

Planning: Techniques for Efficient State-Space Traversal

Jussi Rintanen

Albert-Ludwigs-Universität Freiburg, Germany

July 30, 2005

(Albert-Ludwigs-Universität Freiburg)

1

Introduction

Related research topics

Overlapping

- ▶ Program synthesis
- ▶ Model-Checking and Reachability analysis in Computer-Aided Verification
- ▶ controller synthesis for Discrete-Event Systems (DES)

Orthogonal

- ▶ Reasoning about Action
- ▶ Knowledge Representation
- ▶ ...

(Albert-Ludwigs-Universität Freiburg)

3

Introduction

Why is planning difficult?

- ▶ Solutions to simplest planning problems are **paths from an initial state to a goal state** in the transition graph.
- ▶ **Q:** Why don't we just use Dijkstra's $O(n \log n)$ algorithm?
A: Constructing transition graphs with $10^9, 10^{12}, 10^{15}, \dots$ states is not feasible!
- ▶ Planning with uncertainty very difficult already with state spaces with 1000 or 10000 states.

(Albert-Ludwigs-Universität Freiburg)

5

Introduction

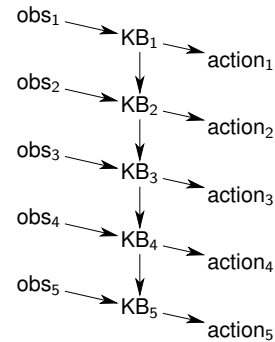
Outline of the tutorial

1. Framework for representing planning problems
2. Approximations of **reachability**: basis of
 - ▶ distance heuristics and planning by heuristic search
 - ▶ techniques for simplifying problem instances
 - ▶ pruning techniques for partial representations (SAT/CSP)
3. Solution techniques:
 - 3.1 heuristic state-space search
 - 3.2 planning by satisfiability testing
 - 3.3 search with logic-based data structures (BDDs)

(Albert-Ludwigs-Universität Freiburg)

7

What is planning?



Planning is **decision making** about which actions to take.

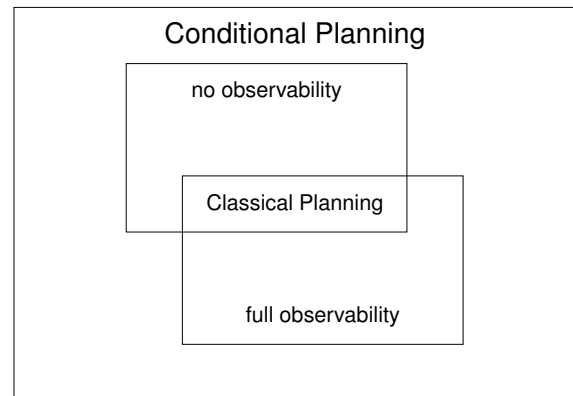
- ▶ knowledge base (KB) about the world
- ▶ general-purpose problem representation (PDDL, logic, ...)
- ▶ algorithms for solving any problem expressible in the representation

(Albert-Ludwigs-Universität Freiburg)

2

Introduction

Types of planning problems



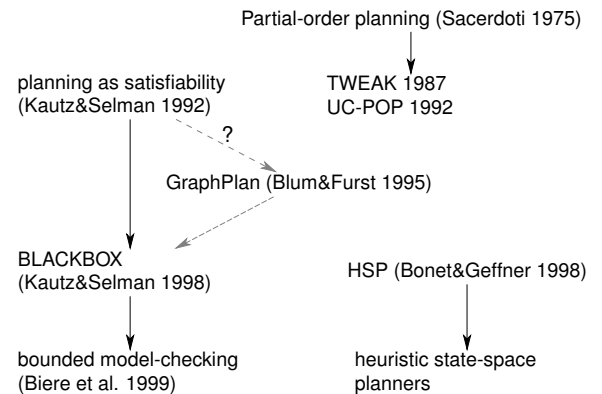
(Albert-Ludwigs-Universität Freiburg)

4

Introduction

Historical timeline

Main approaches to deterministic/classical planning

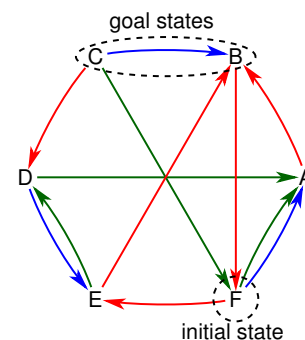


(Albert-Ludwigs-Universität Freiburg)

6

Transition systems

Transition systems

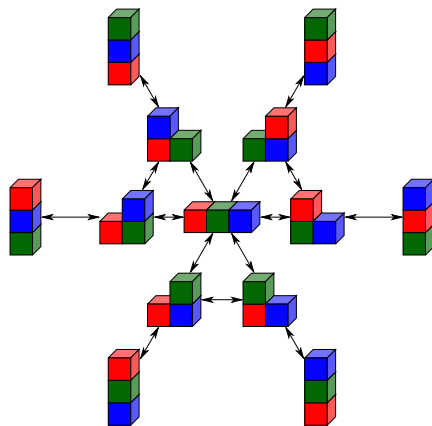


(Albert-Ludwigs-Universität Freiburg)

8

Blocks world

The transition graph for three blocks



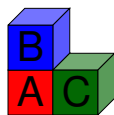
(Albert-Ludwigs-Universität Freiburg)

5

Blocks world with Boolean state variables

Example

$s(AonB)=0$ $s(AonC)=0$ $s(AonTABLE)=1$
 $s(BonA)=1$ $s(BonC)=0$ $s(BonTABLE)=0$
 $s(ConA)=0$ $s(ConB)=0$ $s(ConTABLE)=1$



Not all valuations correspond to an intended state: e.g. state s' such that $s'(AonB) = 1$ and $s'(BonA) = 1$.

(Albert-Ludwigs-Universität Freiburg)

11

Operators

Actions are represented as **operators** $\langle c, e \rangle$ where

- c (**the precondition**) is a propositional formula over A describing the set of states in which the action can be taken. (States in which an arrow starts.)
- e (**the effect**) describes the successor states of states. (Where the arrows go.)
The description is procedural: how do the values of the state variables change?

(Albert-Ludwigs-Universität Freiburg)

13

Connection to PDDL

- The **Planning Domain Description Language PDDL** (Ghallab et al. 1998) is an input language used by many planning systems.
- Our operators are essentially PDDL operators **after instantiation schema variables**.
- $\phi \triangleright e$ corresponds to (when ϕ e).
- $e_1 \wedge \dots \wedge e_n$ corresponds to (and $e_1 \dots e_n$).
- Most features of PDDL that we ignore today are related to instantiating the schema variables: typing, \exists and \forall quantifier

(Albert-Ludwigs-Universität Freiburg)

15

Succinct representation of transition systems

- a state = a valuation of a **finite set** of **finite-valued** state variables

Example

HOUR : $\{0, \dots, 23\} = 13$
 MINUTE : $\{0, \dots, 59\} = 55$
 LOCATION : $\{51, 52, 82, 101, 102\} = 101$
 WEATHER : $\{\text{sunny, cloudy, rainy}\} = \text{cloudy}$
 HOLIDAY : $\{T, F\} = F$

- Any n -valued state variable can be represented by $\lceil \log_2 n \rceil$ Boolean (2-valued) state variables.
- Actions change the values of the state variables.

(Albert-Ludwigs-Universität Freiburg)

10

Representation of sets as formulas

sets	formulas over A
those $\frac{2^{ A }}{2}$ states in which a is true	$a \in A$
$E \cup F$	$E \vee F$
$E \cap F$	$E \wedge F$
$E \setminus F$ (set difference)	$E \wedge \neg F$
\bar{E} (complement)	$\neg E$
the empty set \emptyset	\perp (constant <i>false</i>)
the universal set	\top (constant <i>true</i>)
question about sets	question about formulas
$E \subseteq F?$	$E \models F?$
$E \subset F?$	$E \models F$ and $F \not\models E?$
$E = F?$	$E \models F$ and $F \models E?$

(Albert-Ludwigs-Universität Freiburg)

12

Effects

For deterministic operators

Effects of operators

Defined recursively:

- a and $\neg a$ for state variables $a \in A$ are effects.
These respectively correspond to assignments $a := 1$ and $a := 0$
- $e_1 \wedge \dots \wedge e_n$ is an effect if e_1, \dots, e_n are effects.
Simultaneously execute e_1, \dots, e_n .
The special case with $n = 0$ is the empty conjunction \top .
- $c \triangleright e$ is an effect if c is a formula and e is an effect.
Execute e if c is true.

(Albert-Ludwigs-Universität Freiburg)

14

Operators

Semantics

Changes caused by an operator (= the active effects)

Assign each effect e and state s a set $[e]_s$ of literals as follows.

- $[a]_s = \{a\}$ and $[\neg a]_s = \{\neg a\}$ for $a \in A$.
- $[e_1 \wedge \dots \wedge e_n]_s = [e_1]_s \cup \dots \cup [e_n]_s$.
- $[c \triangleright e]_s = [e]_s$ if $s \models c$ and $[c \triangleright e]_s = \emptyset$ otherwise.

Literals in $[e]_s$ are the **active effects** of e in s .

Applicability of an operator

$\langle c, e \rangle$ is **applicable in a state s** iff $s \models c$ and $[e]_s$ is consistent.

(Albert-Ludwigs-Universität Freiburg)

16

Operators

The successor state of a state

Successor states

The **successor state** $app_o(s)$ of s with respect to $o = \langle c, e \rangle$ is obtained from s by making literals in $[e]_s$ true.

This is defined only if o is applicable in s .

We identify an operator o with the **binary relation** such that sos' iff $s' = app_o(s)$.

Example

$\langle a, \neg a \wedge (\neg c \triangleright \neg b) \rangle$ is applicable in state s such that $s \models a \wedge b \wedge c$ because $s \models a$ and $[\neg a \wedge (\neg c \triangleright \neg b)]_s = \{\neg a\}$ is consistent.

Hence $app_{\langle a, \neg a \wedge (\neg c \triangleright \neg b) \rangle}(s) \models \neg a \wedge b \wedge c$.

Succinct transition systems

Succinct transition system $\langle A, I, O, G \rangle$

- ▶ A is a finite set of **state variables**,
- ▶ I is an **initial state** (a valuation of A),
- ▶ O is a set of **operators** over A ,
- ▶ G is a formula over A that describes the **goal states**.

Planning by state-space search

There are many alternative ways of doing planning by state-space search.

1. different ways of expressing planning as a search problem:
 - 1.1 **search direction**: forward, backward
 - 1.2 **representation** of search space: states, sets of states
2. different **search algorithms**:
 - 2.1 depth-first, breadth-first, bidirectional, ...
 - 2.2 heuristic search (**systematic**: A*, IDA*, ...; **local**: hill-climbing, simulated annealing, ...), ...
3. different ways of controlling search:
 - 3.1 **heuristics** for heuristic search algorithms
 - 3.2 **pruning techniques**: invariants, symmetry elimination, ...

Progression

- ▶ **Progression** means computing the successor state $app_o(s)$ of s with respect to $o = \langle c, e \rangle$. This is simply by updating s with $[e]_s$.
- ▶ Used in **forward search**: from the initial state toward the goal states.
- + Very easy and efficient to implement.
- Search with only one state at a time.

Operators

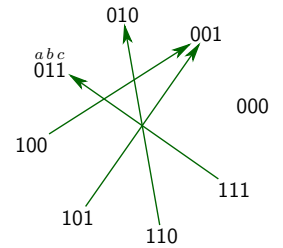
Example

State variables are $A = \{a, b, c\}$.

The operator

$$\langle a, \neg a \wedge (\neg b \triangleright c) \rangle$$

corresponds to the binary relation on the right.



Plans

Plans

A **plan** for $\langle A, I, O, G \rangle$ is a sequence $\pi = o_1, \dots, o_n$ of operators such that $o_1, \dots, o_n \in O$ and there is a sequence of states s_0, \dots, s_n (the **execution** of π) so that

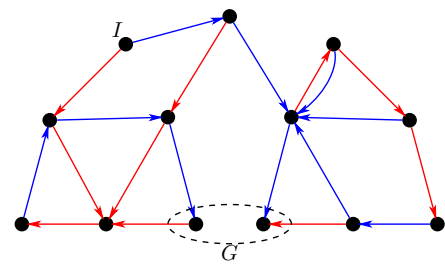
1. $s_0 = I$,
2. $s_i = app_{o_i}(s_{i-1})$ for every $i \in \{1, \dots, n\}$, and
3. $s_n \models G$.

This can be equivalently expressed as

$$app_{o_n}(app_{o_{n-1}}(\dots app_{o_1}(I) \dots)) \models G$$

Planning by forward search

with depth-first search



Regression

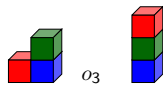
- ▶ **Regression** = computation of predecessors of states
- + Advantage over progression: a formula represents a **set of state**
- More difficult to implement efficiently.

Regression for STRIPS operators

- ▶ Goals are conjunctions of literals $l_1 \wedge \dots \wedge l_n$.
- ▶ **First step**: Choose an operator that makes some of l_1, \dots, l_n true and makes none of them false.
- ▶ **Second step**: Form a new goal by removing the fulfilled goal literals and adding the preconditions of the operator.

Regression for STRIPS operators

Example



$$o_3 = \langle \text{onT} \wedge \text{clr} \wedge \text{clr}, \neg \text{clr} \wedge \neg \text{onT} \wedge \text{on} \rangle$$

$$G = \text{on} \wedge \text{on}$$

$$\phi_1 = \text{regr}_{o_3}^{str}(G) = \text{on} \wedge \text{onT} \wedge \text{clr} \wedge \text{clr}$$

Condition for l to be an active effect: $EPC_l(e)$

Definition

EPC

The condition $EPC_l(e)$ for literal l to be an active effect of e is defined as follows.

$$\begin{aligned} EPC_l(l) &= \top \\ EPC_l(l') &= \perp \text{ when } l \neq l' \text{ (for literals } l') \\ EPC_l(\top) &= \perp \\ EPC_l(e_1 \wedge \dots \wedge e_n) &= EPC_l(e_1) \vee \dots \vee EPC_l(e_n) \\ EPC_l(c \triangleright e) &= c \wedge EPC_l(e) \end{aligned}$$

Important property of EPC

For states s , literals l and effects e ,

$$l \in [e]_s \text{ if and only if } s \models EPC_l(e).$$

Regression

Definition for state variables

Regressing a state variable

The formula $EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))$ expresses the value of $a \in .$ after executing e in terms of values of state variables before executing e : Either

- a becomes true, or
- a was true before and it does not become false.

Regression

General definition

Regression

The regression of formula ϕ with respect to $o = \langle c, e \rangle$ is

$$\text{regr}_o(\phi) = \phi' \wedge c \wedge \bigwedge_{a \in A} \neg (EPC_a(e) \wedge EPC_{\neg a}(e))$$

where

- ϕ' is obtained from ϕ by replacing each $a \in A$ by $EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))$, and
- the last conjunct says that no state variable may become simultaneously true and false.

Regression for general operators

- With disjunction and conditional effects things become trickier. How to regress $a \vee (b \wedge c)$ 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.

Condition for effect l to take place: $EPC_l(e)$

Example

Example

$$\begin{aligned} EPC_a(b \wedge c) &= \perp \vee \perp \equiv \perp \\ EPC_a((b \triangleright a) \wedge (c \triangleright a)) &= (b \wedge \top) \vee (c \wedge \top) \equiv b \vee c \\ EPC_a(b \triangleright (c \triangleright a)) &= b \wedge (c \wedge \top) \equiv b \wedge c \end{aligned}$$

Regression

Definition for state variables

Example

Let $e = (b \triangleright a) \wedge (c \triangleright \neg a) \wedge b$.

$$\begin{array}{l|l} x & EPC_x(e) \vee (x \wedge \neg EPC_{\neg x}(e)) \\ \hline a & b \vee (a \wedge \neg c) \\ b & \top \vee (b \wedge \neg \perp) \equiv \top \\ c & \perp \vee (c \wedge \neg \perp) \equiv c \end{array}$$

Regression

Examples

- $\text{regr}_{(a,b)}(b) = (a \wedge (\top \vee (b \wedge \neg \perp))) \equiv a$
- $\text{regr}_{(a,b)}(b \wedge c) = (a \wedge (\top \vee (b \wedge \neg \perp))) \wedge (\perp \vee (c \wedge \neg \perp)) \equiv a \wedge c$
- $\text{regr}_{(a,b)}(b \vee c) = (a \wedge ((\top \vee (b \wedge \neg \perp)) \vee (\perp \vee (c \wedge \neg \perp)))) \equiv a$
- $\text{regr}_{(a,c \triangleright b)}(b) = (a \wedge (c \vee (b \wedge \neg \perp))) \equiv a \wedge (c \vee b)$

Important property of regression

Let ϕ be a formula over A , o an operator over A , and S the set of all states i.e. valuations of A . Then $\{s \in S | s \models \text{regr}_o(\phi)\} = \{s \in S | \text{app}_o(s) \models \phi\}$.

Regression

Generation of search trees

Problem Formulas obtained with regression may become big.

Cause **Disjunctivity** in the formulas. Formulas **without disjunctions** easily convertible to small formulas $l_1 \wedge \dots \wedge l_n$ where l_i are literals and n is at most the number of state variables.

Solution Handle disjunctivity when generating search trees.

Alternatives:

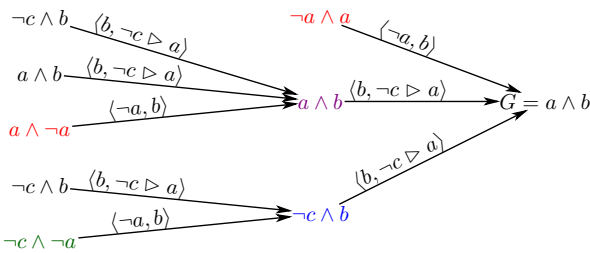
1. Do nothing. (May lead to very big formulas!)
2. Eliminate all disjunctivity (Anderson, Smith, Weld 1998)
3. Reduce disjunctivity only if formula becomes too big.

Regression: generation of search trees

One branch for every disjunct of $\text{regr}_o(\phi)$ in DNF

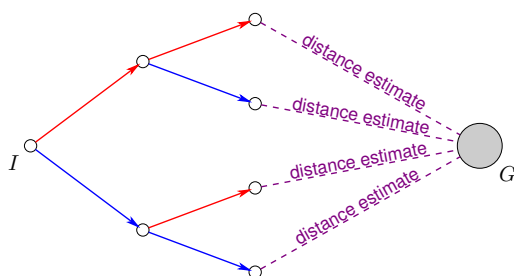
Reach goal $a \wedge b$ from state I such that $I \models \neg a \wedge \neg b \wedge \neg c$.

$\text{regr}_{(b, \neg c \triangleright a)}(G) = (\neg c \vee a) \wedge b$ in DNF is $(\neg c \wedge b) \vee (a \wedge b)$. It is split to $\neg c \wedge b$ and $a \wedge b$.



Planning by heuristic search

Forward search



The formula $\text{regr}_{o_1}(\text{regr}_{o_2}(\dots \text{regr}_{o_{n-1}}(\text{regr}_{o_n}(\phi))))$ has size $\mathcal{O}(|\phi| |o_1| |o_2| \dots |o_{n-1}| |o_n|)$, i.e. the product of the sizes of ϕ and the operators.

The worst-case size is $\mathcal{O}(2^n)$.

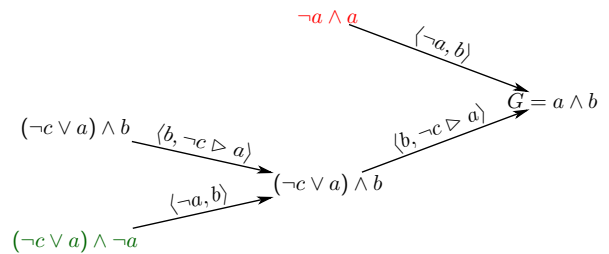
Simplifications

1. $\perp \wedge \phi \equiv \perp$, $\top \wedge \phi \equiv \phi$, $\perp \vee \phi \equiv \phi$, $\top \vee \phi \equiv \top$
2. $a \vee \phi \equiv a \vee \phi[\perp/a]$, $\neg a \vee \phi \equiv a \vee \phi[\top/a]$, $a \wedge \phi \equiv a \wedge \phi[\top/a]$, $\neg a \wedge \phi \equiv a \wedge \phi[\perp/a]$
3. Associativity and commutativity for \vee and \wedge .

Regression: generation of search trees

Unrestricted regression (= do nothing about formula size)

Reach goal $a \wedge b$ from state I such that $I \models \neg a \wedge \neg b \wedge \neg c$.

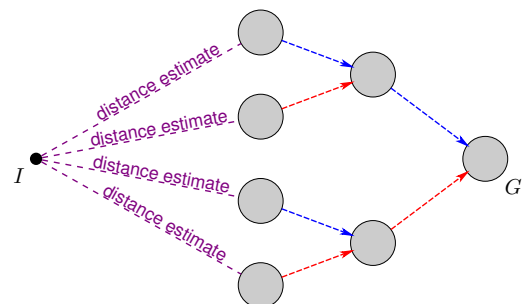


Planning by exhaustive state-space search

- ▶ Idea: exhaustively generate all states that are reachable from the initial state (by progression).
- ▶ Best implementations are by the **computer-aided verification** community for **reachability analysis** and **model-checking**: SPIN, Murphi, ...
- ▶ Has not been much used in planning.
- ▶ Applicable to problems with up to 10^6 , 10^7 states.
- ▶ Most efficient and **reliable** technique to find plans if the state space is small: runtime an almost linear function of the number of states; cost per state is very small.

Planning by heuristic search

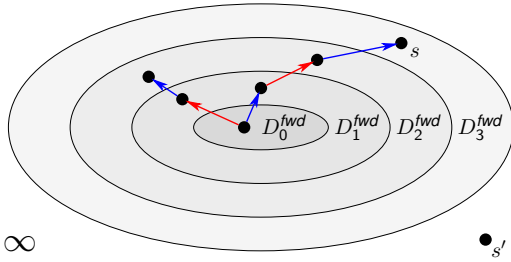
Backward search



Distances

Illustration

Forward distance of state s is 3 because $s \in D_3^{fwd} \setminus D_2^{fwd}$.



As $D_i^{fwd} = D_3^{fwd}$ for all $i > 3$, forward distance of state s' is ∞ .

(Albert-Ludwigs-Universität Freiburg)

42

Distances Distance sets

Distances

Forward distance sets

Let I be a state and O a set of operators. Define the **forward distance sets** D_i^{fwd} for I, O by

$$D_0^{fwd} = \{I\}$$

$$D_i^{fwd} = D_{i-1}^{fwd} \cup \bigcup_{o \in O} \text{img}_o(D_{i-1}^{fwd}) \text{ for all } i \geq 1$$

(Albert-Ludwigs-Universität Freiburg)

44

Heuristics Max-heuristic

Distance estimation with max-distances

- ▶ Bonet & Geffner proposed the **max-heuristic** for controlling heuristic search algorithms like A^* , IDA^* .
- ▶ Max-distances are a **lower bound** for forward distances: since they do not overestimate they are **admissible** as a heuristic.
- ▶ They can be used with A^* and IDA^* to find **optimal plans**.
- ▶ Basic insight: estimate distances one state variable at a time.

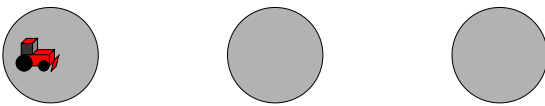
(Albert-Ludwigs-Universität Freiburg)

46

Heuristics Max-heuristic

Distance estimation with max-distances

Tractor example



t	T1	T2	T3	A1	A2	A3	B1	B2	B3
0	T	F	F	F	F	T	F	F	T
1	TF	TF	F	F	F	T	F	F	T
2	TF	TF	TF	F	F	T	F	F	T
3	TF	TF	TF	F	TF	TF	F	TF	TF
4	TF	TF	TF	TF	TF	TF	TF	TF	TF

Distance of $A1 \wedge B1$ is 4.

(Albert-Ludwigs-Universität Freiburg)

48

Images

Images

The **image** of a state s with respect to an action o is

$$\text{img}_o(s) = \{s' | sos'\}.$$

This can be generalized to sets T of states:

$$\text{img}_o(T) = \bigcup_{s \in T} \text{img}_o(s)$$

(Albert-Ludwigs-Universität Freiburg)

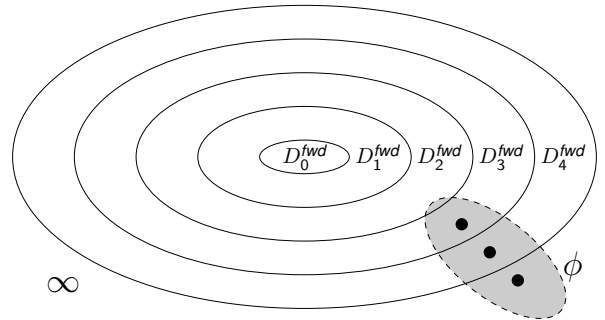
45

Distances Distance sets

Distances

of formulas

$\delta_I^{fwd}(\phi) = 3$ since $s \models \phi$ for some $s \in D_3^{fwd}$ and for no $s \in D_2^{fwd}$.



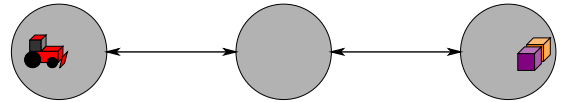
(Albert-Ludwigs-Universität Freiburg)

45

Heuristics Max-heuristic

Distance estimation with max-distances

Tractor example



1. Tractor moves:
 - 1.1 from 1 to 2: $T12 = \langle T1, T2 \wedge \neg T1 \rangle$
 - 1.2 from 2 to 1: $T21 = \langle T2, T1 \wedge \neg T2 \rangle$
 - 1.3 from 2 to 3: $T23 = \langle T2, T3 \wedge \neg T2 \rangle$
 - 1.4 from 3 to 2: $T32 = \langle T3, T2 \wedge \neg T3 \rangle$
2. Tractor pushes A:
 - 2.1 from 2 to 1: $A21 = \langle T2 \wedge A2, T1 \wedge A1 \wedge \neg T2 \wedge \neg A2 \rangle$
 - 2.2 from 3 to 2: $A32 = \langle T3 \wedge A3, T2 \wedge A2 \wedge \neg T3 \wedge \neg A3 \rangle$
3. Tractor pushes B:
 - 3.1 from 2 to 1: $B21 = \langle T2 \wedge B2, T1 \wedge B1 \wedge \neg T2 \wedge \neg B2 \rangle$
 - 3.2 from 3 to 2: $B32 = \langle T3 \wedge B3, T2 \wedge B2 \wedge \neg T3 \wedge \neg B3 \rangle$

(Albert-Ludwigs-Universität Freiburg)

47

Heuristics Max-heuristic

Sets of literals representing sets of states

t	T1	T2	T3	A1	A2	A3	B1	B2	B3
0	T	F	F	F	F	T	F	F	T
1	TF	TF	F	F	F	T	F	F	T
2	TF	TF	TF	F	F	T	F	F	T
3	TF	TF	TF	F	TF	TF	F	TF	TF
4	TF	TF	TF	TF	TF	TF	TF	TF	TF

These lines are best represented as **sets of literals**.

$$D_0^{max} = \{T1, \neg T2, \neg T3, \neg A1, \neg A2, A3, \neg B1, \neg B2, B3\}$$

$$D_1^{max} = \{\neg T3, \neg A1, \neg A2, A3, \neg B1, \neg B2, B3\}$$

$$D_2^{max} = \{\neg A1, \neg A2, A3, \neg B1, \neg B2, B3\}$$

$$D_3^{max} = \{\neg A1, \neg B1\}$$

$$D_4^{max} = \{\}$$

The sets D_i^{max} approximate **forward distance sets**:

$$D_i^{fwd} \subseteq \{s \in S | s \models D_i^{max}\} \text{ for all } i \geq 0.$$

(Albert-Ludwigs-Universität Freiburg)

48

Distances of literals

EPC for operator application

$$EPC_i((c, e)) = EPC_i(e) \wedge c \wedge \bigwedge_{a \in A} \neg(EPC_a(e) \wedge EPC_{\neg a}(e))$$

Computation of literal sets for max-distances

Let $L = A \cup \{\neg a | a \in A\}$ be the set of literals over A . Define the sets D_i^{max} for $i \geq 0$ as follows.

$$D_0^{max} = \{l \in L | I \models l\}$$

$$D_i^{max} = D_{i-1}^{max} \setminus \{l \in L | o \in O, SAT(D_{i-1}^{max} \cup \{EPC_l(o)\})\}$$

Max-distances underestimate

Example

Estimated distance of $\text{lamp1on} \wedge \text{lamp2on} \wedge \text{lamp3on}$ with

- $\langle T, \text{lamp1on} \rangle$
- $\langle T, \text{lamp2on} \rangle$
- $\langle T, \text{lamp3on} \rangle$

is 1. Actual distance is 3.

In general, the max-distance estimate of $G_1 \wedge \dots \wedge G_n$ is the **maximum** of the estimates for G_1, \dots, G_n .

If goals are independent, the **sum** of the estimates is more accurate.

Distance estimation with relaxed plans

Motivation

operators	distance estimate of $a \wedge b \wedge c$		actual
	max	sum	
$\langle T, a \wedge b \wedge c \rangle$	1	3	1
$\langle T, a \rangle, \langle T, b \rangle, \langle T, c \rangle$	1	3	3

- ▶ To get better estimates the properties of the operators have to be observed.
- ▶ Hoffmann & Nebel (2001) proposed a method for approximately counting the number of operators needed to achieve the goals.
- ▶ This method may both overestimate and underestimate the distance (just like the sum-heuristic.) It is **not admissible**.

Comparison of the heuristics

- ▶ For the Tractor example:
 - ▶ max-distance is 4 (never overestimates).
 - ▶ operators in relaxed plan: 6 (may under or overestimate).
 - ▶ operators in actual shortest plan: 8.
 - ▶ sum-distance is 10 (may under or overestimate).
- ▶ None of the heuristics properly **dominates** any other.
- ▶ The sum-heuristic and the relaxed plan heuristic are used in practice for non-optimal planners.
- ▶ Only the max-heuristic is **admissible**. There are more accurate admissible generalizations of the max-heuristic (Haslum & Geffr 2000).

Max-distances of literals, states and formulas

Max-distances of formulas

The **max-distance** of a formula ϕ (from I with O) is

$$\delta_I^{max}(\phi) = \begin{cases} 0 & \text{if } SAT(D_0^{max} \cup \{\phi\}) \\ d & \text{if } SAT(D_d^{max} \cup \{\phi\}) \text{ and not } SAT(D_{d-1}^{max} \cup \{\phi\}) \\ & \text{for } d \geq 1 \end{cases}$$

Application of max-distances

1. With backward search, estimate the distance of a regressed formula ϕ from the initial state I by $\delta_I^{max}(\phi)$.
2. With forward search, estimate the distance from a progressed state s to the goals G by $\delta_s^{max}(G)$.

Distance estimation with sum-distances

Tractor example

t	T1	T2	T3	A1	A2	A3	B1	B2	B3
0	T	F	F	F	F	T	F	F	T
1	TF	TF	F	F	F	T	F	F	T
2	TF	TF	TF	F	F	T	F	F	T
3	TF	TF	TF	F	TF	TF	F	TF	TF
4	TF	TF	TF	F	TF	TF	F	TF	TF
5	TF	TF	TF	TF	TF	TF	TF	TF	TF

Apply $A21 = \langle T2 \wedge A2, T1 \wedge A1 \wedge \neg T2 \wedge \neg A2 \rangle$

Distance of $T2 \wedge A2$ is 1+3.

Hence distance of $A1$ is 1+3+1 = 5 (max-distance is 4.) Distance $A1 \wedge B1$ is 5 + 5 = 10.

Distance estimation with relaxed plans

t	T1	T2	T3	A1	A2	A3	B1	B2	B3
0	T	F	F	F	F	T	F	F	T
1	TF	TF	F	F	F	T	F	F	T
2	TF	TF	TF	F	F	T	F	F	T
3	TF	TF	TF	F	TF	TF	F	TF	TF
4	TF	TF	TF	TF	TF	TF	TF	TF	TF

Estimate distance of $A1 \wedge B1$ by finding a relaxed plan:

t	relaxed plan
0	T12
1	T23
2	A32, B32
3	A21, B21

Distance estimate = number of operators in relaxed plan = 6

Invariants

Motivation

Example

Consider the goal formula

$$AonB \wedge BonC$$

regressed with operator

$$\langle AonC \wedge Aclear \wedge Bclear, AonB \wedge \neg Bclear \wedge Cclear \rangle$$

giving the new goal

$$AonC \wedge Aclear \wedge Bclear \wedge BonC.$$

No state satisfying this formula is reachable from any legal blocks world state.

Invariants

Goal: Restriction to states that are reachable.

Problem: Testing reachability is computationally as complex as testing whether a plan exists.

Solution: Use an **approximate** notion of reachability.

Implementation: Compute in polynomial time **formulas** that characterize a **superset** of the reachable states. Computation of C_0, C_1, C_2, \dots generalizes the computation of $D_0^{max}, D_1^{max}, D_2^{max}, \dots$. Formulas in $\bigcap_{i \geq 0} C_i$ are **invariants**.

Preservation test and weakening

Does ϕ preserve truth of $l_1 \vee \dots \vee l_m$ when C is true

For every l in $l_1 \vee \dots \vee l_m$ test if for some $o = \langle c, e \rangle \in O$

1. l can become false: is $C \cup \{EPC_{\neg l}(o)\}$ satisfiable?
2. If it can, will some other l' in $l_1 \vee \dots \vee l_m$ still be true: is $C \cup \{EPC_l(o)\} \models EPC_{l'}(e) \vee (l' \wedge \neg EPC_{\neg l}(e))$ satisfiable?

If $l_1 \vee \dots \vee l_m$ cannot be proved to be true after executing o it will be removed.

Weaken $\phi = l_1 \vee \dots \vee l_m$

If ϕ is removed and $m + 1 \leq n$, add all such $\phi \vee a$ and $\phi \vee \neg a$ for $a \in$ that are **preserved** by all operators.

If ϕ is not an invariant, **weaker** formulas $\phi \vee a$ and $\phi \vee \neg a$ might still b

Pruning techniques for state-space search

- ▶ Many pruning techniques for planning have been proposed but most of them are only applicable to a narrow class of problems (usually some of the standard benchmarks.)
- ▶ Many problems have **symmetries**: many objects are **interchangeable** and actions related to them are as well, which makes transition systems symmetric.
- ▶ 10 interchangeable objects may induce a symmetric state space which every state is symmetric with $10! - 1 \sim 3 \cdot 10^6$ other states. Easy problems become difficult to solve if symmetries are not recognized.

Symmetry reduction

Main approaches

Modeling Planning problems can be **modeled** so that symmetries do not show up explicitly.

State-space search Search takes place in the **quotient state space**: the equivalence class $[s_1] = \{s_1, s_2, \dots, s_n\}$ of states that are symmetric with s_1 is represented by its **canonical element** s_1 . Every encountered state s' is mapped to the canonical element s of its equivalence class $[s'] = [s]$. (Starke 1991; Emerson & Sistla 1996)

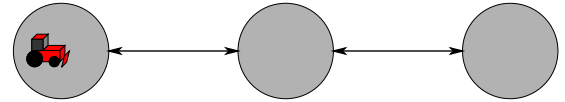
SAT planning Symmetry-breaking constraints (Joslin & Roy 1996, ...) for SAT/CSP.

Computation of invariants

Example

Similar to distance estimation with D_i^{max} : compute sets C_i of **n -literal clauses** characterizing (giving an **upper bound!**) the states that are reachable in i steps.

Example



$n = 2$ (maximum clause length)

$C_0 = \{T1, \neg T2, \neg T3\} \sim \{100\}$

$C_1 = \{T1 \vee T2, \neg T1 \vee \neg T2, \neg T3\} \sim \{010, 100\}$

$C_2 = \{\neg T1 \vee \neg T2, \neg T1 \vee \neg T3, \neg T2 \vee \neg T3\} \sim \{000, 001, 010, 100\}$

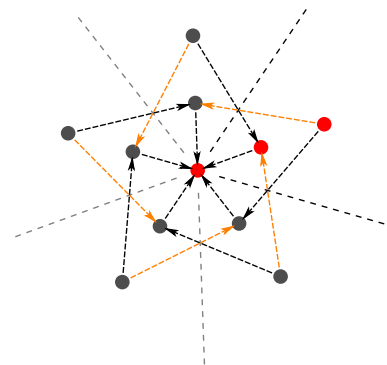
$C_i = C_2$ for all $i \geq 3$

Summary

- ▶ Distance heuristics:
 - ▶ Approximations of the **distances** between states.
 - ▶ Max-heuristic: admissible, not very informative
 - ▶ Sum-heuristic: more informative, may over and underestimate
 - ▶ relaxed plans: more informative, may over and underestimate
- ▶ Invariants:
 - ▶ More exact characterization of reachability
 - ▶ Application: pruning search space for backward search, planning as satisfiability
 - ▶ Efficient polynomial time algorithms exist.

Symmetry reduction

Example



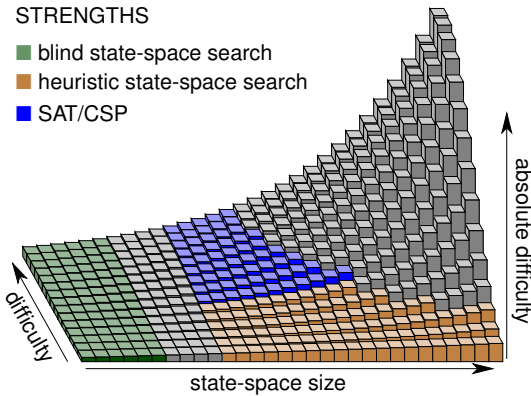
When does heuristic state-space search fail?

- ▶ Distance heuristics work well only on easy problems.
 - ▶ On **hard instances** of NP-hard problems heuristics necessarily fail (otherwise they would solve NP-hard problems efficiently).
 - ▶ Many easy problems have a structure that confuses the heuristic.
- ▶ When heuristics fail, blind state-space search is more efficient (because of much lower cost per state) but only applicable to problems with up to $10^6 \dots 10^7$ states.
- ▶ What is needed: techniques that do search with **many** states or transition sequences at a time.

Relative strengths of different approaches

STRENGTHS

- blind state-space search
- heuristic state-space search
- SAT/CSP



(Albert-Ludwigs-Universität Freiburg)

66

SAT Planning Relations in PL

Sets (of states) as formulas

Formulas over A represent sets $a \vee b$ over $A = \{a, b, c\}$ represents the set $\{010, 011, 100, 101, 110, 111\}$.Formulas over $A \cup A'$ represent binary relations $a \wedge a' \wedge (b \leftrightarrow b')$ over $A \cup A'$ where $A = \{a, b\}$, $A' = \{a', b'\}$ represents the binary relation $\{(10, 10), (11, 11)\}$.Valuations 1010 and 1111 of $A \cup A'$ can be viewed respectively aspairs of valuations $(10, 10)$ and $(11, 11)$ of A .

(Albert-Ludwigs-Universität Freiburg)

66

SAT Planning Plans in PL

Planning as satisfiability

Transition relation in the propositional logic

For $\langle A, I, O, G \rangle$ define

$$\mathcal{R}_1(A, A') = \bigvee_{o \in O} \tau_A(o) \quad (+\text{invariants}).$$

Plans of length t in the propositional logicPlans of length t correspond to satisfying valuations of

$$\Phi_t^{seq} = I^0 \wedge \mathcal{R}_1(A^0, A^1) \wedge \mathcal{R}_1(A^1, A^2) \wedge \dots \wedge \mathcal{R}_1(A^{t-1}, A^t) \wedge G^t$$

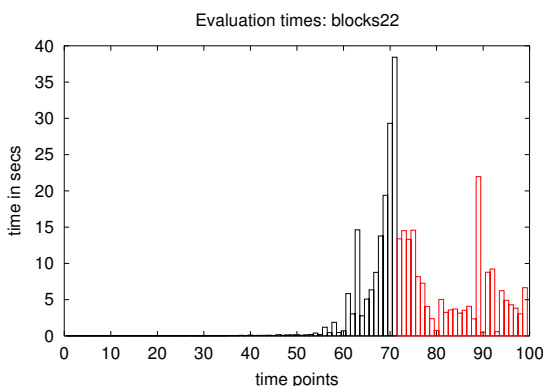
where $I^0 = \bigwedge \{a^0 | a \in A, I(a) = 1\} \cup \{\neg a^0 | a \in A, I(a) = 0\}$ and G^t is $\bigwedge \{a^t | a \in A, G(a) = 1\} \cup \{\neg a^t | a \in A, G(a) = 0\}$ with propositional variables a replaced by a^t and $A^i = \{a^i | a \in A\}$ for $i \in \{0, \dots, t\}$

(Albert-Ludwigs-Universität Freiburg)

70

SAT Planning Runtime profiles

Profile of runtimes for different plan lengths



(Albert-Ludwigs-Universität Freiburg)

72

Planning in the propositional logic

- ▶ Great idea by Henry Kautz and Bart Selman (1992): planning by **satisfiability testing** in the propositional logic.
- ▶ Benefits:
 1. Plan search constrained to plans of given length n : more constraints, more inferences, more pruning.
 2. High numbers of plans can be considered without constructing all them explicitly.
 3. Very efficient SAT solvers: zChaff, Siege, BerkMin, ...
- ▶ **Bounded model-checking** (1999-, M-USD business) in Compute Aided Verification is a direct offspring of **planning as satisfiability**

(Albert-Ludwigs-Universität Freiburg)

67

SAT Planning Ops in PL

Translating operators into formulas

Operators as formulas

Let $o = \langle c, e \rangle$ be an operator and A a set of state variables.

$$\tau_A(o) = \bigwedge_{a \in A} \bigwedge_{c} (EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))) \leftrightarrow a' \quad (1)$$

$$\bigwedge_{a \in A} \neg (EPC_a(e) \wedge EPC_{\neg a}(e)) \quad (3)$$

(2) says the **new value of a** is 1 if it was assigned 1 **or** the the old val was 1 and it was not assigned 0.

(3) says no state variable is assigned both 0 and 1.

Determinism

The value of every $a' \in A'$ is a function of the valuation of A = the operator is **deterministic** & the binary relation is **a partial function**.

(Albert-Ludwigs-Universität Freiburg)

66

SAT Planning Plans in PL

Planning as satisfiability

Example

Example

Consider

$$\begin{aligned} I &= b \wedge c \\ G &= (b \wedge \neg c) \vee (\neg b \wedge c) \\ o_1 &= \langle T, (c \triangleright \neg c) \wedge (\neg c \triangleright c) \rangle \\ o_2 &= \langle T, (b \triangleright \neg b) \wedge (\neg b \triangleright b) \rangle. \end{aligned}$$

Formula for plans of length 3 is

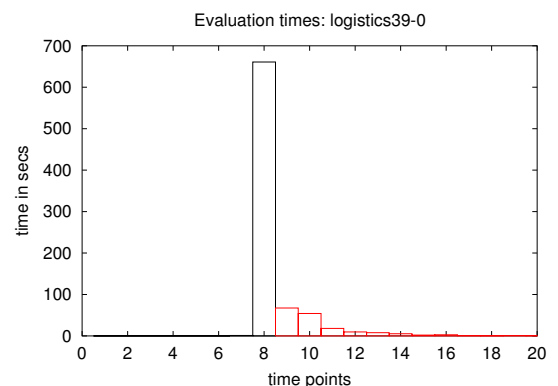
$$\begin{aligned} &(b^0 \wedge c^0) \\ &\wedge (((b^0 \leftrightarrow b^1) \wedge (c^0 \leftrightarrow \neg c^1)) \vee ((b^0 \leftrightarrow \neg b^1) \wedge (c^0 \leftrightarrow c^1))) \\ &\wedge (((b^1 \leftrightarrow b^2) \wedge (c^1 \leftrightarrow \neg c^2)) \vee ((b^1 \leftrightarrow \neg b^2) \wedge (c^1 \leftrightarrow c^2))) \\ &\wedge (((b^2 \leftrightarrow b^3) \wedge (c^2 \leftrightarrow \neg c^3)) \vee ((b^2 \leftrightarrow \neg b^3) \wedge (c^2 \leftrightarrow c^3))) \\ &\wedge ((b^3 \wedge \neg c^3) \vee (\neg b^3 \wedge c^3)). \end{aligned}$$

(Albert-Ludwigs-Universität Freiburg)

71

SAT Planning Runtime profiles

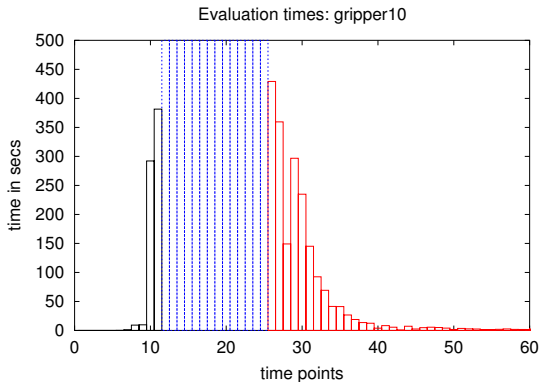
Profile of runtimes for different plan lengths



(Albert-Ludwigs-Universität Freiburg)

73

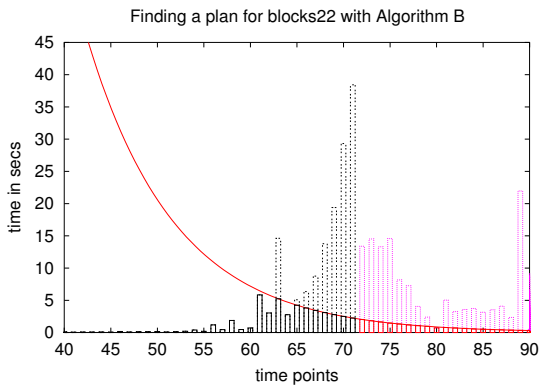
Profile of runtimes for different plan lengths



(Albert-Ludwigs-Universität Freiburg)

74

A parallel evaluation strategy



(Albert-Ludwigs-Universität Freiburg)

80

Parallel operator application

Representation in the propositional logic

The formula for operator $o = \langle c, e \rangle$

$$\tau_A(o) = \wedge \bigwedge_{a \in A} ((EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))) \leftrightarrow a') \\ \wedge \bigwedge_{a \in A} \neg (EPC_a(e) \wedge EPC_{\neg a}(e)).$$

is equivalent to

$$\wedge \bigwedge_{a \in A} (EPC_a(e) \rightarrow a') \\ \wedge \bigwedge_{a \in A} (EPC_{\neg a}(e) \rightarrow \neg a') \\ \wedge \bigwedge_{a \in A} ((a \wedge \neg EPC_{\neg a}(e)) \rightarrow a') \\ \wedge \bigwedge_{a \in A} ((\neg a \wedge \neg EPC_a(e)) \rightarrow \neg a').$$

This separates the **changes** from **non-changes**.

(Albert-Ludwigs-Universität Freiburg)

85

Interpretation of parallelism

Example

Operators $\langle a, \neg b \rangle$ and $\langle b, \neg a \rangle$ have non-contradictory effects and preconditions. However, neither

1. $\langle b, \neg a \rangle, \langle a, \neg b \rangle$ nor
2. $\langle a, \neg b \rangle, \langle b, \neg a \rangle$

is executable.

First interpretation of parallelism

Operators have to be **executable in every order**.

(Albert-Ludwigs-Universität Freiburg)

87

Parallel evaluation strategies

- ▶ The plan search strategy that performs satisfiability tests of formulas for lengths 0,1,2,... until a satisfiable formula (plan) is found always **proves that the plan** has optimal length.
- ▶ Usually **proving optimality is much more difficult than finding a plan**.
- ▶ This is the reason why planning by satisfiability has not been considered competitive for **non-optimal planning**.
- ▶ There are parallel evaluation strategies that very effectively avoid the optimality proofs (Rintanen 2004).

(Albert-Ludwigs-Universität Freiburg)

75

Planning as satisfiability with parallel plans

- ▶ **Efficiency** of satisfiability planning is strongly **dependent on the plan length** because satisfiability algorithms have runtime $O(2^n)$ where n is the formula size, and formula sizes are linearly proportional to plan length.
- ▶ Formula sizes can be reduced by allowing **several operators in parallel**.
- ▶ On many problems this leads to big speed-ups.

(Albert-Ludwigs-Universität Freiburg)

84

Parallel actions

Representation in the propositional logic

Propositional variables for operator application

For each operator o there is a propositional variable o that is true if and only if the operator is executed.

Translation of operators

For the set O of all operators let $\tau_A(O)$ be the conjunction of

$$(o \rightarrow c) \wedge \\ \bigwedge_{a \in A} (o \wedge EPC_a(e) \rightarrow a') \wedge \\ \bigwedge_{a \in A} (o \wedge EPC_{\neg a}(e) \rightarrow \neg a')$$

for all $o = \langle c, e \rangle \in O$ and the **explanatory frame axioms**

$$\bigwedge_{a \in A} (a \wedge \neg a') \rightarrow ((o_1 \wedge EPC_{\neg a}(e_1)) \vee \dots \vee (o_n \wedge EPC_{\neg a}(e_n)))$$

$$\bigwedge_{a \in A} (\neg a \wedge a') \rightarrow ((o_1 \wedge EPC_a(e_1)) \vee \dots \vee (o_n \wedge EPC_a(e_n)))$$

where $O = \{o_1, \dots, o_n\}$ and e_1, \dots, e_n are their respective effects.

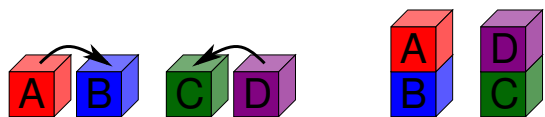
(Albert-Ludwigs-Universität Freiburg)

86

Interference

Example

Actions do not interfere



Actions can be taken simultaneously.

Actions interfere



If A is moved first, B won't be clear and cannot be moved.

(Albert-Ludwigs-Universität Freiburg)

88

Interference

Interference

Two operators o and o' **interfere** if

- o may make the precondition of o' false, or
- o may change the set of active effects of o' ,

or the other way round.

Example

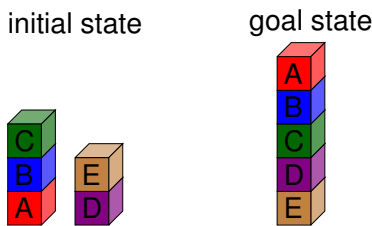
$\langle c, d \rangle$ and $\langle \neg d, f \rangle$ interfere.
 $\langle c, d \rangle$ and $\langle d, f \rangle$ do not interfere.

Important property of interference

Any set of **non-interfering** operators that are simultaneously applicable in a state s can be executed in **any order**, leading to the same state in all cases.

Planning as satisfiability

Example



Problem solved almost without search:

- Formulas for lengths 1 to 4 shown unsatisfiable without any search.
- Formula for plan length 5 is satisfiable: 3 nodes in the search tree
- Plans have 5 to 7 operators, optimal plan has 5.

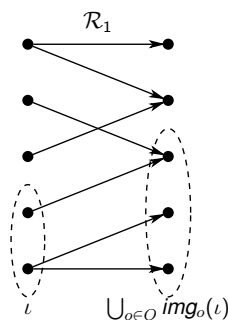
Other notions of parallel plans

- Non-interference** is too strong a requirement: Why require possibility of **all execution orders**?
- It suffices that **at least one execution order is possible** (Dimopoulos et al. 1997).
- Efficient translations of this notion of parallel plans lead to much faster planning (Rintanen et al. 2004).

Sets and transition relations “symbolically”

$$\exists A^0. (\iota^0 \wedge \mathcal{R}_1(A^0, A^1))$$

projection to time 1: $\bigcup_{o \in O} \text{img}_o(\iota)$
 This is the set of states that are reachable from ι by executing one operator.



Formula for parallel actions

Definition

Define $\mathcal{R}_2(A, A', O)$ as

$$\tau_A(O) \wedge \bigwedge_{o \in O, o' \in O} \text{interfere} \quad \neg(o \wedge o') \quad (+\text{invariants})$$

Planning as satisfiability

Example

	0 1 2 3 4 5	0 1 2 3 4 5	0 1 2 3 4 5
clear(a)	FF	FFF TT	FFFTT
clear(b)	F	F FTTF	FFTTTF
clear(c)	TT	FTTTF	TTTTFF
clear(d)	FTTF	FTTFFF	FTTFFF
clear(e)	TTF	TTF	TTF
on(a,b)	FFF T	FFFFT	FFFFT
on(a,c)	FFFFFF	FFFFFF	FFFFFF
on(a,d)	FFFFFF	FFFFFF	FFFFFF
on(a,e)	FFFFFF	FFFFFF	FFFFFF
on(b,a)	TT	FTT FF	TTTTFF
on(b,b)	FF	FTT	FFFTT
on(b,d)	FFFFFF	FFFFFF	FFFFFF
on(b,e)	FFFFFF	FFFFFF	FFFFFF
on(c,a)	FFFFFF	FFFFFF	FFFFFF
on(c,b)	T	FFF TTFFF	TTFFF
on(c,d)	FFFTT	FFFTT	FFFTT
on(c,e)	FFFFFF	FFFFFF	FFFFFF
on(d,a)	FFFFFF	FFFFFF	FFFFFF
on(d,b)	FFFFFF	FFFFFF	FFFFFF
on(d,c)	FFFFFF	FFFFFF	FFFFFF
on(d,e)	FFTTT	FFTTT	FFTTT
on(e,a)	FFFFFF	FFFFFF	FFFFFF
on(e,b)	FFFFFF	FFFFFF	FFFFFF
on(e,c)	FFFFFF	FFFFFF	FFFFFF
on(e,d)	T	TTTT	TTTT
ontable(a)	TTT	TTTT	TTTT
ontable(b)	F	FF	FF
ontable(c)	F	FF	FF
ontable(d)	TTTT	TTTT	TTTT
ontable(e)	TTTTT	TTTTT	TTTTT

- State variable values inferred from **initial values** and **goals**.
- Branch: $\neg \text{clear}(b)^1$.
- Branch: $\text{clear}(a)^3$.
- Plan found:

	0 1 2 3 4
fromtable(a,b)	FFFTT
fromtable(b,c)	FFFTF
fromtable(c,d)	FFTF
fromtable(d,e)	FTFFF
totable(b,a)	FFTF
totable(c,b)	FTFFF
totable(e,d)	TTTT

Applications of Planning as Satisfiability

Testing the **existence of a path in a transition system** (a path that satisfies certain properties) is a basic problem that shows up everywhere.

- Classical planning (Kautz & Selman, 1992/1996)
- As a **subprocedure** of algorithms for **conditional planning**: Testin the validity of a candidate **conditional plan** (Rintanen 1999, Castellini et al. 2003).
 Applicable to the general conditional planning problem with partial observability.
- Bounded model-checking** (Biere et al. 1999)
 “Testing the validity of a given plan” \sim “model-checking”

Existential and universal abstraction

Definition

Existential abstraction of a formula ϕ with respect to $a \in A$:

$$\exists a. \phi = \phi[T/a] \vee \phi[\perp/a].$$

Universal abstraction is defined analogously by using conjunction instead of disjunction.

Definition

Universal abstraction of a formula ϕ with respect to $a \in A$:

$$\forall a. \phi = \phi[T/a] \wedge \phi[\perp/a].$$

\exists -abstraction

Examples

Example

$$\begin{aligned} & \exists b.((a \rightarrow b) \wedge (b \rightarrow c)) \\ &= ((a \rightarrow \top) \wedge (\top \rightarrow c)) \vee ((a \rightarrow \perp) \wedge (\perp \rightarrow c)) \\ &\equiv c \vee \neg a \\ &\equiv a \rightarrow c \end{aligned}$$

$$\begin{aligned} \exists ab.(a \vee b) &= \exists b.(\top \vee b) \vee (\perp \vee b) \\ &= ((\top \vee \top) \vee (\perp \vee \top)) \vee ((\top \vee \perp) \vee (\perp \vee \perp)) \\ &\equiv (\top \vee \top) \vee (\top \vee \perp) \equiv \top \end{aligned}$$

Example

\exists -abstraction is also known as **forgetting**:
 $\exists \text{mon} \exists \text{tue}((\text{mon} \vee \text{tue}) \wedge (\text{mon} \rightarrow \text{work}) \wedge (\text{tue} \rightarrow \text{work}))$
 $\equiv \exists \text{tue}((\text{work} \wedge (\text{tue} \rightarrow \text{work})) \vee (\text{tue} \wedge (\text{tue} \rightarrow \text{work}))) \equiv \text{work}$

Images by \exists -abstraction

Let

- $A = \{a_1, \dots, a_n\}$,
- $A' = \{a'_1, \dots, a'_n\}$,
- ϕ be a formula over A that represents a set T of states, and
- $\tau_A(o)$ the formula over $A \cup A'$ that represents the operator o (a binary relation on states).

The **image** $\{s' \in S \mid s \in T, sos'\}$ of T with respect to o is represented by

$$\exists A.(\phi \wedge \tau_A(o))$$

which is a formula over A' .

To obtain a formula over A we have to rename the variables.

$$\text{img}_o(\phi) = (\exists A.(\phi \wedge \tau_A(o)))[A/A']$$

Preimages by \exists -abstraction

Let

- $A = \{a_1, \dots, a_n\}$,
- $A' = \{a'_1, \dots, a'_n\}$,
- ϕ be a formula over A that represents a set T of states, and
- $\tau_A(o)$ the formula over $A \cup A'$ that represents an operator o (a binary relation on states).

The **preimage** $\{s \in S \mid sos', s' \in T\}$ of T with respect to o is represented by

$$\text{preimg}_o(\phi) = \exists A'.(\phi[A'/A] \wedge \tau_A(o))$$

which is a formula over A .

Preimages vs. regression

Preimages vs. regression

For deterministic operators o

$$\text{regr}_o(\phi) \equiv \text{preimg}_o(\phi).$$

\exists -abstraction in terms of truth-tables

Example

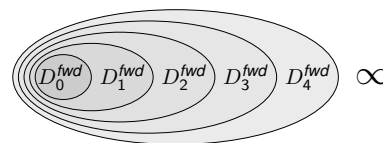
$\forall c$ and $\exists c$ correspond to **combining pairs of lines** with the same valuation for variables other than c .

Example

$$\exists c.(a \vee (b \wedge c)) \equiv a \vee b \quad \forall c.(a \vee (b \wedge c)) \equiv a$$

a	b	c	$a \vee (b \wedge c)$	$a \vee b$	$\exists c.(a \vee (b \wedge c))$	$a \vee b$	$\forall c.(a \vee (b \wedge c))$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	1	1	1	1
0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1
1	0	1	1	1	1	1	1
1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1

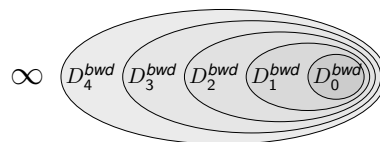
Forward distances with formulae



Forward distances with formulae

$$\begin{aligned} D_0^{fwd} &= \bigwedge \{a \in A \mid I \models a\} \wedge \bigwedge \{\neg a \mid a \in A, I \not\models a\} \\ D_i^{fwd} &= D_{i-1}^{fwd} \vee \bigvee_{o \in O} \text{img}_o(D_{i-1}^{fwd}) \text{ for all } i \geq 1 \end{aligned}$$

Backward distances with formulae



Backward distances with formulae

$$\begin{aligned} D_0^{bwd} &= G \\ D_i^{bwd} &= D_{i-1}^{bwd} \vee \bigvee_{o \in O} \text{preimg}_o(D_{i-1}^{bwd}) \text{ for all } i \geq 1 \end{aligned}$$

Nondeterministic operators

Nondeterministic operators

A nondeterministic operator $o = \langle c, e_1 \mid \dots \mid e_n \rangle$ assigns a state s the successor states $\text{app}_{\langle c, e_1 \rangle}(s), \dots, \text{app}_{\langle c, e_n \rangle}(s)$.

The application of o in s is defined if the application of all of $\langle c, e_1 \rangle, \dots, \langle c, e_n \rangle$ is defined in s .

Translation into the propositional logic

For $o = \langle c, e_1 \mid \dots \mid e_n \rangle$ define

$$\tau_A^{nd}(o) = \tau_A(\langle c, e_1 \rangle) \vee \dots \vee \tau_A(\langle c, e_n \rangle)$$

Regression for nondeterministic operators

Definition

The regression operation for deterministic operators can be generalized to **regression for nondeterministic operators**.

Definition (Regression for nondeterministic operators)

Let ϕ be a propositional formula and $o = \langle c, e_1 | \dots | e_n \rangle$ a nondeterministic operator. Define

$$\text{regr}_o^{\text{nd}}(\phi) = \text{regr}_{\langle c, e_1 \rangle}(\phi) \wedge \dots \wedge \text{regr}_{\langle c, e_n \rangle}(\phi).$$

(Albert-Ludwigs-Universität Freiburg)

106

Nondeterminism Regression

Regression for nondeterministic operators

Example

Example

Let $o = \langle d, b | \neg c \rangle$. Then

$$\begin{aligned} \text{regr}_o^{\text{nd}}(b \leftrightarrow c) &= \text{regr}_{\langle d, b \rangle}(b \leftrightarrow c) \wedge \text{regr}_{\langle d, \neg c \rangle}(b \leftrightarrow c) \\ &= (d \wedge (T \leftrightarrow c)) \wedge (d \wedge (b \leftrightarrow \perp)) \\ &\equiv d \wedge c \wedge \neg b. \end{aligned}$$

(Albert-Ludwigs-Universität Freiburg)

107

Nondeterminism Strong preimages

Strong preimages by \exists/\forall -abstraction

Let

- ▶ $A = \{a_1, \dots, a_n\}$,
- ▶ $A' = \{a'_1, \dots, a'_n\}$,
- ▶ ϕ be a formula over A that represents a set T of states, and
- ▶ $\tau_A(o)$ the formula over $A \cup A'$ that represents an operator o (a binary relation on states).

The **strong preimage** $\{s \in S \mid \exists s', s' \in T, \text{img}_o(s) \subseteq T\}$ of T with respect to o is represented by

$$\text{spreimg}_o(\phi) = \forall A'. (\tau_A^{\text{nd}}(o) \rightarrow \phi[A'/A]) \wedge \exists A'. \tau_A^{\text{nd}}(o)$$

which is a formula over A .

(Albert-Ludwigs-Universität Freiburg)

108

Nondeterminism Backward distances

Backward distances with formulas

Nondeterministic operators

With regression (equivalently, with strong preimages) we can compute formulas that represent backward distance sets for nondeterministic operators.

Definition

Let G be a formula and O a set of operators. The **backward distance sets** D_i^{bwd} for G, O are represented by the following formulae.

$$\begin{aligned} D_0^{\text{bwd}} &= G \\ D_i^{\text{bwd}} &= D_{i-1}^{\text{bwd}} \vee \bigvee_{o \in O} \text{regr}_o^{\text{nd}}(D_{i-1}^{\text{bwd}}) \text{ for all } i \geq 1 \end{aligned}$$

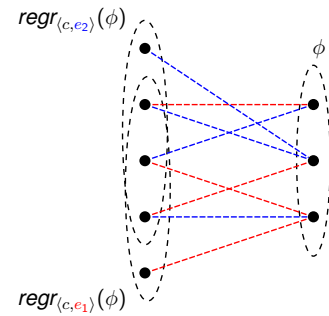
(Albert-Ludwigs-Universität Freiburg)

111

Regression for nondeterministic operators

Illustration

$$\text{regr}_{\langle c, e_1 | e_2 \rangle}(\phi) = \text{regr}_{\langle c, e_1 \rangle}(\phi) \wedge \text{regr}_{\langle c, e_2 \rangle}(\phi)$$



(Albert-Ludwigs-Universität Freiburg)

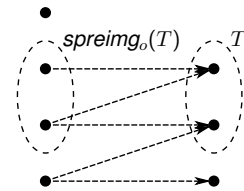
106

Nondeterminism Strong preimages

Strong preimages

Strong preimage

The **strong preimage** of a set T of states with respect to an operator is the set of those states from which a state in T is always reached when o is executed.



(Albert-Ludwigs-Universität Freiburg)

106

Nondeterminism vs. Regression

Strong preimages vs. regression

Strong preimages vs. regression

For nondeterministic operators o

$$\text{regr}_o^{\text{nd}}(\phi) \equiv \text{spreimg}_o(\phi).$$

(Albert-Ludwigs-Universität Freiburg)

110

Nondeterminism Complexity

Complexity issues

Complexity of the symbolic set operations

operation	complexity	
\wedge	intersection	polynomial size
\vee	union	polynomial size
$\exists A$	for relational projection	exponential in $ A $
\equiv	for the fixpoint condition	co-NP-hard

(Albert-Ludwigs-Universität Freiburg)

111

(Albert-Ludwigs-Universität Freiburg)

112

Shannon expansion

Definition

3-place connective **if-then-else** is defined by

$$\text{ite}(a, \phi_1, \phi_2) = (a \wedge \phi_1) \vee (\neg a \wedge \phi_2)$$

where a is a propositional variable.

Definition

Shannon expansion of a formula ϕ with respect to $a \in A$ is

$$\phi \equiv (a \wedge \phi[\top/a]) \vee (\neg a \wedge \phi[\perp/a]) = \text{ite}(a, \phi[\top/a], \phi[\perp/a])$$

Binary decision diagrams

Canonicity

Transformation to ordered BDDs

1. Fix an ordering a_1, \dots, a_n on all propositional variables.
2. Apply Shannon expansion to all variables in this order.
3. Represent the resulting formulae as directed acyclic graphs (DAG) so that shared subformulae occur only once.

Theorem

Let ϕ_1 and ϕ_2 be two ordered BDDs obtained by using the same variable ordering. Then $\phi_1 \equiv \phi_2$ if and only if ϕ_1 and ϕ_2 are isomorph. (the same DAG.)

Properties of normal forms

Trade-offs between different normal forms

Normal forms that allow faster reasoning are more expensive to construct from an arbitrary propositional formula and may be much bigger.

Properties of different normal forms

	\vee	\wedge	\neg	$\phi \in \text{TAUT?}$	$\phi \in \text{SAT?}$	$\phi \equiv \phi'$
circuits	poly	poly	poly	co-NP-hard	NP-hard	co-NP-hard
formulae	poly	poly	poly	co-NP-hard	NP-hard	co-NP-hard
DNF	poly	exp	exp	co-NP-hard	in P	co-NP-hard
CNF	exp	poly	exp	in P	NP-hard	co-NP-hard
BDD	exp	exp	poly	in P	in P	in P

For BDDs one \vee/\wedge is polynomial time/size (size is doubled) but repeated \vee lead to exponential size.

Satisfiability algorithms vs. BDDs

Comparison: formula size, runtime

technique	size of $\mathcal{R}_1(P, P')$	runtime for plan length n
satisfiability	not a problem	exponential in n
BDDs	major problem	often close to linear in n

Comparison: application domain

technique	types of problems
satisfiability	lots of state variables, short plans
BDDs	few state variables, long plans

Binary decision diagrams

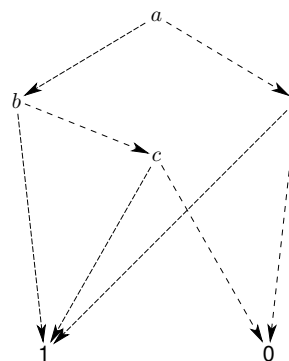
Example

By repeated application of Shannon expansion any propositional formula can be transformed to an equivalent formula containing no other connectives than *ite* and propositional variables only in the first position of *ite*.

Example

$$\begin{aligned} & (a \vee b) \wedge (b \vee c) \\ \equiv & \text{ite}(a, (\top \vee b) \wedge (b \vee c), (\perp \vee b) \wedge (b \vee c)) \\ \equiv & \text{ite}(a, b \vee c, b) \\ \equiv & \text{ite}(a, \text{ite}(b, \top \vee c, \perp \vee c), \text{ite}(b, \top, \perp)) \\ \equiv & \text{ite}(a, \text{ite}(b, \top, c), \text{ite}(b, \top, \perp)) \\ \equiv & \text{ite}(a, \text{ite}(b, \top, \text{ite}(c, \top, \perp)), \text{ite}(b, \top, \perp)) \end{aligned}$$

Binary decision diagrams: $(a \vee b) \wedge (b \vee c)$



Applications of BDDs

The BDD-based techniques are very general and powerful, and hence applicable to many types of problems.

- ▶ Classical planning: BDDs rarely competitive with other techniques (too general).
- ▶ Conditional planning with full observability: Implementation technique; traversal of a high number of states simultaneously.
- ▶ Conditional planning with partial observability: each **belief state** naturally represented as one formula/BDD.