The Graphplan Planner
• Searching the Planning Graph
Literature

Neoclassical Planning

• concerned with restricted state-transition systems
• representation is usually restricted to propositional STRIPS
• no loss in expressiveness due to lack of functions in STRIPS, but loss of potential
• neoclassical vs. classical planning
  • classical planning: search space consists of nodes containing partial plans
  • every action in a partial plan will appear in the final plan
  • neoclassical planning: nodes can be seen as sets of partial plans
  • actions may appear in final plan; disjunctive planning
• resulted in significant speed-up and revival of planning research
  • speed-up: blocks world: less than 10 blocks to hundreds
Overview

- The Propositional Representation
- The Planning-Graph Structure
- The Graphplan Algorithm

now: the restricted representation used by most neoclassical planning algorithms: propositional STRIPS

- The Planning-Graph Structure
- The Graphplan Algorithm
### Classical Representations

- **propositional representation**
  - world state is set of propositions
  - action consists of precondition propositions, propositions to be added and removed

- **STRIPS representation**
  - like propositional representation, but first-order literals instead of propositions

- **state-variable representation**
  - state is tuple of state variables \(\{x_1, \ldots, x_n\}\)
  - action is partial function over states

---

Classical Representations

- **propositional representation**
  - world state is set of propositions
  - action consists of precondition propositions, propositions to be added and removed

- **STRIPS representation**
  - named after STRIPS planner
  - like propositional representation, but first-order literals instead of propositions
  - most popular for restricted state-transitions systems

- **state-variable representation**
  - state is tuple of state variables \(\{x_1, \ldots, x_n\}\)
  - action is partial function over states
  - useful where state is characterized by attributes over finite domains
  - equally expressive: planning domain in one representation can also be represented in the others
Let \( L = \{ p_1, \ldots, p_n \} \) be a finite set of proposition symbols. A propositional planning domain on \( L \) is a restricted state-transition system \( \Sigma = (S, A, \gamma) \) such that:

- \( S \subseteq 2^L \), i.e. each state \( s \) is a subset of \( L \)
  - \( s \) is set of propositions that currently hold, i.e. \( p \) is true is \( s \) iff \( p \in s \) (closed world)
- \( A \subseteq 2^L \times 2^L \times 2^L \), i.e. each action \( a \) is a triple \((\text{precond}(a), \text{effects}^-(a), \text{effects}^+(a))\) where \( \text{effects}^-(a) \) and \( \text{effects}^+(a) \) must be disjoint
  - preconditions, negative effects, and positive effects
  - \( a \) is applicable in \( s \) iff \( \text{precond}(a) \subseteq s \)
- \( \gamma : S \times A \rightarrow 2^L \) where
  - \( \gamma(s, a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a) \) if \( \text{precond}(a) \subseteq s \)
  - \( \gamma(s, a) = \text{undefined otherwise} \)
- \( S \) is closed under \( \gamma \)
  - if \( s \in S \) then for every applicable action \( a \) \( \gamma(s, a) \in S \)
DWR Example: State Space

• from introduction
DWR Example: Propositional States

- $L\{\text{onpallet, onrobot, holding, at1, at2}\}$
- $S\{s_0, ..., s_5\}$
  - $s_0 = \{\text{onpallet, at2}\}$
  - $s_1 = \{\text{holding, at2}\}$
  - $s_2 = \{\text{onpallet, at1}\}$
  - $s_3 = \{\text{holding, at1}\}$
  - $s_4 = \{\text{onrobot, at1}\}$
  - $s_5 = \{\text{onrobot, at2}\}$

DWR Example: Propositional States

- $L\{\text{onpallet, onrobot, holding, at1, at2}\}$
  - meaning: container is on the ground, container on the robot, crane is holding the container, robot is at location1, robot is at location2
- $S\{s_0, ..., s_5\}$
  - as shown in graph
  - $s_0 = \{\text{onpallet, at1}\}$
  - $s_1 = \{\text{holding, at1}\}$
  - $s_2 = \{\text{onpallet, at1}\}$
  - $s_3 = \{\text{holding, at1}\}$
  - $s_4 = \{\text{onrobot, at1}\}$
  - $s_5 = \{\text{onrobot, at2}\}$
DWR Example: Propositional Actions

- \( a \) : \text{precond}(a), \text{effects}^-(a), \text{effects}^+(a)
  - \( a \) is action name

- \text{take} : \{\text{onpallet}\}, \{\text{onpallet}\}, \{\text{holding}\}
- \text{put} : \{\text{holding}\}, \{\text{holding}\}, \{\text{onpallet}\}
- \text{load} : \{\text{holding,at1}\}, \{\text{holding}\}, \{\text{onrobot}\}
- \text{unload} : \{\text{onrobot,at1}\}, \{\text{onrobot}\}, \{\text{holding}\}
- \text{move1} : \{\text{at2}\}, \{\text{at2}\}, \{\text{at1}\}
- \text{move2} : \{\text{at1}\}, \{\text{at1}\}, \{\text{at2}\}
DWR Example: Propositional State Transitions

- columns: action $a$; rows: state $s$; table cell entry: $\gamma(s,a)$ or empty if action not applicable
  - example: $\gamma(s_0,\text{take})=s_1$
Propositional Planning Problems

- A propositional planning problem is a triple $\mathcal{P}=(\Sigma, s_i, g)$ where:
  - $\Sigma=(S,A,\gamma)$ is a propositional planning domain on $L=\{p_1,\ldots,p_n\}$
  - $s_i \in S$ is the initial state
  - $g \subseteq L$ is a set of goal propositions that define the set of goal states $S_g=\{s \in S \mid g \subseteq s\}$

Propositional Planning Problems

- A propositional planning problem is a triple $\mathcal{P}=(\Sigma, s_i, g)$ where:
  - $\Sigma=(S,A,\gamma)$ is a propositional planning domain on $L=\{p_1,\ldots,p_n\}$
  - $s_i \in S$ is the initial state
  - $g \subseteq L$ is a set of goal propositions that define the set of goal states $S_g=\{s \in S \mid g \subseteq s\}$
  - goal states are implicit in the problem
DWR Example: Propositional Planning Problem

• $\Sigma$: propositional planning domain for DWR domain
  • see previous slides

• $s_i$: any state
  • example: initial state = $s_0 \in S$
  • note: $s_0$ is not necessarily initial state

• $g$: any subset of $L$
  • example: $g=\{\text{onrobot,at2}\}$, i.e. $S_g=\{s_5\}$
Classical Plans

- A plan is any sequence of actions \( \pi = (a_1, \ldots, a_k) \), where \( k \geq 0 \).
  - The length of plan \( \pi \) is \( |\pi| = k \), the number of actions.
  - If \( \pi_1 = (a_1, \ldots, a_k) \) and \( \pi_2 = (a'_1, \ldots, a'_j) \) are plans, then their concatenation is the plan \( \pi_1 \cdot \pi_2 = (a_1, \ldots, a_k, a'_1, \ldots, a'_j) \).
  - The extended state transition function for plans is defined as follows:
    - \( \gamma(s, \pi) = s \) if \( k = 0 \) (\( \pi \) is empty)
    - \( \gamma(s, \pi) = \gamma(\gamma(s, a_1), (a_2, \ldots, a_k)) \) if \( k > 0 \) and \( a_1 \) applicable in \( s \)
    - \( \gamma(s, \pi) = \text{undefined} \) otherwise

Classical Plans

• Note: exactly as for STRIPS case

• A plan is any sequence of actions \( \pi = (a_1, \ldots, a_k) \), where \( k \geq 0 \).
  • The length of plan \( \pi \) is \( |\pi| = k \), the number of actions.
  • If \( \pi_1 = (a_1, \ldots, a_k) \) and \( \pi_2 = (a'_1, \ldots, a'_j) \) are plans, then their concatenation is the plan \( \pi_1 \cdot \pi_2 = (a_1, \ldots, a_k, a'_1, \ldots, a'_j) \).
  • The extended state transition function for plans is defined as follows:
    • \( \gamma(s, \pi) = s \) if \( k = 0 \) (\( \pi \) is empty)
    • \( \gamma(s, \pi) = \gamma(\gamma(s, a_1), (a_2, \ldots, a_k)) \) if \( k > 0 \) and \( a_1 \) applicable in \( s \)
    • \( \gamma(s, \pi) = \text{undefined} \) otherwise
Classical Solutions

• Let $P=(\Sigma, s, g)$ be a propositional planning problem. A plan $\pi$ is a solution for $P$ if $g \subseteq \gamma(s, \pi)$.
  • A solution $\pi$ is redundant if there is a proper subsequence of $\pi$ is also a solution for $P$.
  • $\pi$ is minimal if no other solution for $P$ contains fewer actions than $\pi$.

Classical Solutions

• note: exactly as for STRIPS case

• Let $P=(\Sigma, s, g)$ be a propositional planning problem. A plan $\pi$ is a solution for $P$ if $g \subseteq \gamma(s, \pi)$.
  • A solution $\pi$ is redundant if there is a proper subsequence of $\pi$ is also a solution for $P$.
  • $\pi$ is minimal if no other solution for $P$ contains fewer actions than $\pi$. 
### DWR Example: Plans and Solutions

<table>
<thead>
<tr>
<th>plan $\pi$</th>
<th>$\pi$</th>
<th>$\gamma(s_i,\pi)$</th>
<th>sol.</th>
<th>red.</th>
<th>min.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>0</td>
<td>$s_0$</td>
<td>no</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(move2,move2)</td>
<td>2</td>
<td>undefined</td>
<td>no</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(take,move1)</td>
<td>2</td>
<td>$s_3$</td>
<td>no</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(take,move1,put,move2, take,move1,load,move2)</td>
<td>8</td>
<td>$s_3$</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>(take,move1,load,move2)</td>
<td>4</td>
<td>$s_3$</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>(move1, take, load, move2)</td>
<td>4</td>
<td>$s_3$</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

DWR Example: Plans and Solutions

- as before: $s_i = s_0$; $g = \{\text{onrobot,at2}\}$, i.e. $S_g = \{s_5\}$
The successor function $\Gamma^m : 2^S \rightarrow 2^S$ for a propositional domain $\Sigma = (S, A, \gamma)$ is defined as:

- $\Gamma(s) = \{ \gamma(s, a) | a \in A \text{ and } a \text{ applicable in } s \}$ for $s \in S$
- $\Gamma((s_1, \ldots, s_n)) = \bigcup_{k \in [1,n]} \Gamma(s_k)$
- $\Gamma^0((s_1, \ldots, s_n)) = \{s_1, \ldots, s_n\}$
- $\Gamma^m((s_1, \ldots, s_n)) = \Gamma(\Gamma^{m-1}((s_1, \ldots, s_n)))$

The transitive closure of $\Gamma$ defines the set of all reachable states:

- $\Gamma^\gamma(s) = \bigcup_{k \in [0,\infty]} \Gamma^k(\{s\})$ for $s \in S$
Relevant Actions and Regression Sets

Let $\mathcal{P}=(\Sigma,s_i,g)$ be a propositional planning problem. An action $a\in A$ is relevant for $g$ if

- $g \cap \text{effects}^+(a) \neq \emptyset$ and
- $g \cap \text{effects}^-(a) = \emptyset$.

The regression set of $g$ for a relevant action $a\in A$ is:

- $\gamma^{-1}(g,a) = (g - \text{effects}^+(a)) \cup \text{precond}(a)$
- note: $\gamma(s,a) \in S_g$ iff $\gamma^{-1}(g,a) \subseteq s$

Relevant Actions and Regression Sets

Let $\mathcal{P}=(\Sigma,s_i,g)$ be a propositional planning problem. An action $a\in A$ is relevant for $g$ if

- $g \cap \text{effects}^+(a) \neq \emptyset$ and
- $g \cap \text{effects}^-(a) = \emptyset$.

intuition: $a$ is relevant for $g$ if it can contribute toward producing a state in $S_g$

The regression set of $g$ for a relevant action $a\in A$ is:

- $\gamma^{-1}(g,a) = (g - \text{effects}^+(a)) \cup \text{precond}(a)$
- $\mathcal{P}=(\Sigma,s_i,g)$ has a solution if $\exists a\in A : \mathcal{P}=(\Sigma,s_i,\gamma^{-1}(g,a))$
- note: $\gamma(s,a) \in S_g$ iff $\gamma^{-1}(g,a) \subseteq s$
- $\gamma^{-1}(g,a)$: minimal set of propositions that must hold in a state $s$ from which action $a$ leads to a goal state
Regression Function

- The regression function $\Gamma^{-m}$ for a propositional domain $\Sigma=(S, A, \gamma)$ on $L$ is defined as:
  - $\Gamma^{-1}(g) = \{\gamma^{-1}(g, a) \mid a \in A \text{ is relevant for } g\}$ for $g \in 2^L$
  - $\Gamma^0({g_1, \ldots, g_n}) = \{g_1, \ldots, g_n\}$
  - $\Gamma^{-1}({g_1, \ldots, g_n}) = \bigcup_{k \in \{1, \ldots, n\}} \Gamma^{-1}(g_k)$
  - $\Gamma^{-m}({g_1, \ldots, g_n}) = \Gamma^{-1}(\Gamma^{-m-1}({g_1, \ldots, g_n}))$

- The transitive closure of $\Gamma^{-1}$ defines the set of all regression sets:
  - $\Gamma(g) = \bigcup_{k \in [0, \infty]} \Gamma^{-k}(\{g\})$ for $g \in 2^L$

---

Regression Function

- note: exactly as for STRIPS case

- **The regression function** $\Gamma^{-m}$ for a propositional domain $\Sigma=(S, A, \gamma)$ on $L$ is defined as:
  - $\Gamma^{-1}(g) = \{\gamma^{-1}(g, a) \mid a \in A \text{ is relevant for } g\}$ for $g \in 2^L$
  - $\Gamma^0({g_1, \ldots, g_n}) = \{g_1, \ldots, g_n\}$
  - $\Gamma^{-1}({g_1, \ldots, g_n}) = \bigcup_{k \in \{1, \ldots, n\}} \Gamma^{-1}(g_k)$
  - $\Gamma^{-m}({g_1, \ldots, g_n}) = \Gamma^{-1}(\Gamma^{-m-1}({g_1, \ldots, g_n}))$

- The transitive closure of $\Gamma^{-1}$ defines the set of all regression sets:
  - $\Gamma(g) = \bigcup_{k \in [0, \infty]} \Gamma^{-k}(\{g\})$ for $g \in 2^L$
Statement of a Propositional Planning Problem

A statement of a propositional planning problem is a triple $P=(A,s_i,g)$ where:

- $A$ is a set of actions in an appropriate propositional planning domain $\Sigma=(S,A,\gamma)$ on $L$
- $s_i$ is the initial state in an appropriate propositional planning problem $\mathcal{P}=(\Sigma,s_i,g)$
- $g$ is a set of goal propositions in the same propositional planning problem $\mathcal{P}$

• advantage: statement does not require explicit enumeration of $S$ and $\gamma$
• problem: $L$, $S$ and $\gamma$ are ambiguous
Example: Ambiguity in Statement of a Planning Problem

- statement: $P = \{a_1\}, s_i, g$ where $a_1 = \{(p_1), \{p_1\}, \{p_2\}\}$, $s_i = \{p_1\}$, and $g = \{p_2\}$

- $P_1 = (\Sigma_1, s_i, g)$ where
  - $\Sigma_1 =$
    - $\{(p_1), \{p_2\}\}$
    - $\{a_1\}$
    - $\{(p_1), a_1 \rightarrow (p_2)\}$
  - $L_1 = \{p_1, p_2\}$

- $P_2 = (\Sigma_2, s_i, g)$ where
  - $\Sigma_2 =$
    - $\{(p_1), \{p_2\}, \{p_1, p_3\}, \{p_2, p_3\}\}$
    - $\{a_1\}$
    - $\{(p_1, p_3), a_1 \rightarrow (p_2, p_3)\}$
  - $L_2 = \{p_1, p_2, p_3\}$

- $P$ is statement of planning problem:
  - $P_1 = (\Sigma_1, s_i, g)$ where
    - $\Sigma_1 =$
      - $\{(p_1), \{p_2\}\}$
      - $\{a_1\}$
      - $\{(p_1), a_1 \rightarrow (p_2)\}$
    - $L_1 = \{p_1, p_2\}$

- alternative:
  - $P_2 = (\Sigma_2, s_i, g)$ where
    - $\Sigma_2 =$
      - $\{(p_1), \{p_2\}, \{p_1, p_3\}, \{p_2, p_3\}\}$
      - $\{a_1\}$
      - $\{(p_1, p_3), a_1 \rightarrow (p_2, p_3)\}$
    - $L_2 = \{p_1, p_2, p_3\}$

- $P_2$ plays no role in $P_2$

- regression sets $\Gamma^<\{g\}$ and reachable states $\Gamma^>\{s_i\}$ are identical in $P_1$ and $P_2$
Statement Ambiguity

• Proposition: Let $P_1$ and $P_2$ be two propositional planning problems that have the same statement. Then both, $P_1$ and $P_2$, have
  • the same set of reachable states $\Gamma^>(\{s_i\})$ and
  • the same set of solutions.

• statements are unambiguous enough to be acceptable specifications of planning problems
Properties of the Propositional Representation

- **Expressiveness**: For every propositional planning domain there is a corresponding state-transition system, but what about vice versa?
  - depends on definition of “corresponding”

- **Conciseness**: propositional action representation is concise because it does not mention what does not change
  - truth values of propositions not mentioned in an action do not change through the application of the action, they persist

- **Consistency**: not every assignment of truth values to propositions must correspond to a state in the underlying state-transition system
  - example from DWR domain: state {onrobot,holding,at1,at2} is inconsistent
  - if domain definition and initial state are correct, inconsistent states should not be reachable

- note: state-space and plan-space search still applicable
Grounding a STRIPS Planning Problem

Let $P=(O,s_i,g)$ be the statement of a STRIPS planning problem and $C$ the set of all the constant symbols that are mentioned in $s_i$. Let ground($O$) be the set of all possible instantiations of operators in $O$ with constant symbols from $C$ consistently replacing variables in preconditions and effects.

Then $P'=$(ground($O$),$s_i$,g) is a statement of a STRIPS planning problem and $P'$ has the same solutions as $P$.

• the number of operators will increase exponentially here

• the problems are equivalent (except for exponential increase in size)
Translation: Propositional Representation to Ground STRIPS

Let $P=(A,s_i,g)$ be a statement of a propositional planning problem. In the actions $A$:

- replace every action $(\text{precond}(a), \text{effects}^-(a), \text{effects}^+(a))$ with an operator $o$ with
  - some unique name($o$),
  - $\text{precond}(o) = \text{precond}(a)$, and
  - $\text{effects}(o) = \text{effects}^+(a) \cup \{\neg p \mid p \in \text{effects}^-(a)\}$.

- adds negation sign to negative effects
- result is a statement of a ground STRIPS planning problem
Translation: Ground STRIPS to Propositional Representation

- Let $P=(O,s,g)$ be a ground statement of a classical planning problem.
  - In the operators $O$, in the initial state $s$, and in the goal $g$ replace every atom $P(v_1,\ldots,v_n)$ with a propositional atom $Pv_1,\ldots,v_n$.
  - In every operator $o$:
    - for all $\neg p$ in precond($o$), replace $\neg p$ with $p'$.
    - if $p$ in effects($o$), add $\neg p'$ to effects($o$).
    - if $\neg p$ in effects($o$), add $p'$ to effects($o$).
  - In the goal replace $\neg p$ with $p'$.
  - For every operator $o$ create an action (precond($o$), effects$^-(a)$, effects$^+(a)$).

- problem: operators may contain negated preconditions
- In the operators $O$, in the initial state $s$, and in the goal $g$ replace every atom $P(v_1,\ldots,v_n)$ with a propositional atom $Pv_1,\ldots,v_n$.
- idea: introduce new proposition symbols that represent the negations of existing propositions
- In every operator $o$:
  - for all $\neg p$ in precond($o$), replace $\neg p$ with $p'$.
  - if $p$ in effects($o$), add $\neg p'$ to effects($o$).
  - if $\neg p$ in effects($o$), add $p'$ to effects($o$).
- In the goal replace $\neg p$ with $p'$.
- For every operator $o$ create an action (precond($o$), effects$^-(a)$, effects$^+(a)$).
- result is a statement of a propositional planning problem
Overview

- The Propositional Representation
  - just done: the restricted representation used by most neoclassical planning algorithms: propositional STRIPS

- The Planning-Graph Structure
  - now: defining a new graph that is more efficient to generate and a necessary criterion for solution containment

- The Graphplan Algorithm
Example: Simplified DWR Problem

- figure

initial state:

- 2 locations: loc1 and loc2, connected by path
- 2 robots: robr and robq, both unloaded initially at loc1 and loc2 respectively
- 2 containers: conta and contb, initially at loc1 and loc2 respectively

- robots can load and unload autonomously
- locations may contain unlimited number of robots and containers
- problem: swap locations of containers
Simplified DWR Problem: STRIPS Actions

- **move(r,l,l')**
  
  - move robot r from location l to adjacent location l' (4 possible actions; with rigid adjacent relation evaluated)
  
  - **precond:** at(r,l), adjacent(l,l')
  
  - **effects:** at(r,l'), ¬at(r,l)

- **load(c,r,l)**
  
  - load container c onto robot r at location l (8 possible actions)
  
  - **precond:** at(r,l), in(c,l), unloaded(r)
  
  - **effects:** loaded(r,c), ¬in(c,l), ¬unloaded(r)

- **unload(c,r,l)**
  
  - unload container c from robot r at location l (8 possible actions)
  
  - **precond:** at(r,l), loaded(r,c)
  
  - **effects:** unloaded(r), in(c,l), ¬loaded(r,c)
Simplified DWR Problem: State Proposition Symbols

• idea: represent each atom that may occur in a state by a single (short) proposition symbol

• robots:
  • \(r_1\) and \(r_2\): at(robr,loc1) and at(robr,loc2)
  • \(q_1\) and \(q_2\): at(robq,loc1) and at(robq,loc2)
  • \(ur\) and \(uq\): unloaded(robr) and unloaded(robq)

• containers:
  • \(a_1, a_2, ar, and aq\): in(conta,loc1), in(conta,loc2), loaded(conta,robr), and loaded(conta,robq)
  • \(b_1, b_2, br, and bq\): in(contb,loc1), in(contb,loc2), loaded(contb,robr), and loaded(contb,robq)

• initial state: \(\{r_1, q_2, a_1, b_2, ur, uq\}\)
Simplified DWR Problem: Action Symbols

- move actions:
  - Mr12: move(robr,loc1,loc2), Mr21: move(robr,loc2,loc1), Mq12: move(robq,loc1,loc2), Mq21: move(robq,loc2,loc1)

- load actions:
  - Lar1: load(conta,robr,loc1); Lar2, Laq1, Laq2, Lar1, Lbr2, Lbq1, and Lbq2 correspondingly

- unload actions:
  - Uar1: unload(conta,robr,loc1); Uar2, Uaq1, Uaq2, Uar1, Ubr2, Ubq1, and Ubq2 correspondingly

- 14 state symbols: lower case, italic
- 20 action symbols: uppercase, not italic
Solution Existence

• **Proposition:** A propositional planning problem \( \mathcal{P}=(\Sigma, s_i, g) \) has a solution iff \( S_g \cap \Gamma^-(\{s_i\}) \neq \{\} \).

• **Proposition:** A propositional planning problem \( \mathcal{P}=(\Sigma, s_i, g) \) has a solution iff \( \exists s \in \Gamma^<\{(g)\} : s \subseteq s_i \).

\( \Gamma \) and \( \Gamma^\leq \) are regression sets that denote the set of states that can be reached from the initial state and the set of states that can be reached from some state in the regression set, respectively. Regression sets are a formalization of the planning process, capturing the set of states that can be reached from the initial state by performing a given set of actions.

• … iff there is a goal state that is also a reachable state

• **Proposition:** A propositional planning problem \( \mathcal{P}=(\Sigma, s_i, g) \) has a solution iff \( \exists s \in \Gamma^<\{(g)\} : s \subseteq s_i \).

• … iff there is a minimal set of propositions amongst all regression sets that is a subset of the initial state
Reachability Tree

- tree structure, where:
  - root is initial state $s_i$
  - children of node $s$ are $\Gamma(s)$
  - arcs are labelled with actions

- all nodes in reachability tree are $\Gamma^>(s_i)$
  - all nodes to depth $d$ are $\Gamma^d(s_i)$
  - solves problems with up to $d$ actions in solution

- problem: $O(k^d)$ nodes;
  $k =$ applicable actions per state
DWR Example: Reachability Tree

• [figure]
• corresponds directly to forward-search search tree
• actually: should be graph (corresponding to state space)
Planning Graph: Nodes

- layered directed graph $G = (N, E)$:
  - $N = P_0 \cup A_1 \cup P_1 \cup A_2 \cup P_2 \cup \ldots$
  - layered = each node belongs to exactly one layer
  - proposition and action layers alternate
  - state proposition layers: $P_0$, $P_1$, ...
  - action layers: $A_1$, $A_2$, ...

- first proposition layer $P_0$:
  - propositions in initial state $s_i$: $P_0 = s_i$

- action layer $A_j$:
  - all actions $a$ where: $\text{precond}(a) \subseteq P_{j-1}$

- proposition layer $P_j$:
  - all propositions $p$ where: $p \in P_{j-1}$ or $\exists a \in A_j: p \in \text{effects}^+(a)$
  - propositions at layer $P_j$ are all propositions in the union of all nodes in the reachability tree at depth $j$
  - note: negative effects are not deleted from next layer

- note: $P_{j-1} \subseteq P_j$; propositions in the graph monotonically increase from one proposition layer to the next
Planning Graph: Arcs

• directed and layered = arcs only from one layer to the next

• from proposition $p \in P_{j-1}$ to action $a \in A_j$:
  • if: $p \in \text{precond}(a)$

• from action $a \in A_j$ to layer $p \in P_j$:
  • positive arc if: $p \in \text{effects}^+(a)$
  • negative arc if: $p \in \text{effects}^-(a)$

• no arcs between other layers

• note: $A_{j-1} \subseteq A_j$; actions in the graph monotonically increase from one action layer to the next
Planning Graph Example

• start with initial proposition layer
  • next action layer: applicable action; links from preconditions (black)
  • next proposition layer: previous proposition plus positive effects; links to positive effects (green); links to negative effects (red)
  • next action layer ($A_2$); precondition links; next proposition layer ($P_2$); effect links
  • next action layer ($A_3$); precondition links; next proposition layer ($P_3$); effect links
  • action layers contain “inclusive disjunctions” of actions
Reachability in the Planning Graph

reachability analysis:

- if a goal $g$ is reachable from initial state $s_i$
- then there will be a proposition layer $P_g$ in the planning graph such that $g \subseteq P_g$
- necessary condition, but not sufficient
- low complexity:
  - planning graph is of polynomial size and
  - can be computed in polynomial time

necessary vs. sufficient:

- reachability tree:
  - nodes contain propositions that must necessarily hold
  - propositions in one node are consistent

- planning graph:
  - proposition layers contains propositions that may possibly hold
  - propositions in one layer usually inconsistent (e.g. robots/containers in two places at once)
  - similarly, incompatible actions in one layer may interfere with each other

low complexity:

- planning graph is of polynomial size and
- can be computed in polynomial time
- need more conditions (for sufficient criterion)
Independent Actions: Examples

- **Mr12 and Lar1:**
  - cannot occur together
  - Mr12 deletes precondition \( r_1 \) of Lar1

- **Mr12 and Mr21:**
  - cannot occur together
  - Mr12 deletes positive effect \( r_1 \) of Mr21

- **Mr12 and Mq21:**
  - may occur in same action layer
Independent Actions

• Two actions $a_1$ and $a_2$ are independent iff:
  • $\text{effects}^-(a_1) \cap (\text{precond}(a_2) \cup \text{effects}^+(a_2)) = \{}$ and
  • $\text{effects}^-(a_2) \cap (\text{precond}(a_1) \cup \text{effects}^+(a_1)) = \{}$.

• A set of actions $\pi$ is independent iff every pair of actions $a_1, a_2 \in \pi$ is independent.

Independent Actions

• idea: independent actions can be executed in any order (in same layer)

• Two actions $a_1$ and $a_2$ are independent iff:
  • $\text{effects}^-(a_1) \cap (\text{precond}(a_2) \cup \text{effects}^+(a_2)) = \{}$ and
  • $\text{effects}^-(a_2) \cap (\text{precond}(a_1) \cup \text{effects}^+(a_1)) = \{}$.

• two actions are dependent iff:
  • one deletes a precondition of the other or
  • one deletes a positive effect of the other

• A set of actions $\pi$ is independent iff every pair of actions $a_1, a_2 \in \pi$ is independent.

• note: independence does not depend on planning problem; can be pre-computed

• note: independence relation is symmetrical (follows from definition)
Pseudo Code: independent

function independent(a₁, a₂)
    return true iff the two given actions are independent
    for all p ∈ effects⁻(a₁)
        if p ∈ precond(a₂) or p ∈ effects⁺(a₂) then
            return false
    for all p ∈ effects⁻(a₂)
        if p ∈ precond(a₁) or p ∈ effects⁺(a₁) then
            return false
    return true

• complexity:
  • let b be max. number of preconditions, positive, and negative effects of any action
  • element test in hash-set takes constant time
  • complexity: $O(b)$
Applying Independent Actions

- A set $\pi$ of independent actions is applicable to a state $s$ iff $\bigcup_{a \in \pi} \text{precond}(a) \subseteq s$.
- The result of applying the set $\pi$ in $s$ is defined as:
  \[ \gamma(s, \pi) = (s - \text{effects}^-(\pi)) \cup \text{effects}^+(\pi) \]
  where:
  - $\text{precond}(\pi) = \bigcup_{a \in \pi} \text{precond}(a)$,
  - $\text{effects}^+(\pi) = \bigcup_{a \in \pi} \text{effects}^+(a)$, and
  - $\text{effects}^-(\pi) = \bigcup_{a \in \pi} \text{effects}^-(a)$.

Applying Independent Actions

- A set $\pi$ of independent actions is **applicable** to a state $s$ iff $\bigcup_{a \in \pi} \text{precond}(a) \subseteq s$.
- Note: applying a set of independent actions can be done in any order.
- The result of applying the set $\pi$ in $s$ is defined as:
  \[ \gamma(s, \pi) = (s - \text{effects}^-(\pi)) \cup \text{effects}^+(\pi) \]
  where:
  - $\text{precond}(\pi) = \bigcup_{a \in \pi} \text{precond}(a)$,
  - $\text{effects}^+(\pi) = \bigcup_{a \in \pi} \text{effects}^+(a)$, and
  - $\text{effects}^-(\pi) = \bigcup_{a \in \pi} \text{effects}^-(a)$.
Execution Order of Independent Actions

• Proposition: If a set $\pi$ of independent actions is applicable in state $s$ then, for any permutation $\langle a_1, \ldots, a_k \rangle$ of the elements of $\pi$:
  • the sequence $\langle a_1, \ldots, a_k \rangle$ is applicable to $s$, and
  • the state resulting from the application of $\pi$ to $s$ is the same as from the application of $\langle a_1, \ldots, a_k \rangle$, i.e.:
    $$\gamma(s, \pi) = \gamma(s, \langle a_1, \ldots, a_k \rangle).$$
Layered Plans

- Let $P = (A, s_0, g)$ be a statement of a propositional planning problem and $G = (N, E)$, $N = P_0 \cup A_1 \cup P_1 \cup A_2 \cup P_2 \cup \ldots$, the corresponding planning graph.
- A layered plan over $G$ is a sequence of sets of actions: $\prod = \langle \pi_1, \ldots, \pi_k \rangle$ where:
  * $\pi_i \subseteq A_i \subseteq A$,
  * $\pi_i$ is applicable in state $P_{i-1}$, and
  * the actions in $\pi_i$ are independent.

Layered Plans

- Let $P = (A, s_0, g)$ be a statement of a propositional planning problem and $G = (N, E)$, $N = P_0 \cup A_1 \cup P_1 \cup A_2 \cup P_2 \cup \ldots$, the corresponding planning graph.
- A **layered plan** over $G$ is a sequence of sets of actions: $\prod = \langle \pi_1, \ldots, \pi_k \rangle$ where:
  * $\pi_i \subseteq A_i \subseteq A$,
  * $\pi_i$ is applicable in state $P_{i-1}$, and
  * the actions in $\pi_i$ are independent.
Layered Solution Plan

• A layered plan $\Pi = \langle \pi_1, \ldots, \pi_k \rangle$ is a solution to a to a planning problem $P=(A,s_i,g)$ iff:
  • $\pi_1$ is applicable in $s_i$,
  • for $j \in \{2 \ldots k\}$, $\pi_j$ is applicable in state $\gamma(\ldots \gamma(\gamma(s_i, \pi_1), \pi_2), \ldots, \pi_{j-1})$, and
  • $g \subseteq \gamma(\ldots \gamma(\gamma(s_i, \pi_1), \pi_2), \ldots, \pi_k)$.

• note: independence of actions still not sufficient criterion for solution
Execution Order in Layered Solution Plans

- Proposition: If $\prod = \langle \pi_1, \ldots, \pi_k \rangle$ is a solution to a planning problem $P=(A,s_i,g)$, then:
  - a sequence of actions corresponding to any permutation of the elements of $\pi_1$,
  - followed by a sequence of actions corresponding to any permutation of the elements of $\pi_2$,
  - ...
  - followed by a sequence of actions corresponding to any permutation of the elements of $\pi_k$

is a path from $s_i$ to a goal state.
Problem: Dependent Propositions: Example

- $r_2$ and $ar$:
  - $r_2$: positive effect of Mr12
  - $ar$: positive effect of Lar1
  - but: Mr12 and Lar1 not independent
    - hence: $r_2$ and $ar$ incompatible in $P_1$
- $r_1$ and $r_2$:
  - positive and negative effects of same action: Mr12
    - hence: $r_1$ and $r_2$ incompatible in $P_1$
  - both cases: compatible if they are also
    - two positive effects of one action
    - the positive effects of two independent actions
  - incompatible propositions: cannot be reached through preceding action layer ($A_1$)
No-Operation Actions

- No-Op for proposition $p$:
  - for every action layer and every proposition that may persist
  - name: $A_p$
  - precondition: $p$
  - effect: $p$

- $r_1$ and $r_2$:
  - $r_1$: positive effect of $A_{r1}$
  - $r_2$: positive effect of $M_{r12}$
  - but: $A_{r1}$ and $M_{r12}$ not independent
  - hence: $r_1$ and $r_2$ incompatible in $P_1$

- only one incompatibility test

previous slide: two types of incompatibility (positive effects of dependent actions + positive and negative effects of same action)

- with no-ops: only first type needed (simplification)
Mutex Propositions

• Two propositions $p$ and $q$ in proposition layer $P_j$ are **mutex** (mutually exclusive) if:
  • every action in the preceding action layer $A_j$ that has $p$ as a positive effect (incl. no-op actions) is mutex with every action in $A_j$ that has $q$ as a positive effect, and
  • there is no single action in $A_j$ that has both, $p$ and $q$, as positive effects.

• notation: $\mu P_j = \{ (p,q) \mid p,q \in P_j \text{ are mutex} \}$

• note: mutex relation for propositions is symmetrical (follows from definition)

• proposition layer $P_1$ contains 8 mutex pairs
**Pseudo Code: mutex for Propositions**

- **function mutex(p₁, p₂, μA_j)**
  - **input:** two propositions (from same layer), mutex relation between the actions in the preceding layer

- **for all a₁ ∈ p₁.producers()**
  - **producers:** actions in the preceding layer that have p₁ as a positive effect; should be stored with proposition node

- **for all a₂ ∈ p₂.producers()**
  - **producers:** see above

- **if (a₁, a₂) ∉ μA_j then**
  - test whether the action are in the given set of mutually exclusive actions

- **return false**
  - if not: consistent producers found; propositions are not mutex

- **return true**
  - no consistent producers found; propositions are mutex

- **note:** single action producing both is covered: action cannot be mutex with itself

- **complexity:** let m be number of actions in domain (incl. no-ops); \(O(m^2)\)
Mutex Actions: Example

- $r_1$ and $r_2$ are mutex in $P_1$
- $r_1$ is precondition for Lar1 in $A_2$
- $r_2$ is precondition for Mr21 in $A_2$
- hence: Lar1 and Mr21 are mutex in $A_2$

- dependency between actions in action layer $A_j$ leads to mutex between propositions in $P_j$
- mutex between propositions in $P_j$ leads to mutex between actions in action layer $A_{j+1}$
Mutex Actions

- Two actions \( a_1 \) and \( a_2 \) in action layer \( A_j \) are mutex if:
  - \( a_1 \) and \( a_2 \) are dependent, or
  - a precondition of \( a_1 \) is mutex with a precondition of \( a_2 \).

- notation:
  \[
  \mu A_j = \{ (a_1,a_2) \mid a_1,a_2 \in A_j \text{ are mutex} \}
  \]

Mutex Actions

- Two actions \( a_1 \) and \( a_2 \) in action layer \( A_j \) are **mutex** if:
  - \( a_1 \) and \( a_2 \) are dependent, or
  - dependent actions are necessarily mutex
  - a precondition of \( a_1 \) is mutex with a precondition of \( a_2 \).
  - dependency is domain-specific, i.e. not problem-specific
  - mutex-relation is problem specific
    - pair of actions/propositions may be mutex in one layer
    but not so in another

- notation:
  \[
  \mu A_j = \{ (a_1,a_2) \mid a_1,a_2 \in A_j \text{ are mutex} \}
  \]

- action layer \( A_1 \) contains 2 mutex (dependent) pairs
- action layer \( A_2 \) contains 24 mutex pairs (not all dependent)
- note: mutex relation (for actions and propositions) is symmetrical
  (follows from definition)
Pseudo Code: mutex for Actions

• function mutex(a₁, a₂, μP)
  • μP – mutex relations from the preceding proposition layer
  • if ~independant(a₁, a₂) then
    • return true
  • for all p₁ ∈ precond(a₁)
    • for all p₂ ∈ precond(a₂)
      • if (p₁, p₂) ∈ μP then return true
  • return false

• complexity: let b = max number preconditions/pos. effects/neg effects: \(O(b^2)\)
Decreasing Mutex Relations

• Proposition: If $p, q \in P_{j-1}$ and $(p, q) \notin \mu P_{j-1}$ then $(p, q) \notin \mu P_j$.
  • Proof:
    • if $p, q \in P_{j-1}$ then $Ap, Aq \in A_j$
    • if $(p, q) \notin \mu P_{j-1}$ then $(Ap, Aq) \notin \mu A_j$
    • since $Ap, Aq \in A_j$ and $(Ap, Aq) \notin \mu A_j$, $(p, q) \notin \mu P_j$ must hold

• Proposition: If $a_1, a_2 \in A_{j-1}$ and $(a_1, a_2) \notin \mu A_{j-1}$ then $(a_1, a_2) \notin \mu A_j$.
  • Proof:
    • if $a_1, a_2 \in A_{j-1}$ and $(a_1, a_2) \notin \mu A_{j-1}$ then
      • $a_1$ and $a_2$ are independent and
      • their preconditions in $P_{j-1}$ are not mutex
    • both properties remain true for $P_j$
    • hence: $a_1, a_2 \in A_j$ and $(a_1, a_2) \notin \mu A_j$

• mutex relations are monotonically decreasing (between layers with the same propositions)
Removing Impossible Actions

• actions with mutex preconditions $p$ and $q$ are impossible
  • example: preconditions $r_2$ and $a_r$ of $Ua_r2$ in $A_2$ are mutex

• can be removed from the graph
  • example: remove $Ua_r2$ from $A_2$

• action with mutex preconditions can never be part of any layered plan (will violate applicability condition in definition)

• can be removed from the graph
  • example: remove $Ua_r2$ from $A_2$

• mutex pair of actions must remain in graph because one of the actions may be used in final plan

• note: still consistent with monotonically increasing actions
Reachability in Planning Graphs

• Proposition: Let $P = (A, s_i, g)$ be a propositional planning problem and $G = (N, E)$, $N = P_0 \cup A_1 \cup P_1 \cup A_2 \cup P_2 \cup \ldots$, the corresponding planning graph. If
  • $g$ is reachable from $s_i$
    • then
      • there is a proposition layer $P_g$ such that
        • $g \subseteq P_g$ and
        • $\neg \exists \ g_1, g_2 \in g: (g_1, g_2) \in \mu P_g$.

• still only necessary condition, but relatively efficient to compute
Overview

The Propositional Representation

The Planning-Graph Structure

• just done: defining a new graph that is more efficient to generate and a necessary criterion for solution containment

The Graphplan Algorithm

• now: an algorithm for searching the planning graph for a solution plan
The Graphplan Algorithm: Basic Idea

• expand the planning graph, one action layer and one proposition layer at a time
• from the first graph for which $P_g$ is the last proposition layer such that
  • $g \subseteq P_g$ and
  • $\neg \exists g_1, g_2 \in g: (g_1, g_2) \in \mu P_g$
• search backwards from the last (proposition) layer for a solution

The Graphplan Algorithm: Basic Idea

• expand the planning graph, one action layer and one proposition layer at a time
  • similar to iterative deepening: discover new part of the search space with each iteration

• from the first graph for which $P_g$ is the last proposition layer such that
  • $g \subseteq P_g$ and
  • $\neg \exists g_1, g_2 \in g: (g_1, g_2) \in \mu P_g$
  • no need to search for solutions in graph with fewer layers; see last proposition

• search backwards from the last (proposition) layer for a solution

• two major steps:
  • expansion of planning graph to next proposition layer
  • searching a given planning graph for a solution
Planning Graph Data Structure

- k-th planning graph \( G_k \):
  - nodes \( N \):
    - array of proposition layers \( P_0 \ldots P_k \)
      - proposition layer \( j \): set of proposition symbols
    - array of action layers \( A_1 \ldots A_k \)
      - proposition layer \( j \): set of action symbols
  - edges \( E \):
    - precondition links: \( \text{pre}_j \subseteq P_{j-1} \times A_j, j \in \{1 \ldots k\} \)
    - positive effect links: \( e^+_j \subseteq A_j \times P_j, j \in \{1 \ldots k\} \)
    - negative effect links: \( e^-_j \subseteq A_j \times P_j, j \in \{1 \ldots k\} \)
    - proposition mutex links: \( \mu_{Aj} \subseteq A_j \times A_j, j \in \{1 \ldots k\} \)
    - action mutex links: \( \mu_{Pj} \subseteq P_j \times P_j, j \in \{1 \ldots k\} \)

- note: instance of this data structure does not depend on problem
- initial planning graph: \( P_0 = s_i \), rest is empty sets
Pseudo Code: expand

function expand(G_k-1)
    \[ A_k \leftarrow \{ a \in A \mid \text{precond}(a) \subseteq \mathcal{P}_{k-1} \text{ and } \{ (p_1, p_2) \mid p_1, p_2 \in \text{precond}(a) \} \cap \mu_{\mathcal{P}_{k-1}} = \emptyset \} \]
    \[ \mu A_k \leftarrow \{ (a_1, a_2) \mid a_1, a_2 \in A_k, a_1 \neq a_2, \text{ and } \text{mutex}(a_1, a_2, \mu_{\mathcal{P}_{k-1}}) \} \]
    \[ \mu P_k \leftarrow \{ p \mid \exists a \in A_k : p \in \text{effects}^+(a) \} \]
    \[ \mu_{\mathcal{P}_k} \leftarrow \{ (p_1, p_2) \mid p_1, p_2 \in \mathcal{P}_k, p_1 \neq p_2 \text{ and } \text{mutex}(p_1, p_2, \mu A_k) \} \]
    for all \( a \in A_k \)
    \[ \text{pre}_k \leftarrow \text{pre}_k \cup \{ (p \mid p \in \mathcal{P}_{k-1} \text{ and } p \in \text{precond}(a)) \} \times a \]
    \[ e^+_k \leftarrow e^+_k \cup \{ (a \times \{ p \mid p \in \mathcal{P}_k \text{ and } p \in \text{effects}^+(a)) \} \}
    \[ e^-_k \leftarrow e^-_k \cup \{ (a \times \{ p \mid p \in \mathcal{P}_k \text{ and } p \in \text{effects}^+(a)) \} \}

actions with satisfied, non-mutex preconditions (incl. no-ops)

•union of all positive effects

•for all \( a \in A_k \)

••union of all positive effects

••for all \( a \in A_k \)

The Graphplan Planner 59
Planning Graph Complexity

• Proposition: The size of a planning graph up to level $k$ and the time required to expand it to that level are polynomial in the size of the planning problem.

• Proof:
  • problem size: $n$ propositions and $m$ actions
  • $|P_j| \leq n$ and $|A_j| \leq n + m$ (incl. no-op actions)
  • algorithms for generating each layer and all link types are polynomial in size of layer
Fixed-Point Levels

- A fixed-point level in a planning graph \( G \) is a level \( \kappa \) such that for all \( i, i > \kappa \), level \( i \) of \( G \) is identical to level \( \kappa \), i.e. \( P_i = P_\kappa \), \( \mu P_i = \mu P_\kappa \), \( A_i = A_\kappa \), and \( \mu A_i = \mu A_\kappa \).

- **Proposition**: Every planning graph \( G \) has a fixed-point level \( \kappa \), which is the smallest \( k \) such that \( |P_k| = |P_{k+1}| \) and \( |\mu P_k| = |\mu P_{k+1}| \).
  - **Proof**:  
    - \( P_i \) grows monotonically and \( \mu P_i \) shrinks monotonically  
    - \( A_i \) and \( P_i \) only depend on \( P_{i-1} \) and \( \mu P_{i-1} \)  

Fixed-Point Levels

- A **fixed-point level** in a planning graph \( G \) is a level \( \kappa \) such that for all \( i, i > \kappa \), level \( i \) of \( G \) is identical to level \( \kappa \), i.e. \( P_i = P_\kappa \), \( \mu P_i = \mu P_\kappa \), \( A_i = A_\kappa \), and \( \mu A_i = \mu A_\kappa \).

- **Proposition**: Every planning graph \( G \) has a fixed-point level \( \kappa \), which is the smallest \( k \) such that \( |P_k| = |P_{k+1}| \) and \( |\mu P_k| = |\mu P_{k+1}| \).
  - **Proof**:  
    - \( P_i \) grows monotonically and \( \mu P_i \) shrinks monotonically  
    - \( A_i \) and \( P_i \) only depend on \( P_{i-1} \) and \( \mu P_{i-1} \)  

- **time complexity**: \( O(n+m) \) from fixed point level; only copying required
Searching the Planning Graph

- **general idea:**
  - search backwards from the last proposition layer $P_k$ in the current graph
  - let $g$ be the set of goal propositions that need to be achieved at a given proposition layer $P_j$ (initially the last layer)
  - find a set of actions $\pi_j \subseteq A_j$ such that these actions are not mutex and together achieve $g$
  - take the union of the preconditions of $\pi_j$ as the new goal set to be achieved in proposition layer $P_{j-1}$
Planning Graph Search Example

• initial goal: $a_2$ and $b_1$
• only one incoming positive effect link per goal (but no-ops not shown)
• achievable with Uar2 and Ubq1 (which are not mutex; mutex relations not shown)
• precondition links indicate sub-goal at next layer
• new sub-goal at $P_2$: $r_2$, $q_1$, $ar$, $bq$
• only one incoming positive effect link per goal condition (but no-ops not shown)
  • achieve $ar$ and $bq$ with no-ops
  • achieve $r_2$ with Mr12 and $q_1$ with Mq21
• precondition links (for Mr12 and Mq21) indicate some sub-goal at next layer
• complete sub-goal (incl. preconditions of no-ops) at $P_1$: $r_1$, $q_2$, $ar$, $bq$
• only one incoming positive effect link per goal condition (but no-ops not shown)
  • achieve $r_1$ and $q_2$ with no-ops
  • achieve $ar$ with Lar1 and $bq$ with Lbq2
• precondition links (for Lar1 and Lbq2) indicate some sub-goal at next layer
• complete sub-goal (incl. preconditions of no-ops) at $P_0$: complete initial state
Planning Graph as AND/OR-Graph

• OR-nodes:
  • nodes in proposition layers
  • links to actions that support the propositions

• AND-nodes:
  • nodes in action layers
  • k-connectors all preconditions of the action

• search:
  • AO* not best algorithm because it does not exploit layered structure
Repeated Sub-Goals

• ultimate goal leads to possible sub-goals at $P_j$
• possible sub-goals at $P_j$ lead to possible sub-goals at $P_i$
  • search to initial proposition layer to see whether sub-goals can be achieved
  • suppose: sub-goals at $P_i$ cannot be achieved
• backtrack to later layer, say $P_j$
• possible sub-goals at $P_j$ may lead to same possible sub-goals at $P_i$, but in a different way
  • no need to repeat search: same sub-goals at same layer still cannot be achieved
  • generalization: same some sub-goals at same or earlier layer still cannot be achieved
  • otherwise no-op would achieve sub-goal at later layer
The *nogood* Table

- *nogood table* (denoted $\nabla$) for planning graph up to layer $k$:
  - array of $k$ sets of sets of goal propositions
    - inner set: one combination of propositions that cannot be achieved
    - outer set: all combinations that cannot be achieved (at that layer)
  - before searching for set $g$ in $P_j$:
    - check whether $g \in \nabla(j)$
  - when search for set $g$ in $P_j$ has failed:
    - add $g$ to $\nabla(j)$

The *nogood* Table

- *nogood table* (denoted $\nabla$) for planning graph up to layer $k$:
  - array of $k$ sets of sets of goal propositions
    - inner set: one combination of propositions that cannot be achieved
    - outer set: all combinations that cannot be achieved (at that layer)
  - mutex only gives pairs of propositions that cannot be achieved together, *nogood* table gives impossible tuples
  - before searching for set $g$ in $P_j$:
    - check whether $g \in \nabla(j)$
    - actually: in $j$ or later layer
  - when search for set $g$ in $P_j$ has failed:
    - add $g$ to $\nabla(j)$
    - or move?
Pseudo Code: extract

function extract(G,g,i)
    · inputs: planning graph G, set of propositions (sub-goals) g, and layer at which sub-goals need to be achieved i
    · output: a layered plan $\langle \pi_1, \ldots, \pi_i \rangle$ that achieves g at i in G or failure if there is no such plan
    · if $i=0$ then return $\langle \rangle$
    · trivial success with empty plan
    · if $g \in \nabla(i)$ then return failure
    · sub-goals have resulted in failure before
    · $\pi_i \leftarrow \text{gpSearch}(G,g,\{\},i)$
    · perform the search
    · if $\pi_i \neq \text{failure}$ then return $\pi_i$
    · the search was successful
    · $\nabla(i) \leftarrow \nabla(i) + g$
    · unsuccessful search: remember unachievable sub-goals
    · return failure
Pseudo Code: gpSearch

function gpSearch(G, g, π, i)
    if g = {} then
        [] ← extract(\(G, \bigcup_{a \in \pi} \text{precond}(a), i-1\))
        if [] = failure then return failure
        return \(\prod \cdot \langle \pi \rangle\)
    end
    \(p \leftarrow g.\text{selectOne()}\)
    resolvers ← \(\{ a \in A_i \mid p \in \text{effects}^+(a) \text{ and } \lnot \exists a' \in \pi: (a,a') \in \mu A_i \}\)\)
    if resolvers = {} then return failure
    \(a \leftarrow \text{resolvers.choseOne()}\)
    return gpSearch(G, g - \(\text{effects}^+(a)\), π + \(a\), i)

Pseudo Code: gpSearch

- function gpSearch(G, g, π, i)
  - inputs: planning graph G, remaining sub-goals g, and set of actions already committed to π, both at level i
  - outputs: layered plan
  - if g = {} then
    - all actions chosen
    \(\prod \leftarrow \text{extract}(G, \bigcup_{a \in \pi} \text{precond}(a), i-1)\)
    - if \(\prod\) = failure then return failure
    - return \(\prod \cdot \langle \pi \rangle\)
  - \(p \leftarrow g.\text{selectOne()}\)
    - no need to backtrack here; order only important for efficiency
  - resolvers ← \(\{ a \in A_i \mid p \in \text{effects}^+(a) \text{ and } \lnot \exists a' \in \pi: (a,a') \in \mu A_i \}\)\)
    - if resolvers = {} then return failure
    - \(a \leftarrow \text{resolvers.choseOne()}\)
      - non-deterministic choice point; backtrack to here
  - return gpSearch(G, g - \(\text{effects}^+(a)\), π + \(a\), i)
Pseudo Code: graphplan

• function graphplan(A,s,g)  
  • given planning problem, return layered solution plan

• i ← 0; \( \nabla \leftarrow \{\} \); \( P_0 \leftarrow s_i \); \( G \leftarrow (P_0,\{\}) \)

• while \((g \notin P_i \) or \( g^2 \cap \mu P_i \neq \{\})\) and \( \neg \text{fixedPoint}(G)\) do

  • \( i \leftarrow i+1; \text{expand}(G) \)

  • if \( g \notin P_i \) or \( g^2 \cap \mu P_i \neq \{\} \) then return failure

  • \( \eta \leftarrow \text{fixedPoint}(G) \) ? \( |\nabla(\kappa)| : 0 \)
  
  • \( \eta \leftarrow |\nabla(\kappa)| \)

• \( \prod \leftarrow \text{extract}(G,g,i) \)

• while \( \prod = \text{failure} \) do

  • \( i \leftarrow i+1; \text{expand}(G) \)

  • \( \prod \leftarrow \text{extract}(G,g,i) \)

• if \( \prod = \text{failure} \) and \( \text{fixedPoint}(G) \) then

  • if \( \eta = |\nabla(\kappa)| \) then return failure

  • \( \eta \leftarrow |\nabla(\kappa)| \)

• return \( \prod \)
Graphplan Properties

- **Proposition**: The Graphplan algorithm is sound, complete, and always terminates.
  - It returns failure iff the given planning problem has no solution;
  - otherwise, it returns a layered plan $\prod$ that is a solution to the given planning problem.

- Graphplan is orders of magnitude faster than previous techniques!

  *caveat: restriction to propositional STRIPS*
Overview

- The Propositional Representation
- The Planning-Graph Structure
- The Graphplan Algorithm
- Planning-Graph Heuristics

Overview

The Propositional Representation

• The Planning-Graph Structure
• The Graphplan Algorithm
Forward State-Space Search

- idea: apply standard search algorithms (breadth-first, depth-first, A*, etc.) to planning problem:
  - search space is subset of state space
  - nodes correspond to world states
  - arcs correspond to state transitions
  - path in the search space corresponds to plan
DWR Example State

goal: (and
    (in ca p2) (in cb q2) (in cc p2) (in cd q2) (in ce q2) (in cf q2))
Heuristics

- estimate distance to nearest goal state
  - number of unachieved goals (not admissible)
  - number of unachieved goals / max. number of positive effects per operator (admissible)

- example state (prev. slide):
  - actual goal distance: 35 actions
  - $h(s) = 6$
  - $h(s) = 6 / 4$
Finding Better Heuristics

- solve "relaxed" problem and use solution as heuristic
- planning heuristic:
  - planning problem: $P = (O, s, g)$
  - for $p \in g$: $\text{min-layer}(p) = \text{index of first proposition layer in planning graph that contains } p$
  - admissible heuristic: $\max(p \in g): \text{min-layer}(p)$
  - not admissible: $\sum(p \in g): \text{min-layer}(p)$
- no need to compute mutex relations
- no need to re-compute planning graph for ground backward search
The FF Planner (Basics)

- heuristic
  - based on planning graph without negative effects
  - backward search possible in polynomial time
- search strategy
  - enforced hill-climbing: commit to first state with better f-value
Overview

The Propositional Representation

*The Planning-Graph Structure*

*The Graphplan Algorithm*