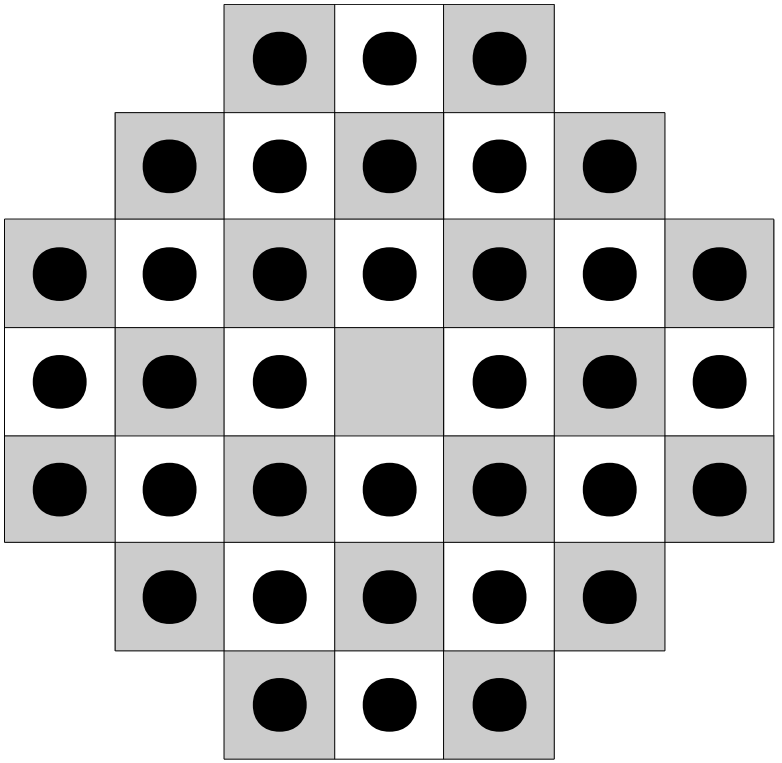
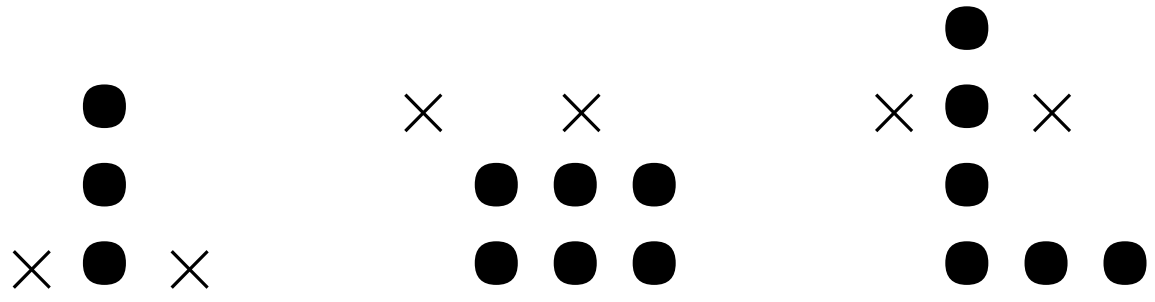
Planning with Heuristics

A More Difficult Planning Problem



Search Space for Peg Solitaire too large for blind search
(Solution depth: 35; branching factor: 4 initially, quickly increasing)

Domain-Dependent Heuristics: A Planning Strategy for Peg Solitaire



3-pattern 6-pattern L-pattern

Plan Skeletons

- A (nondeterministic) GOLOG program can be used to describe a plan skeleton.
- If executed offline, the program defines a heuristics which spans a (reduced) search space.
- Two notions of completeness:
  - strong completeness: all plans are in the skeleton
  - weak completeness: skeleton contains a plan if problem is solvable

A GOLOG Program for Peg Solitaire

proc *SolvePattern*(*p*)
  while ¬*Solved*(*p*) **do**
    $\pi x.\pi y.\pi z.\ (y \in p?;\ Jump(x, y, z))$
  endWhile
endProc;

while ¬*Goal* **do**
  $\pi p.\ (IsPattern(p)?;\ SolvePattern(p))$
endWhile

Jump(*x*, *y*, *z*) :<=> move with peg in *x* over *y* into cell *z*.

Remark: This heuristics is complete but not strongly complete.

A Generic Plan Skeleton

while ¬*Goal* **do**

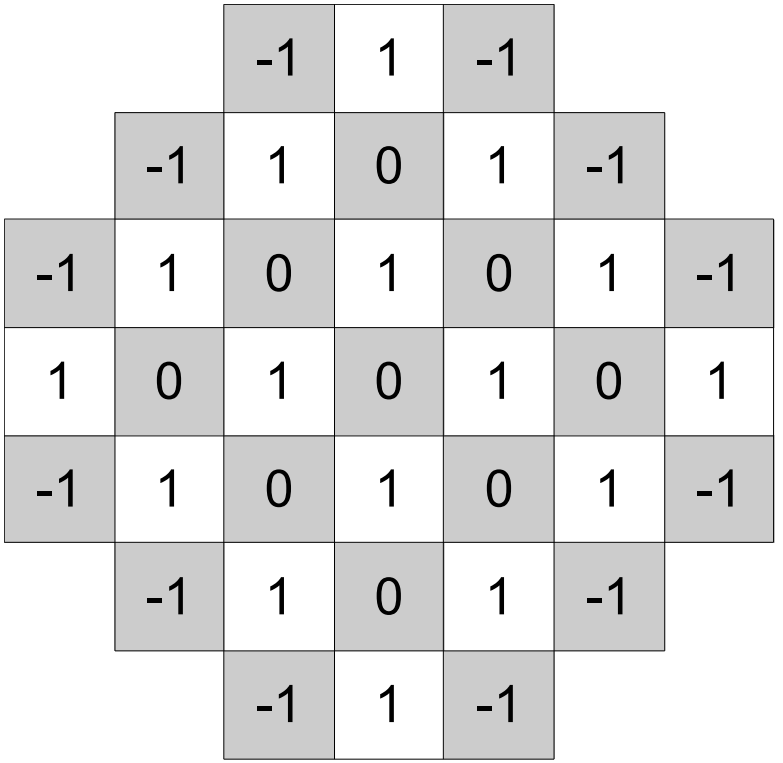
$\pi a.\ a$; *Allowed*?

endWhile

This program describes search with local constraints for action selection.

A branch in the search tree is cut off by the domain-dependent condition *Allowed*.

Example: Resource Count in Peg Solitaire



$Holds(Allowed, s) \stackrel{def}{=} ResourceCount(s) \geq 0$

ResourceCount(*s*) : <=> weighted sum of all occupied cells in *s*

Remark: This heuristics is strongly complete.

Pruning Techniques

General Pruning Techniques

Linear Temporal Logic: Classical logic extended by modalities

symbol	meaning
$\bigcirc\phi$	ϕ holds next
$\Box\phi$	ϕ always holds
$\Diamond\phi$	ϕ eventually holds
$\phi\ \mathcal{U}\ \psi$	ϕ holds until ψ

Example: $\Box[In(x, y) \supset In(x, y)\ \mathcal{U}\ At(y, GoalLocation(x))]$

Semantics

Let S_0, S_1, S_2, ... be an infinite sequence of situations

A formula ϕ is true in S_i, written $S_i \models \phi$, if

- *Holds*(ϕ, S_i), where ϕ atomic;
- $S_{i+1} \models \psi$, where ϕ is $\bigcirc\psi$;
- $S_j \models \psi$ for all $j \geq i$, where ϕ is $\Box\psi$;
- $S_j \models \psi$ for some $j \geq i$, where ϕ is $\Diamond\psi$;
- $S_j \models \psi_2$ for some $j \geq i$ such that $S_k \models \psi_1$ for all $i \leq k < j$, where ϕ is $\psi_1 \mathcal{U} \psi_2$;
- $S_j \not\models \psi$, where ϕ is $\neg\psi$; likewise for $\wedge$, $\vee$, $\supset$, $\equiv$.

Example

Consider the interpretation

S_0	S_1	S_2	S_3	S_4	
A	$\neg A$	$\neg A$	$\neg A$	$\neg A$	...
B	B	$\neg B$	$\neg B$	$\neg B$	

Then

$$S_0 \models \Box(A \supset B)$$

$$S_0 \models \Box\, A \supset B$$

$$S_0 \models \bigcirc\Box\, \neg A$$

$$S_0 \models \Diamond(\neg A \wedge \neg B)$$

$$S_0 \models \Box(B\, \mathcal{U}\, \neg A)$$

A Logistics Domain

Symbol	Type
Load	OBJECT × VEHICLE ↦ ACTION
Unload	OBJECT × VEHICLE ↦ ACTION
Move	VEHICLE × LOCATION ↦ ACTION
Fly	VEHICLE × LOCATION ↦ ACTION

Symbol	Type
At	OBJECT × LOCATION ↦ FLUENT
In	OBJECT × VEHICLE ↦ FLUENT
Location	VEHICLE × LOCATION ↦ FLUENT

- 10 objects, 2 locations and 1 truck each in 4 cities, 3 planes

  → total state space $6 \cdot 10^{14}$

Control Rules for Logistics Domain

1. Move a vehicle only if there are no objects at its present location which need to be loaded into or unloaded from the vehicle;

2. Move a vehicle only to locations where it needs to pick up or to drop goods;

3. Load objects only into the kind of vehicle (truck or plane) with which they need to be transported;

4. Unload objects only where they need to be unloaded.

With these simple control rules, problems with state space 10^{60} and beyond can be solved.

Auxiliary Predicates

MustBeMovedBy(*o*, *v*) :<=> the goal requires to move object *o* by the type of vehicle (truck or plane) to which *v* belongs

MustBeUnloadedAt(*o*, *l*) :<=> the goal requires to unload object *o* at loaction *l* (possibly an airport location)

The Control Rules in Linear Temporal Logic

1. $\Box[Location(v, l) \wedge (\exists o)\,(At(o, l) \wedge MustBeMovedBy(o, v)) \supset \bigcirc Location(v, l)]$

   $\Box[Location(v, l) \wedge (\exists o)\,(In(o, v) \wedge MustBeUnloadedAt(o, l)) \supset \bigcirc Location(v, l)]$

2. $\Box[Location(v, l) \wedge \bigcirc Location(v, l') \wedge l \neq l' \supset$

   $(\exists o)\,(At(o, l') \wedge MustBeMovedBy(o, v) \vee In(o, v) \wedge MustBeUnloadedAt(o, l'))]$

3. $\neg MustBeMovedBy(o, v) \supset \Box\neg In(o, v)$

4. $\Box[In(o, v) \wedge Location(v, l) \wedge \neg MustBeUnloadedAt(o, l) \supset \bigcirc In(o, v)]$

Progression (1)

- Evaluating a temporal logic formula against an action sequence requires to consider the entire history, which can be costly.
- Control formulas can be progressed successively through situations, and then be locally evaluated.
- For example, $\Box\phi$ is valid now if ϕ holds and $\Box\phi$ holds in the next situation.

Progression (2)

Let S be the current situation, then *Progress*(ϕ) is

- *True* if *Holds*(ϕ, S) and ϕ atomic
- *False* if $\neg$*Holds*(ϕ, S) and ϕ atomic
- *Progress* (ψ_1) $\wedge$ *Progress* (ψ_2) if ϕ is $\psi_1 \wedge \psi_2$; likewise for $\vee$, $\neg$, $\supset$, $\equiv$
- $\bigwedge_c$ *Progress*($\psi\{x/c\}$) if ϕ is $(\forall x)\psi$
- $\bigvee_c$ *Progress*($\psi\{x/c\}$) if ϕ is $(\exists x)\psi$

Note: This restricts progression to domains with finitely many ground terms.

Progression (3)

- ψ, if ϕ is $\bigcirc\psi$
- *Progress* $(\psi) \wedge \Box\psi$, if ϕ is $\Box\psi$
- *Progress* $(\psi) \vee \Diamond\psi$, if ϕ is $\Diamond\psi$
- *Progress* $(\psi_2) \vee$ (*Progress* $(\psi_1) \wedge \psi_1 \mathcal{U} \psi_2$), if ϕ is $\psi_1 \mathcal{U} \psi_2$

Preferences

Planning with Preferences

- In the classical definition of planning, all plans are equal
- Additional preferences allow to distinguish “good” from “bad” plans
- A preference can be given quantitatively (by an objective function) or qualitatively by preference formulas

Basic Desire Formulas

Definition 4.3.1

A basic desire can be

- a fluent
- the atomic `final`(*f*), where *f* is a fluent
- the atomic `occurs`(*a*), where *a* is an action
- any combination of the above using the logical connectives and the modalities of linear temporal logic

Atomic Preference Formulas

Definition 4.3.2

An atomic preference formula is of the form

$$\varphi_0 \prec \varphi_1 \prec \ldots \prec \varphi_n$$

where $n \geq 0$ and the φ_i's are basic desire formulas.

Note: If $n = 0$, then a preference formula is simply a basic desire formula.

Preference Formulas

Definition 4.3.3

A preference formula can be

- an atomic preference formula
- a conditional $\varphi \Rightarrow \Phi$
- a general conjunction Φ_1 & ... & Φ_n
- a general disjunction Φ_1 | ... | Φ_n

φ basic desire; Φ, Φ_1, ... , Φ_n preference formulas; $n \geq 2$

Evaluation of Preferences

Each plan determines a sequence S_0, S_1, S_2, ..., S_n of situations.

Consider the infinite sequence S_0, S_1, S_2, ..., S_n, S_n, S_n, ...

Basic desires are evaluated like temporal logic formulas (cf Slide 10), augmented by the definition

- $S_i \models$ `final`(f) if *Holds*(f, S_n)
- $S_i \models$ `occurs`(a) if S_{i+1} = *Do*(a, S_i)

Weight of Basic Desire

Consider S_0, S_1, S_2, ..., S_n, S_n, S_n, ...

Let φ be a basic desire, then

$$\omega(\varphi) = \begin{cases} 0 & \textit{if } S_0 \models \varphi \\ 1 & \textit{otherwise} \end{cases}$$

Weight of Atomic Preference

$$\omega(\varphi_0 \prec \ldots \prec \varphi_n) = \begin{cases} min\{i : \omega(\varphi_i) = 0\} & \textit{if it exists} \\ n+1 & \textit{otherwise} \end{cases}$$

Weight of Preferences

$$\omega(\varphi \Rightarrow \Phi) = \begin{cases} \omega(A) & \text{if } \omega(\varphi)=0 \\ 0 & \text{otherwise} \end{cases}$$

$$\omega(\Phi_1 \& \ldots \& \Phi_n) = \sum_i \omega(\Phi_1)$$

$$\omega(\Phi_1 | \ldots | \Phi_n) = \min_i \omega(\Phi_1)$$

Preferred Plans

- Ideal plans, which satisfy all preferences, have weight 0.
- Plan p_1 is preferred over plan p_2 if the sum of the weights of all preference formulas wrt. p_1 is lower than that wrt. p_2.

Progression of Preference Formulas (1)

- To avoid evaluation of preferences for each plan separately, preference formulas can be progressed.

- Progression of basic desires as temporal logic formulas (cf Slides 24 - 25) augmented by the definition

  - *Progress*(`final`(*f*)) is `final`(*f*); and

  - *Progress*(`occurs`(*a*)) is `next`(*a*), with the auxiliary definition

    $S_i \models$ `next`(*a*) if S_i = *Do*(*a*, *s*) for some *s*

    Progress(`next`(*a*)) is *True* if *S* = *Do*(*a*, *s*) for some *s*, else *False*

Progression of Preference Formulas (2)

- $Progress(\varphi_0 \prec \varphi_1 \prec ... \prec \varphi_n) = Progress(\varphi_0) \prec Progress(\varphi_1) \prec ... \prec Progress(\varphi_n)$
- $Progress(\varphi \Rightarrow \Phi) = Progress(\varphi) \Rightarrow Progress(\Phi)$
- $Progress(\Phi_1 \& ... \& \Phi_n) = Progress(\Phi_1) \& ... \& Progress(\Phi_n)$
- $Progress(\Phi_1 | ... | \Phi_n) = Progress(\Phi_1) | ... | Progress(\Phi_n)$

Correctness of Progression

Consider a sequence S_0, S_1, S_2, ..., S_n, S_n, S_n, ...

A formula Φ has the same weight wrt. S_0, S_1, ... as $Progress^n(\Phi)$ has wrt. S_n, S_n, ...