\text{Move}(A, \text{Table}, C)\)
  - \(g(s) = 1\)
  - \(h(s) = 2\)

- \(\text{Move}(A, \text{Table}, B)\)
  - \(g(s) = 1\)
  - \(h(s) = 1\)

- \(\text{Move}(C, \text{Table}, B)\)
  - \(g(s) = 1\)
  - \(h(s) = 2\)

- \(\text{Move}(B, \text{Table}, C)\)
  - \(g(s) = 1\)
  - \(h(s) = 1\)

- \(\text{Move}(B, \text{Table}, A)\)
  - \(g(s) = 1\)
  - \(h(s) = 2\)

- \(\text{Move}(C, \text{Table}, A)\)
  - \(g(s) = 1\)
  - \(h(s) = 2\)
Heuristics for progression planning

- Compute the successor states

- Move(A, Table, C)
  - $g(s) = 1$
  - $h(s) = 2$

- Move(C, Table, A)
  - $g(s) = 2$
  - $h(s) = 1$

- Move(C, Table, B)
  - $g(s) = 1$
  - $h(s) = 2$

- Move(B, Table, C)
  - $g(s) = 1$
  - $h(s) = 1$

- Move(B, Table, A)
  - $g(s) = 1$
  - $h(s) = 2$

- Move(C, Table, A)
  - $g(s) = 1$
  - $h(s) = 2$
Heuristics for progression planning

Compute the successor states

- Move(A, Table, C)
  - $g(s) = 1$
  - $h(s) = 2$

- Move(C, Table, A)
  - $g(s) = 2$
  - $h(s) = 1$

- Move(C, Table, B)
  - $g(s) = 1$
  - $h(s) = 2$

- Move(B, Table, C)
  - $g(s) = 1$
  - $h(s) = 1$

- Move(B, Table, A)
  - $g(s) = 1$
  - $h(s) = 2$

- Move(C, Table, A)
  - $g(s) = 1$
  - $h(s) = 2$
Heuristics for progression planning

- Pick the most promising successor state \( \text{wrt } f(s) \)

Move(\(A\),Table,\(C\))
\[ g(s) = 1 \]
\[ h(s) = 2 \]

Move(\(C\),Table,\(A\))
\[ g(s) = 2 \]
\[ h(s) = 1 \]

Move(\(C\),Table,\(B\))
\[ g(s) = 1 \]
\[ h(s) = 2 \]
Heuristics for progression planning

- Compute the successor states

Move(A, Table, C)
  - g(s) = 1
  - h(s) = 2

Move(C, Table, A)
  - g(s) = 2
  - h(s) = 1

Move(C, Table, B)
  - g(s) = 1
  - h(s) = 2

Move(A, Table, B)
  - g(s) = 2
  - h(s) = 0

Move(B, Table, A)
  - g(s) = 1
  - h(s) = 2

Move(C, Table, A)
  - g(s) = 1
  - h(s) = 2
Heuristics for progression planning

- Compute the successor states

<table>
<thead>
<tr>
<th>Move (A, Table, C)</th>
<th>g(s) = 1</th>
<th>h(s) = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move (C, Table, A)</td>
<td>g(s) = 2</td>
<td>h(s) = 1</td>
</tr>
<tr>
<td>Move (C, Table, B)</td>
<td>g(s) = 1</td>
<td>h(s) = 2</td>
</tr>
<tr>
<td>Move (A, Table, B)</td>
<td>g(s) = 2</td>
<td>h(s) = 0</td>
</tr>
</tbody>
</table>

- Move (B, Table, A) | g(s) = 1 | h(s) = 2 |
- Move (C, Table, A) | g(s) = 1 | h(s) = 2 |
Heuristics for progression planning

- Pick the most promising successor state wrt $f(s)$

Move(A, Table, C)
- $g(s) = 1$
- $h(s) = 2$

Move(B, Table, A)
- $g(s) = 1$
- $h(s) = 2$

Move(C, Table, A)
- $g(s) = 2$
- $h(s) = 1$

Move(C, Table, B)
- $g(s) = 1$
- $h(s) = 2$

Move(C, Table, A)
- $g(s) = 1$
- $h(s) = 2$
Regression planning

- Start from the **goal** as current goal
- Check if the **initial state** satisfies the current goal
- Compute the **relevant** and **consistent** actions for the current goal
- Compute the **predecessor** states
- Pick one of the **predecessor** states as the current goal
- Repeat until a solution is found or the state space is exhausted
Research in STRIPS planning

- Planning Domain Definition Language (PDDL)
  - Formal language for specifying planning problems
  - Formal syntax similar to a programming language
  - Includes STRIPS and ADL, and many more features

  Provides the ground for performing a direct comparison between planning techniques and evaluating against classes of problems
Research in STRIPS planning

- Planning Domain Definition Language (PDDL)
  - International Planning Competition 1998 – today

- SAT Plan
- TL Plan
- FF
- BlackBox
- SHOP2
- TALPlanner
- ...

Planning problems in PDDL, e.g., Blocks world, Storage, Trucks, ...

Direct comparison between planning techniques! E.g., evaluation of heuristics, ...
Research in STRIPS planning

- Planning Domain Definition Language (PDDL)
  - We will see more of PDDL and off-the-shelve planners in Lecture 3
Next lecture

- Lecture 1: Game-inspired competitions for AI research, AI decision making for non-player characters in games
- Lecture 2: STRIPS planning, state-space search
- Lecture 3: Planning Domain Definition Language (PDDL), using an award winning planner to solve Sokoban
- Lecture 4: Planning graphs, domain independent heuristics for STRIPS planning
- Lecture 5: Employing STRIPS planning in games: SimpleFPS, iThinkUnity3D, SmartWorkersRTS
- Lecture 6: Planning beyond STRIPS
Material


References