

Artificial Intelligence Foundation – JC3001

Lecture 30: Algorithms and Heuristics - III

Prof. Aladdin Ayesh (aladdin.ayesh@abdn.ac.uk)

Dr. Binod Bhattarai (binod.bhattarai@abdn.ac.uk)

Dr. Gideon Ogunniye, (g.ogunniye@abdn.ac.uk)

October 2025

Material adapted from:
Russell and Norvig (AIMA Book): Chapter 11 (11.2–11.3)
Jörg Hoffmann (University of Saarland)
Malte Helmert (University of Basel)
Dana Nau (University of Maryland)

Course Progression

- Part 1: Introduction
 - ① Introduction to AI ✓
 - ② Agents ✓
- Part 2: Problem-solving
 - ① Search 1: Uninformed Search ✓
 - ② Search 2: Heuristic Search ✓
 - ③ Search 3: Local Search ✓
 - ④ Search 4: Adversarial Search ✓
- Part 3: Reasoning and Uncertainty
 - ① Reasoning 1: Constraint Satisfaction ✓
 - ② Reasoning 2: Logic and Inference ✓
 - ③ Probabilistic Reasoning 1: BNs ✓
 - ④ Probabilistic Reasoning 2: HMMs ✓
- Part 4: Planning
 - ① Planning 1: Intro and Formalism ✓
 - ② **Planning 2: Algorithms and Heuristics**
 - ③ Planning 3: Hierarchical Planning
 - ④ Planning 4: Stochastic Planning
- Part 5: Learning
 - ① Learning 1: Intro to ML
 - ② Learning 2: Regression
 - ③ Learning 3: Neural Networks
 - ④ Learning 4: Reinforcement Learning
- Part 6: Conclusion
 - ① Ethical Issues in AI
 - ② Conclusions and Discussion

Objectives

- Planning Algorithms
 - Planning Graphs ✓
 - Planning based on Heuristic Search
 - Satisfiability Testing



Outline

1 Relaxed Planning Graphs (RPG)

► Relaxed Planning Graphs (RPG)

► Planning via Satisfiability Testing

Relaxed Planning Graph

1 Relaxed Planning Graphs (RPG)

- A variation of the Planning Graph from Graphplan
 - Alternating fact and action levels
 - Ignores negative effects from actions, no mutex relations
- Fact level F_0 contains the facts that are true in the initial state,
- Action level A_0 contains actions whose preconditions are reached from F_0 ,
- F_1 contains F_0 plus the add effects of the actions in A_0 , and so on.

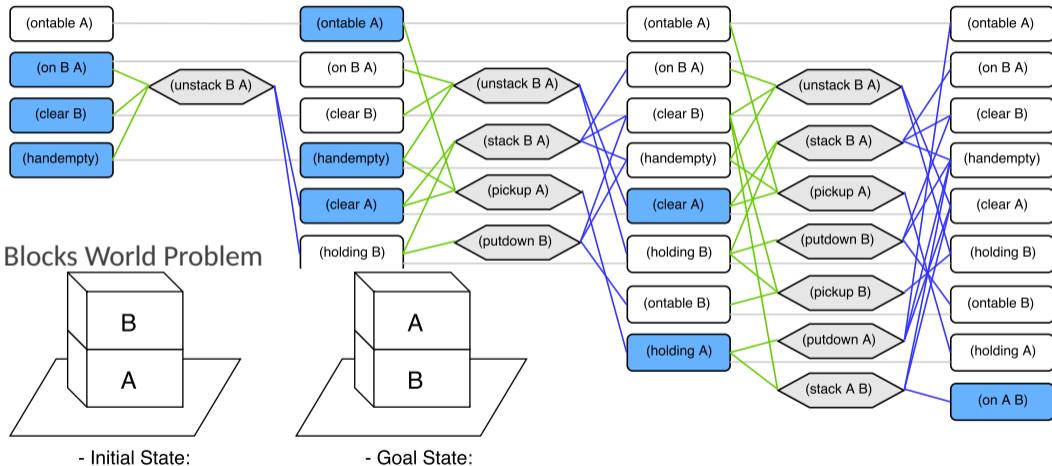
RPG Construction

1 Relaxed Planning Graphs (RPG)

```
1: function BuildFullRPG( $\mathcal{A}, \mathcal{I}, G$ )
2:    $i \leftarrow 0$ 
3:    $\text{RPG.FactLevel}_0 \leftarrow \mathcal{I}$ 
4:   while  $G \not\subseteq \text{RPG.FactLevel}_i$  do
5:      $\text{RPG.ActionLevel}_i \leftarrow \{a \in \mathcal{A} \mid \text{pre}(a) \in \text{RPG.FactLevel}_i\}$ 
6:      $\text{RPG.FactLevel}_{i+1} \leftarrow \text{RPG.FactLevel}_i \cup \text{eff}(a)^+, \forall a \in \text{RPG.ActionLevel}_i$ 
7:     if  $\text{RPG.FactLevel}_{i+1} \equiv \text{RPG.FactLevel}_i$  then
8:       return  $G$  unreachable  $\triangleright$  The algorithm fails if at some point before reaching the facts of the goal no new fact level is added in the graph.
9:      $i \leftarrow i + 1$ 
10:  return RPG
```

Blocks World: RPG

1 Relaxed Planning Graphs (RPG)



(ontable A)
(on B A)
(clear B)
(handempty)

(on A B)

Properties of the RPG

1 Relaxed Planning Graphs (RPG)

- The RPG can be constructed in polynomial time
- Can be used to prove **unsolvability** of a planning problem
 - Goal is unreachable if the relaxed problem reaches a fixed point without reaching the goal
- Provides heuristic estimates of goal distance
 - h^{\max} value is equivalent to the first level in RPG containing G
- Required to extract landmarks

Max Heuristic and (Relaxed) Planning Graph

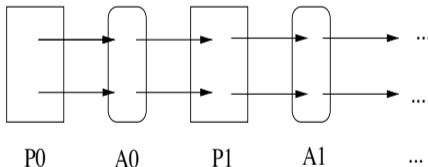
1 Relaxed Planning Graphs (RPG)

- Build reachability graph $P_0, A_0, P_1, A_1, \dots$

$$P_0 = \{p \in s\}$$

$$A_i = \{a \in O \mid \text{Pre}(a) \subseteq P_i\}$$

$$P_{i+1} = P_0 \cup \{p \in \text{Add}(a) \mid a \in A_i\}$$



- Graph implicitly **represents** max heuristic

$$h^{\max}(s) = \min_i \text{ such that } G \subseteq P_i$$

RPG Construction:

1 Relaxed Planning Graphs (RPG)

```
1: function BuildFullRPG( $\mathcal{A}, \mathcal{I}, G$ )
2:    $i \leftarrow 0$ 
3:    $\text{RPG.FactLevel}_0 \leftarrow \mathcal{I}$ 
4:   while  $G \not\subseteq \text{RPG.FactLevel}_i$  do
5:      $\text{RPG.ActionLevel}_i \leftarrow \{a \in \mathcal{A} \mid \text{pre}(a) \in \text{RPG.FactLevel}_i\}$ 
6:      $\text{RPG.FactLevel}_{i+1} \leftarrow \text{RPG.FactLevel}_i \cup \text{eff}(a)^+, \forall a \in \text{RPG.ActionLevel}_i$ 
7:     if  $\text{RPG.FactLevel}_{i+1} \equiv \text{RPG.FactLevel}_i$  then
8:       return  $G$  unreachable  $\triangleright$  The algorithm fails if at some point before reaching the facts of the goal no new fact level is added in the graph.
9:      $i \leftarrow i + 1$ 
10:  return RPG
```

RPG Construction: h^{\max} computation

1 Relaxed Planning Graphs (RPG)

```

1: function BuildFullRPG( $\mathcal{A}, \mathcal{I}, G$ )
2:    $i \leftarrow 0$ 
3:    $\text{RPG.FactLevel}_0 \leftarrow \mathcal{I}$ 
4:   while  $G \not\subseteq \text{RPG.FactLevel}_i$  do
5:      $\text{RPG.ActionLevel}_i \leftarrow \{a \in \mathcal{A} \mid \text{pre}(a) \in \text{RPG.FactLevel}_i\}$ 
6:      $\text{RPG.FactLevel}_{i+1} \leftarrow \text{RPG.FactLevel}_i \cup \text{eff}(a)^+, \forall a \in \text{RPG.ActionLevel}_i$ 
7:     if  $\text{RPG.FactLevel}_{i+1} \equiv \text{RPG.FactLevel}_i$  then
8:       return G-unreachable  $\infty$   $\triangleright$  The algorithm fails if at some point before reaching the
         facts of the goal no new fact level is added in the graph.
9:      $i \leftarrow i + 1$ 
10:  return RPG  $i$ 

```

RPG Construction: h^{\max} computation

1 Relaxed Planning Graphs (RPG)

```
1: function BuildFullRPG( $\mathcal{A}, \mathcal{I}, G$ )
2:    $i \leftarrow 0$ 
3:    $\text{RPG.FactLevel}_0 \leftarrow \mathcal{I}$ 
4:   while  $G \not\subseteq \text{RPG.FactLevel}_i$  do
5:      $\text{RPG.ActionLevel}_i \leftarrow \{a \in \mathcal{A} \mid \text{pre}(a) \in \text{RPG.FactLevel}_i\}$ 
6:      $\text{RPG.FactLevel}_{i+1} \leftarrow \text{RPG.FactLevel}_i \cup \text{eff}(a)^+, \forall a \in \text{RPG.ActionLevel}_i$ 
7:     if  $\text{RPG.FactLevel}_{i+1} \equiv \text{RPG.FactLevel}_i$  then
8:       return  $\infty$   $\triangleright$  If goals are unreachable in the relaxation, the problem is unsolvable
9:      $i \leftarrow i + 1$ 
10:  return  $i$ 
```

Fast Forward Heuristic h^{FF}

1 Relaxed Planning Graphs (RPG)

- A planning heuristic similar to h^{max} , but with no-ops in action levels
 - Inadmissible: sometimes it overestimates goal distance
 - Usually underestimates and provides a closer bound than h^{max}
- Backward extraction algorithm on the relaxed planning graph



Outline

2 Planning via Satisfiability Testing

► Relaxed Planning Graphs (RPG)

► Planning via Satisfiability Testing

Motivation

2 Planning via Satisfiability Testing

- Propositional satisfiability: given a boolean formula
→e.g., $(P \vee Q) \wedge (\neg Q \vee R \vee S) \wedge (\neg R \vee \neg P)$,
does there exist a model
→i.e., an assignment of truth values to the propositions
that makes the formula true?
- This was the very first problem shown to be NP-complete
- Lots of research on algorithms for solving it
 - Known algorithms for solving all but a small subset in average-case polynomial time
- Therefore
 - Try translating classical planning problems into satisfiability problems, and solving them that way

Overall Approach

2 Planning via Satisfiability Testing

- A bounded planning problem is a pair (P, n) :
 - P is a planning problem; n is a positive integer
 - Any solution for P of length n is a solution for (P, n)
- Planning algorithm:
- Do iterative deepening like we did with Graphplan:
 - for $n = 0, 1, 2, \dots$,
 - encode (P, n) as a satisfiability problem Φ
 - If Φ is satisfiable, then
From the set of truth values that satisfies Φ ,
a solution plan can be constructed, so return it and exit

Notation

2 Planning via Satisfiability Testing

- For satisfiability problems we need to use propositional logic
- Need to encode ground atoms into propositions
 - Use $at(r1, loc1)$ itself as the proposition
- Write plans starting at a_0
 - $\pi = \langle a_0, a_1, \dots, a_{n-1} \rangle$
- We consider a function $\gamma(s, a)$ to mean the state resulting from **executing** action a in state s

- If $\pi = \langle a_0, a_1, \dots, a_{n-1} \rangle$ is a solution for (P, n) , it generates these states:
 $s_0, s_1 = \gamma(s_0, a_0), s_2 = \gamma(s_1, a_1), s_n = \gamma(s_{n-1}, a_{n-1})$
- **Fluent:** proposition saying a particular atom is true in a particular state
 - $at(r1, loc1, i)$ is a fluent that is true iff $at(r1, loc1)$ is in s_i
 - We will use l_i to denote the fluent for literal l in state s_i
→e.g., if $l = at(r1, loc1)$
then $l_i = at(r1, loc1, i)$
 - a_i is a fluent saying that a is the i^{th} step of π
→e.g., if $a = move(r1, loc2, loc1)$
then $a_i = move(r1, loc2, loc1, i)$

Encoding Planning Problems

2 Planning via Satisfiability Testing

- Encode (P, n) as a formula Φ such that
 $\pi = \langle a_0, a_1, \dots, a_{n-1} \rangle$ is a solution for (P, n) if and only if
 Φ can be satisfied in a way that makes the fluents a_0, \dots, a_{n-1} true

Encoding Planning Problems

2 Planning via Satisfiability Testing

- Encode (P, n) as a formula Φ such that
 $\pi = \langle a_0, a_1, \dots, a_{n-1} \rangle$ is a solution for (P, n) if and only if
 Φ can be satisfied in a way that makes the fluents a_0, \dots, a_{n-1} true
- Let
 - $A = \{\text{all actions in the planning domain}\}$
 - $S = \{\text{all states in the planning domain}\}$
 - $L = \{\text{all literals in the language}\}$
- Φ is the **conjunct of many** other formulas ...

Formulas in Φ

2 Planning via Satisfiability Testing

- Formula describing the **initial state**:

$$\bigwedge \{l_0 \mid l \in s_0\} \wedge \bigwedge \{\neg l_0 \mid l \in L - s_0\}$$

Formulas in Φ

2 Planning via Satisfiability Testing

- Formula describing the **initial state**:

$$\bigwedge \{l_0 \mid l \in s_0\} \wedge \bigwedge \{\neg l_0 \mid l \in L - s_0\}$$

- Formula describing the **goal**:

$$\bigwedge \{l_n \mid l \in g^+\} \wedge \bigwedge \{\neg l_n \mid l \in g^-\}$$

Formulas in Φ

2 Planning via Satisfiability Testing

- Formula describing the **initial state**:

$$\bigwedge \{l_0 \mid l \in s_0\} \wedge \bigwedge \{\neg l_0 \mid l \in L - s_0\}$$

- Formula describing the **goal**:

$$\bigwedge \{l_n \mid l \in g^+\} \wedge \bigwedge \{\neg l_n \mid l \in g^-\}$$

- For every action a in A , formulas describing what changes a would make if it were the i^{th} step of the plan:

$$a_i \Rightarrow (\bigwedge \{p_i \mid p \in pre(a)\} \wedge \bigwedge \{e_{i+1} \mid e \in eff(a)\})$$

Formulas in Φ

2 Planning via Satisfiability Testing

- Formula describing the **initial state**:

$$\bigwedge \{l_0 \mid l \in s_0\} \wedge \bigwedge \{\neg l_0 \mid l \in L - s_0\}$$

- Formula describing the **goal**:

$$\bigwedge \{l_n \mid l \in g^+\} \wedge \bigwedge \{\neg l_n \mid l \in g^-\}$$

- For every action a in A , formulas describing what changes a would make if it were the i^{th} step of the plan:

$$a_i \Rightarrow (\bigwedge \{p_i \mid p \in pre(a)\} \wedge \bigwedge \{e_{i+1} \mid e \in eff(a)\})$$

- Complete exclusion axiom:

- For all actions a and b in steps i , formulas saying **they cannot occur at the same time**
 $\neg a_i \vee \neg b_i$
- this guarantees there can be only **one action at a time**

Formulas in Φ

2 Planning via Satisfiability Testing

- Formula describing the **initial state**:

$$\bigwedge \{l_0 \mid l \in s_0\} \wedge \bigwedge \{\neg l_0 \mid l \in L - s_0\}$$

- Formula describing the **goal**:

$$\bigwedge \{l_n \mid l \in g^+\} \wedge \bigwedge \{\neg l_n \mid l \in g^-\}$$

- For every action a in A , formulas describing what changes a would make if it were the i^{th} step of the plan:

$$a_i \Rightarrow (\bigwedge \{p_i \mid p \in pre(a)\} \wedge \bigwedge \{e_{i+1} \mid e \in eff(a)\})$$

- Complete exclusion axiom:

- For all actions a and b in steps i , formulas saying **they cannot occur at the same time**
 $\neg a_i \vee \neg b_i$
- this guarantees there can be only **one action at a time**

- Is this enough?

Frame Axioms

2 Planning via Satisfiability Testing

- Frame axioms:
 - Formulas describing what does not change between steps i and $i + 1$
- Several ways to write these
- One way: explanatory frame axioms
 - One axiom for every literal l
 - Says that if l changes between s_i and s_{i+1} , then the action at step i must be responsible:

$$\left(\neg l_i \wedge l_{i+1} \Rightarrow \bigvee_{a \in A} \{a_i \mid l \in \text{eff}^+(a)\} \right) \\ \wedge \left(l_i \wedge \neg l_{i+1} \Rightarrow \bigvee_{a \in A} \{a_i \mid l \in \text{eff}^-(a)\} \right)$$

Example

2 Planning via Satisfiability Testing

- Objects:
r1 - robot
l1, l2 - locations
- Predicates: $\text{at}(R, L)$
- Action: $\text{move}(R, L1, L2)$
pre: $\text{at}(R, L1)$
eff: $\text{not at}(R, L1), \text{at}(R, L2)$

Example

2 Planning via Satisfiability Testing

- **init:** $at(r1, l1, 0) \wedge \neg at(r1, l2, 0)$
- **goal:** $at(r1, l2, 1) \wedge \neg at(r1, l1, 1)$
- **Actions:**
 $move(r1, l1, l2, 0) \Rightarrow at(r1, l1, 0) \wedge at(r1, l2, 1) \wedge \neg at(r1, l1, 1)$
 $move(r1, l2, l1, 0) \Rightarrow at(r1, l2, 0) \wedge at(r1, l1, 1) \wedge \neg at(r1, l2, 1)$
- **Complete exclusion axiom:**
 $\neg move(r1, l1, l2, 0) \vee \neg move(r1, l2, l1, 0)$
- **Explanatory frame axioms:**
 $\neg at(r1, l1, 0) \wedge at(r1, l1, 1) \Rightarrow move(r1, l2, l1, 0)$
 $\neg at(r1, l2, 0) \wedge at(r1, l2, 1) \Rightarrow move(r1, l1, l2, 0)$
 $at(r1, l1, 0) \wedge \neg at(r1, l1, 1) \Rightarrow move(r1, l1, l2, 0)$
 $at(r1, l2, 0) \wedge \neg at(r1, l2, 1) \Rightarrow move(r1, l2, l1, 0)$

Extracting a Plan

2 Planning via Satisfiability Testing

- Suppose we find an assignment of truth values that satisfies Φ .
 - This means P has a solution of length n
- For $i = 1, \dots, n$, there will be exactly one action a such that $a_i = \text{true}$
 - This is the i^{th} action of the plan.

- How to find an assignment of truth values that satisfies Φ ?
 - Use a satisfiability algorithm
- Example: the Davis-Putnam algorithm
 - First need to put Φ into conjunctive normal form
e.g., $\Phi = D \wedge (\neg D \vee A \vee \neg B) \wedge (\neg D \vee \neg A \vee \neg B) \wedge (\neg D \vee \neg A \vee B) \wedge A$
 - Write Φ as a set of clauses (disjuncts of literals)
 $\Phi = \{\{D\}, \{\neg D, A, \neg B\}, \{\neg D, \neg A, \neg B\}, \{\neg D, \neg A, B\}, \{A\}\}$
 - Two special cases:
 - If $\Phi = \emptyset$ then Φ is always true
 - If $\Phi = \{\dots, \emptyset, \dots\}$ then Φ is always false (hence unsatisfiable)

Discussion

2 Planning via Satisfiability Testing

- Recall the overall approach:
 - for $n = 0, 1, 2, \dots$,
 - encode (P, n) as a satisfiability problem Φ
 - If Φ is satisfiable, then
From the set of truth values that satisfies Φ ,
a solution plan can be constructed, so return it and exit
- How well does this work?
 - By itself, not very practical (takes too much memory and time)
 - But it can be combined with other techniques
→e.g., planning graphs

Planning Algorithms Summary

2 Planning via Satisfiability Testing

- Planning Algorithms
 - Planning Graphs
 - Planning based on Heuristic Search
 - Satisfiability Testing

Any Questions.