Principles of AI Planning

5. State-space search: progression and regression

Malte Helmert
Albert-Ludwigs-Universität Freiburg

October 31st, 2008

State-space search

- state-space search: one of the big success stories of AI
- many planning algorithms based on state-space search
  (we’ll see some other algorithms later, though)
- will be the focus of this and the following topics
- we assume prior knowledge of basic search algorithms
  - uninformed vs. informed
  - systematic vs. local
- background on search: Russell & Norvig, Artificial Intelligence – A Modern Approach, chapters 3 and 4

Satisficing or optimal planning?

Must carefully distinguish two different problems:
- satisficing planning: any solution is OK
  (although shorter solutions typically preferred)
- optimal planning: plans must have shortest possible length

Both are often solved by search, but:
- details are very different
- almost no overlap between good techniques for satisficing planning and good techniques for optimal planning
- many problems that are trivial for satisficing planners are impossibly hard for optimal planners
Planning by state-space search

How to apply search to planning? ⇾ many choices to make!

Choice 1: Search direction
- progression: forward from initial state to goal
- regression: backward from goal states to initial state
- bidirectional search

Choice 2: Search space representation
- search nodes are associated with states
- search nodes are associated with sets of states

Choice 3: Search algorithm
- uninformed search:
  depth-first, breadth-first, iterative depth-first, ...
- heuristic search (systematic):
  greedy best-first, A*, Weighted A*, IDA*, ...
- heuristic search (local):
  hill-climbing, simulated annealing, beam search, ...

Choice 4: Search control
- heuristics for informed search algorithms
- pruning techniques: invariants, symmetry elimination, helpful actions pruning, ...
Search-based satisficing planners

FF (Hoffmann & Nebel, 2001)

- search direction: forward search
- search space representation: single states
- search algorithm: enforced hill-climbing (informed local)
- heuristic: FF heuristic (inadmissible)
- pruning technique: helpful actions (incomplete)

⇝ one of the best satisficing planners

Search-based optimal planners

Fast Downward + $h_{HHH}$ (Helmert, Haslum & Hoffmann, 2007)

- search direction: forward search
- search space representation: single states
- search algorithm: $A^*$ (informed systematic)
- heuristic: merge-and-shrink abstractions (admissible)
- pruning technique: none

⇝ one of the best optimal planners

Our plan for the next lectures

Choices to make:
1. search direction: progression/regression/both  
   ⇝ this chapter
2. search space representation: states/sets of states  
   ⇝ this chapter
3. search algorithm: uninformed/heuristic; systematic/local  
   ⇝ next chapter
4. search control: heuristics, pruning techniques  
   ⇝ following chapters

Planning by forward search: progression

Progression: Computing the successor state $app_o(s)$ of a state $s$ with respect to an operator $o$.

Progression planners find solutions by forward search:
- start from initial state
- iteratively pick a previously generated state and progress it through an operator, generating a new state
- solution found when a goal state generated

pro: very easy and efficient to implement
Search space representation in progression planners

Two alternative search spaces for progression planners:
1. search nodes correspond to states
   ▶ when the same state is generated along different paths, it is not considered again (duplicate detection)
   ▶ pro: fast
   ▶ con: memory intensive (must maintain closed list)
2. search nodes correspond to operator sequences
   ▶ different operator sequences may lead to identical states (transpositions)
   ▶ pro: can be very memory-efficient
   ▶ con: much wasted work (often exponentially slower)

⇝ first alternative usually preferable
Progression planning example (depth-first search)

I

G

Progression planning example (depth-first search)

I

G

Progression planning example (depth-first search)

I

G

Progression planning example (depth-first search)

I

G

Progression planning example (depth-first search)

Going through a transition graph in forward and backward directions is not symmetric:
- forward search starts from a single initial state;
- backward search starts from a set of goal states
- when applying an operator \( o \) in a state \( s \) in forward direction, there is a unique successor state \( s' \);
- if we applied operator \( o \) to end up in state \( s' \),
  there can be several possible predecessor states \( s \)

\( \Rightarrow \) most natural representation for backward search in planning associates sets of states with search nodes
Planning by backward search: regression

Regression: Computing the possible predecessor states $\text{regr}_o(S)$ of a set of states $S$ with respect to the last operator $o$ that was applied.

Regression planners find solutions by backward search:

- start from set of goal states
- iteratively pick a previously generated state set and regress it through an operator, generating a new state set
- solution found when a generated state set includes the initial state

Pro: can handle many states simultaneously
Con: basic operations complicated and expensive

Search space representation in regression planners

identify state sets with logical formulae:

- search nodes correspond to state sets
- each state set is represented by a logical formula: $\phi$ represents $\{s \in S \mid s \models \phi\}$
- many basic search operations like detecting duplicates are NP-hard or coNP-hard

Regression planning example (depth-first search)
Regression for STRIPS planning tasks

Definition (STRIPS planning task)
A planning task is a STRIPS planning task if all operators are STRIPS operators and the goal is a conjunction of literals.

Regression for STRIPS planning tasks is very simple:
- Goals are conjunctions of literals $l_1 \land \cdots \land l_n$.
- **First step**: Choose an operator that makes some of $l_1, \ldots, l_n$ true and makes none of them false.
- **Second step**: Remove goal literals achieved by the operator and add its preconditions.
- Outcome of regression is again conjunction of literals.
Regression

**STRIPS regression**

**Definition**

Let $\phi = \phi_1 \land \cdots \land \phi_k$, $\gamma = \gamma_1 \land \cdots \land \gamma_n$ and $\eta = \eta_1 \land \cdots \land \eta_m$ be non-contradictory conjunctions of literals.

The **STRIPS regression** of $\phi$ with respect to $o = \langle \gamma, \eta \rangle$ is

$$sregr_o(\phi) := \bigwedge((\{\phi_1, \ldots, \phi_k\} \setminus \{\eta_1, \ldots, \eta_m\}) \cup \{\gamma_1, \ldots, \gamma_n\})$$

provided that this conjunction is non-contradictory and that $\neg \phi_i \neq \eta_j$ for all $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, m\}$.

(Otherwise, $sregr_o(\phi)$ is undefined.)

(A conjunction of literals is contradictory iff it contains two complementary literals.)

---

**Regression for general planning tasks**

- With disjunctions and conditional effects, things become more tricky. How to regress $A \lor (B \land C)$ with respect to $\langle Q, D \triangleright B \rangle$?
- The story about goals and subgoals and fulfilling subgoals, as in the STRIPS case, is no longer useful.
- We present a general method for doing regression for any formula and any operator.
- Now we extensively use the idea of representing sets of states as formulae.

---

**Effect preconditions**

**Definition (effect precondition)**

The **effect precondition** $EPC_l(e)$ for literal $l$ and effect $e$ is defined as follows:

$$EPC_l(l) = \top$$

$$EPC_l(l') = \bot \text{ if } l \neq l' \text{ (for literals $l'$)}$$

$$EPC_l(e_1 \land \cdots \land e_n) = EPC_l(e_1) \lor \cdots \lor EPC_l(e_n)$$

$$EPC_l(c \triangleright e) = EPC_l(e) \land c$$

**Intuition:** $EPC_l(e)$ describes the situations in which effect $e$ causes literal $l$ to become true.
**Effect precondition examples**

**Example**

\[
\text{EPC}_a(b \land c) = \bot \lor \bot \equiv \bot \\
\text{EPC}_a(a \land (b \triangleright a)) = \top \lor (\top \land b) \equiv \top \\
\text{EPC}_a((c \triangleright a) \land (b \triangleright a)) = (\top \land c) \lor (\top \land b) \equiv c \lor b
\]

**Effect preconditions: connection to change sets**

**Lemma (A)**

Let \( s \) be a state, \( l \) a literal and \( e \) an effect. Then \( l \in [e]_s \) if and only if \( s \models \text{EPC}_l(e) \).

**Proof.**

Induction on the structure of the effect \( e \).

**Base case 1,** \( e = l \): \( l \in [l]_s = \{l\} \) by definition, and \( s \models \text{EPC}_l(l) = \top \) by definition. Both sides of the equivalence are true.

**Base case 2,** \( e = l' \) for some literal \( l' \neq l \): \( l' \not\in [l']_s = \{l'\} \) by definition, and \( s \not\models \text{EPC}_l(l') = \bot \) by definition. Both sides are false.

**Proof (ctd.)**

**Inductive case 1,** \( e = e_1 \land \cdots \land e_n \):

\[
\text{\( l \in [e]_s \) iff \( l \in [e_1]_s \cup \cdots \cup [e_n]_s \) (Def \([e_1]_s \land \cdots \land [e_n]_s\))}
\]

if \( l \in [e'_j]_s \) for some \( e'_j \in \{e_1, \ldots, e_n\} \)

if \( s \models \text{EPC}_{e'_j}(e'_j) \) for some \( e'_j \in \{e_1, \ldots, e_n\} \) (IH)

if \( s \models \text{EPC}_{e_1}(e_1) \lor \cdots \lor \text{EPC}_{e_n}(e_n) \)

if \( s \models \text{EPC}_{e_1 \land \cdots \land e_n} \).

(Def EPC)

**Inductive case 2,** \( e = c \triangleright e' \):

\[
\text{\( l \in [c \triangleright e']_s \) iff \( l \in [e']_s \) and \( s \models c \) (Def \([c \triangleright e']_s\))}
\]

if \( s \models \text{EPC}_{e'_j}(e'_j) \) and \( s \models c \) (IH)

if \( s \models \text{EPC}_{e'_j}(e'_j) \land c \)

if \( s \models \text{EPC}_{c \triangleright e'} \).

(Def EPC)

**Remark**

Notice that in terms of \( \text{EPC}_a(e) \), any operator \( \langle c, e \rangle \) can be expressed in normal form as

\[
\left\langle c, \bigwedge_{a \in A} ((\text{EPC}_a(e) \triangleright a) \land (\text{EPC}_{\neg a}(e) \triangleright \neg a)) \right\rangle.
\]
Regressing state variables

The formula $EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))$ expresses the value of state variable $a \in A$ after applying $o$ in terms of values of state variables before applying $o$.

Either:
- $a$ became true, or
- $a$ was true before and it did not become false.

Regressing state variables: examples

Example
Let $e = (b \triangleright a) \land (c \triangleright \neg a) \land b \land \neg d$.

<table>
<thead>
<tr>
<th>variable</th>
<th>$EPC_{\neg a}(e) \lor (\cdots \land \neg EPC_{\neg a}(e))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>$b \lor (a \land \neg c)$</td>
</tr>
<tr>
<td>$b$</td>
<td>$\top \lor (b \land \bot) \equiv \top$</td>
</tr>
<tr>
<td>$c$</td>
<td>$\bot \lor (c \land \bot) \equiv c$</td>
</tr>
<tr>
<td>$d$</td>
<td>$\bot \lor (d \land \bot) \equiv \bot$</td>
</tr>
</tbody>
</table>

Regressing state variables: correctness

Lemma (B)
Let $a$ be a state variable, $o = (c, e)$ an operator, $s$ a state, and $s' = app_o(s)$.
Then $s \models EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))$ if and only if $s' \models a$.

Proof.
($\Rightarrow$): Assume $s \models EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))$.
Do a case analysis on the two disjuncts.
1. Assume that $s \models EPC_a(e)$. By Lemma A, we have $a \in [e]_s$ and hence $s' \models a$.
2. Assume that $s \models a \land \neg EPC_{\neg a}(e)$. By Lemma, we have $A \neg a \not\in [e]_s$.
   Hence $a$ remains true in $s'$.

($\Leftarrow$): We showed that if the formula is true in $s$, then $a$ is true in $s'$. For the second part, we show that if the formula is false in $s$, then $a$ is false in $s'$.
- So assume $s \not\models EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))$.
- Then $s \models \neg EPC_a(e) \land (\neg a \lor EPC_{\neg a}(e))$ (de Morgan).
- Analyze the two cases: $a$ is true or it is false in $s$.
  1. Assume that $s \models a$. Now $s \models EPC_{\neg a}(e)$ because $s \models \neg a \lor EPC_{\neg a}(e)$.
     Hence by Lemma A $\neg a \not\in [e]_s$ and we get $s' \not\models a$.
  2. Assume that $s \not\models a$. Because $s \models \neg EPC_a(e)$, by Lemma A we get $a \not\in [e]_s$ and hence $s' \not\models a$.

Therefore in both cases $s' \not\models a$. 

M. Helmert (Universität Freiburg)
Regression: general definition

We base the definition of regression on formulae $EPC_1(e)$.

Definition (general regression)

Let $\phi$ be a propositional formula and $o = (c, e)$ an operator.

The regression of $\phi$ with respect to $o$ is

$$\text{regr}_{o}(\phi) = c \land \phi_r \land f$$

where

1. $\phi_r$ is obtained from $\phi$ by replacing each $a \in A$ by $EPC_a(e) \lor (a \land \neg EPC_{\neg a}(e))$, and

2. $f = \bigwedge_{a \in A} \neg (EPC_a(e) \land EPC_{\neg a}(e))$.

The formula $f$ says that no state variable may become simultaneously true and false.

Regression example: blocks world

Consider blocks world operators to move blocks $A$ and $B$ onto the table from the other block if they are clear:

$$o_1 = (T, (A\text{-on-}B \land A\text{-clear}) \triangleright (A\text{-on-}T \land B\text{-clear} \land \neg A\text{-on-}B))$$

$$o_2 = (T, (B\text{-on-}A \land B\text{-clear}) \triangleright (B\text{-on-}T \land A\text{-clear} \land \neg B\text{-on-}A))$$

Proof by regression that $o_2, o_1$ puts both blocks onto the table from any blocks world state:

$$G = A\text{-on-}T \land B\text{-on-}T$$

$$\phi_1 = \text{regr}_{o_1}(G) \equiv ((A\text{-on-}B \land A\text{-clear}) \lor A\text{-on-}T) \land B\text{-on-}T$$

$$\phi_2 = \text{regr}_{o_2}(\phi_1) \equiv ((A\text{-on-}B \land ((B\text{-on-}A \land B\text{-clear}) \lor A\text{-clear})) \lor A\text{-on-}T) \land ((B\text{-on-}A \land B\text{-clear}) \lor B\text{-on-}T)$$

All three legal 2-block states satisfy $\phi_2$.

Similar plans exist for any number of blocks.

Regression examples

- $\text{regr}_{(a,b)}(b) \equiv a \land (T \lor (b \land \neg \bot)) \land T \equiv a$
- $\text{regr}_{(a,b)}(b \land c \land d) \equiv a \land (T \lor (b \land \neg \bot)) \land (\bot \lor (c \land \neg \bot)) \land (T \lor (d \land \neg \bot)) \land T \equiv a \land (c \land \neg \bot)$

Regression example: binary counter

$\neg b_0 \triangleright b_0 \land ((\neg b_1 \land b_0) \triangleright (b_1 \land \neg b_0)) \land ((\neg b_2 \land b_1 \land b_0) \triangleright (b_2 \land \neg b_1 \land \neg b_0))$

$EPC_{b_0}(e) = \neg b_0 \land b_0$

$EPC_{b_1}(e) = \neg b_1 \land b_0$

$EPC_{b_2}(e) = \neg b_0$

$EPC_{\neg b_1}(e) = \bot$

$EPC_{\neg b_2}(e) = \neg b_1 \land b_0$

$EPC_{\neg b_2}(e) = (\neg b_1 \land b_0) \lor (\neg b_2 \land b_1 \land b_0) \equiv (\neg b_1 \lor \neg b_2) \land b_0$

Regression replaces state variables as follows:

- $b_2$ by $b_2 \lor (b_2 \land \neg b_0) \lor (b_2 \land \neg \bot) \equiv (b_1 \land b_0) \lor b_2$
- $b_1$ by $b_1 \lor (b_1 \land b_0) \lor (b_1 \land \neg (b_1 \land b_0)) \equiv (b_1 \land b_0) \lor (b_1 \land b_0)$
- $b_0$ by $b_0 \lor (b_0 \land \neg (b_1 \land \neg b_2) \land b_0) \equiv b_0 \lor (b_1 \land b_0)$
General regression: correctness

Theorem (correctness of $\text{regr}_o(\phi)$)

Let $\phi$ be a formula, $o$ an operator, $s$ any state and $s' = \text{app}_o(s)$. Then $s \models \text{regr}_o(\phi)$ if and only if $s' \models \phi$.

Proof.

Let $e$ be the effect of $o$. We show by structural induction over subformulas $\phi'$ of $\phi$ that $s \models \phi'$ iff $s' \models \phi'$, where $\phi'$ is $\phi$ with every $a \in A$ replaced by $EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))$.

The rest of $\text{regr}_o(\phi)$ just states that $o$ is applicable in $s$.

Inductive case 1 & 2 $\phi' = \top$ or $\phi' = \bot$: trivial, as $\phi'_r = \phi'$.

Base case 3 $\phi' = a$ for some $a \in A$.

Then $\phi'_r = EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))$.

By Lemma B, $s \models \phi'_r$ iff $s' \models \phi'$.

$\square$

Emptiness and subsumption testing

The following two tests are useful when performing regression searches, to avoid exploring unpromising branches:

- Testing that a formula $\text{regr}_o(\phi)$ does not represent the empty set (= search is in a dead end).
  For example, $\text{regr}_{(a, \neg p)}(p) \equiv a \wedge \bot \equiv \bot$.

- Testing that a regression step does not make the set of states smaller (= more difficult to reach).
  For example, $\text{regr}_{(b, c)}(a) \equiv a \wedge b$.

Both of these problems are NP-hard.

General regression: correctness

Proof (ctd.)

Inductive case 1 $\phi' = \neg \psi$: By the induction hypothesis $s \models \psi$ iff $s' \models \psi$. Hence $s \models \phi'_r$ iff $s' \models \phi'$ by the logical semantics of $\neg$.

Inductive case 2 $\phi' = \psi \vee \psi'$: By the induction hypothesis $s \models \psi$, iff $s' \models \psi$, and $s \models \psi'$, iff $s' \models \psi'$. Hence $s \models \phi'_r$ iff $s' \models \phi'$ by the logical semantics of $\vee$.

Inductive case 3 $\phi' = \psi \wedge \psi'$: By the induction hypothesis $s \models \psi$, iff $s' \models \psi$, and $s \models \psi'$, iff $s' \models \psi'$. Hence $s \models \phi'_r$ iff $s' \models \phi'$ by the logical semantics of $\wedge$.

$\square$

Formula growth

The formula $\text{regr}_{a_1}(\text{regr}_{a_2}(\ldots \text{regr}_{a_n}(\phi)))$ may have size $O(|\phi| \cdot |a_1| \cdot |a_2| \ldots |a_{n-1}| \cdot |a_n|)$, i.e., the product of the sizes of $\phi$ and the operators.

$\leadsto$ worst-case exponential size $O(m^n)$

Logical simplifications

- $\bot \wedge \phi \equiv \bot$, $T \wedge \phi \equiv \phi$, $\bot \vee \phi \equiv \phi$, $T \vee \phi \equiv T$

- $a \vee \phi \equiv a \vee \phi[\bot/a]$, $\neg a \vee \phi \equiv \neg a \vee \phi[T/a]$, $a \wedge \phi \equiv a \wedge \phi[T/a]$, $\neg a \wedge \phi \equiv \neg a \wedge \phi[\bot/a]$

- idempotency, absorption, commutativity, associativity, . . .
Restricting formula growth in search trees

Problem: Very big formulae obtained by regression.

Cause: Disjunctivity in the formulae: Formulae without disjunctions easily convertible to small formulae \( l_1 \land \cdots \land l_n \) where \( l_i \) are literals and \( n \) is at most the number of state variables.

Idea: Handle disjunctivity when generating search trees.

Alternatives:
1. Do nothing. (May lead to very big formulae!)
2. Always eliminate all disjunctivity.
3. Reduce disjunctivity if formula becomes too big.

Full splitting

- Planners for STRIPS operators only need to use formulae \( l_1 \land \cdots \land l_n \) where \( l_i \) are literals.
- Some general planners also restrict to this class of formulae. This is done as follows:
  1. Transform \( \text{reg}_{\phi}(\phi) \) to disjunctive normal form (DNF):
     \[
     (l_1^1 \land \cdots \land l_n^1) \lor \cdots \lor (l_1^m \land \cdots \land l_n^m).
     \]
  2. Generate one subtree of the search tree for each disjunct \( l_1^i \land \cdots \land l_n^i \).
- The DNF formulae need not exist in its entirety explicitly: can generate one disjunct at a time.
- \( \Rightarrow \) branching is both on the choice of operator and on the choice of the disjunct of the DNF formula.
- \( \Rightarrow \) increased branching factor and bigger search trees, but avoids big formulae.
General splitting strategies

- **With full splitting** search tree can be **exponentially bigger** than without splitting. (But it is not necessary to construct the DNF formulae explicitly!)
- **Without splitting** the formulae may have **size that is exponential** in the number of state variables.
- A compromise is to split formulae only when necessary: combine benefits of the two extremes.
- There are several ways to split a formula $\phi$ to $\phi_1, \ldots, \phi_n$ such that $\phi \equiv \phi_1 \lor \cdots \lor \phi_n$. For example:
  - Transform $\phi$ to $\phi_1 \lor \cdots \lor \phi_n$ by equivalences like distributivity:
    $$(\phi \lor \phi') \land \psi \equiv (\phi \land \psi) \lor (\phi' \land \psi).$$
  - Choose state variable $a$, set $\phi_1 = a \land \phi$ and $\phi_2 = \neg a \land \phi$, and simplify with equivalences like $a \land \psi \equiv a \land [\top/a]$. 

M. Helmert (Universität Freiburg)