

State-Space Traversal Techniques for AI Planning

Jussi Rintanen

National ICT Australia, Canberra

July 16, 2006

Introduction

1 / 105

Related research topics

Overlapping

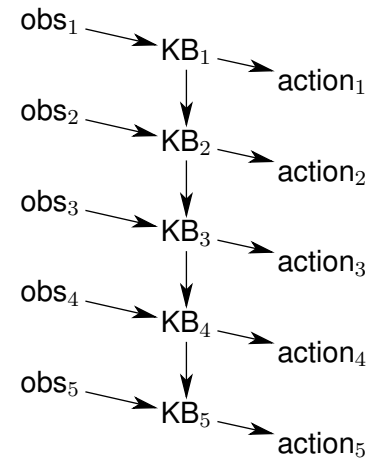
- ▶ Program synthesis
- ▶ Model-Checking and Reachability analysis in Computer-Aided Verification
- ▶ controller synthesis for Discrete-Event Systems (DES)

Orthogonal

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

3 / 105

What is planning?



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

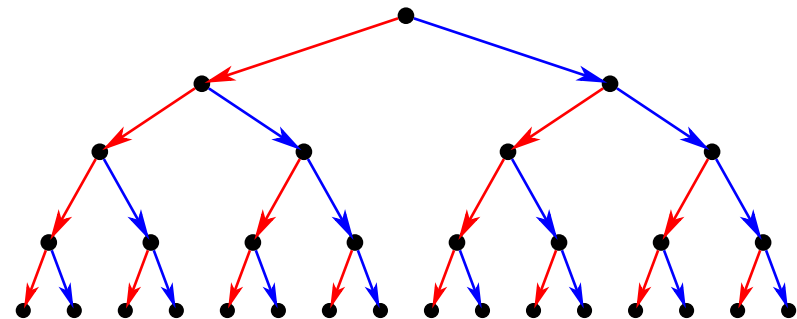
- ▶ knowledge base (KB) about the world
- ▶ general-purpose problem representation (PDDL, logic, ...)
- ▶ algorithms for solving any problem expressible in the representation

Introduction

2 / 105

Search trees for classical planning

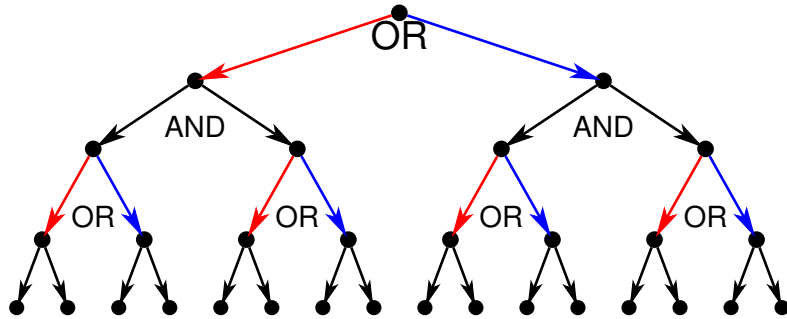
Prototypical search algorithms: depth/breadth first, A*



4 / 105

Search tree for conditional planning

Prototypical solution technique: AND-OR search, AO*



Contents of the tutorial

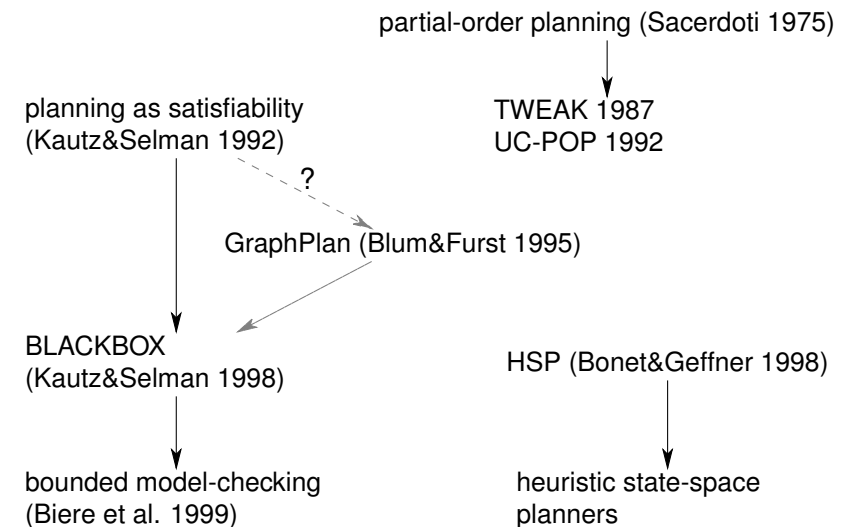
1. Framework for representing planning problems
2. Approximations of **reachability**: basis of
 - ▶ distance heuristics and planning by heuristic search
 - ▶ techniques for simplifying problem instances
 - ▶ pruning techniques for partial representations (SAT/CSP)
3. Solution techniques:
 - ▶ heuristic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

Why is planning difficult?

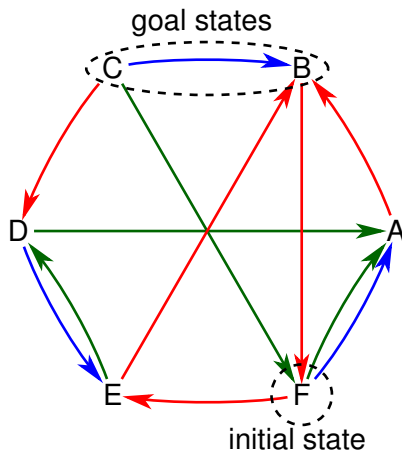
- ▶ Solving classical planning by exhaustively enumerating the state space is feasible for up to 10^6 or 10^7 states and the most efficient technique if there are few states ($< 10^5$).
- ▶ Otherwise more clever techniques are needed.
 1. heuristic search (A*, ...)
 2. "symbolic" representations of sets of states
- ▶ Planning with observability restrictions, several agents (game-theoretic planning) do not directly reduce to state space traversal.

Historical timeline

Main approaches to deterministic/classical planning



Transition systems



Succinct representation of transition systems

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

Example

HOUR : {0, ..., 23} = 13

MINUTE : {0, ..., 59} = 55

LOCATION : { 51, 52, 82, 101, 102 } = 101

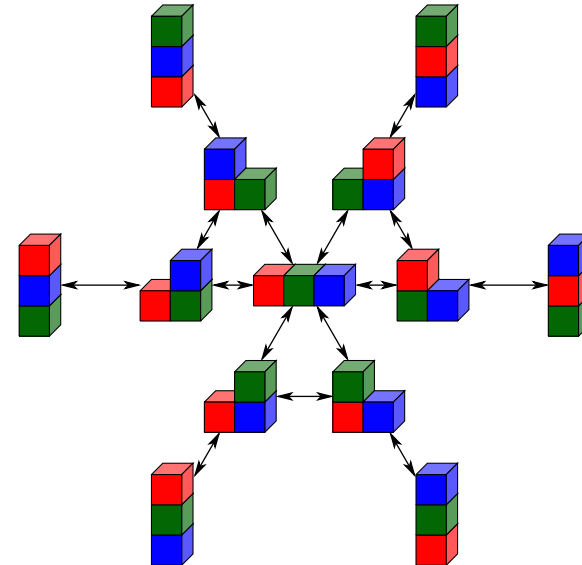
WEATHER : { sunny, cloudy, rainy } = cloudy

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.

Blocks world

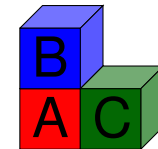
The transition graph for three blocks



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. s' such that $s'(AonB) = 1$ and $s'(BonA) = 1$.

Representation of sets as formulas

state sets	formulas over A
those $\frac{2^{ A }}{2}$ states where a is true	$a \in A$
\overline{E} (complement)	$\neg E$
$E \cup F$	$E \vee F$
$E \cap F$	$E \wedge F$
$E \setminus F$ (set difference)	$E \wedge \neg F$
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 $F \models E?$

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.

STRIPS operators

have the form $\langle a_1 \wedge \dots \wedge a_n, l_1 \wedge \dots \wedge l_m \rangle$ where $\{a_1, \dots, a_n\} \subseteq A$ and l_1, \dots, l_m are literals.

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 do the arrows go.**)
The description is procedural: how do the values of the state variables change?

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 of 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 basic PDDL that our operators do not have are related to instantiating the schema variables: typing, \exists and \forall quantification.

Operators

Example

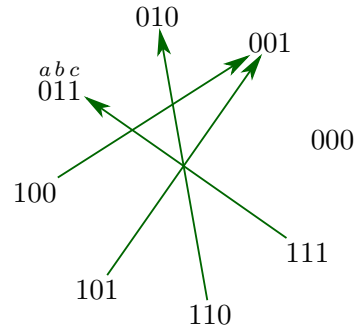
State variables are $A = \{a, b, c\}$.

The operator

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

corresponds to the binary relation on the right.

Clearly, one operator could represent an **arbitrarily high number of arrows!**



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

Operators

Semantics

Changes caused by an operator

Associate with each effect e and state s a set $[e]_s$ of literals:

1. $[a]_s = \{a\}$ for $a \in A$.
2. $[\neg a]_s = \{\neg a\}$ for $a \in A$.
3. $[e_1 \wedge \dots \wedge e_n]_s = [e_1]_s \cup \dots \cup [e_n]_s$.
4. $[c \triangleright e]_s = [e]_s$ 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.

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**.

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

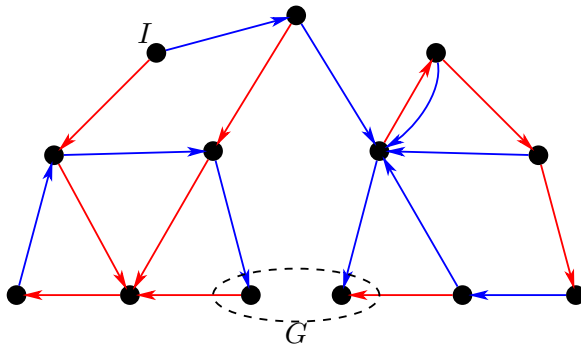
1. $s_0 = I$,
2. $s_i = \text{app}_{o_i}(s_{i-1})$ for every $i \in \{1, \dots, n\}$, and
3. $s_n \models G$.

This can be equivalently expressed as

$$\text{app}_{o_n}(\text{app}_{o_{n-1}}(\dots \text{app}_{o_1}(I) \dots)) \models G.$$

Planning by forward search

with depth-first search



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*, best first, ...; **local**: hill-climbing, simulated annealing, ...), ...
3. different ways of controlling search:
 - 3.1 **different heuristics** for heuristic search algorithms
 - 3.2 **pruning techniques**: invariants, symmetry elimination, ...

Progression

- ▶ **Progression** means computing the successor state $\text{app}_o(s)$ 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.

Regression

- ▶ **Regression** = computation of predecessors of states
- + Advantage over progression: a formula represents a **set of states**.
- 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.

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 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, does not work.

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

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

Regressing a state variable

The formula $EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))$ expresses the value of $a \in A$ **after** executing e in terms of values of state variables **before** executing e : a will be true if 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

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

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

1. $\text{regr}_{\langle a, b \rangle}(b) = (a \wedge (\top \vee (b \wedge \neg \perp))) \equiv a$
2. $\text{regr}_{\langle a, b \rangle}(b \wedge c) = (a \wedge (\top \vee (b \wedge \neg \perp)) \wedge (\perp \vee (c \wedge \neg \perp))) \equiv a \wedge c$
3. $\text{regr}_{\langle a, b \rangle}(b \vee c) = (a \wedge ((\top \vee (b \wedge \neg \perp)) \vee (\perp \vee (c \wedge \neg \perp)))) \equiv a$
4. $\text{regr}_{\langle a, c \triangleright b \rangle}(b) = (a \wedge (c \vee (b \wedge \neg \perp))) \equiv a \wedge (c \vee b)$

Regression

Properties

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

$$\{s \in S \mid s \models \text{regr}_o(\phi)\} = \{s \in S \mid \text{app}_o(s) \models \phi\}.$$

Regression

Generation of search trees

Problem Regression may produce very big formulas.

Cause **Disjunctivity** in the formulas. Formulas **without disjunctions** easily convertible to formulas $l_1 \wedge \dots \wedge l_n$ where l_i are literals and $n \leq |A|$.

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

Complexity issues

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 $\mathcal{O}(2^n)$.

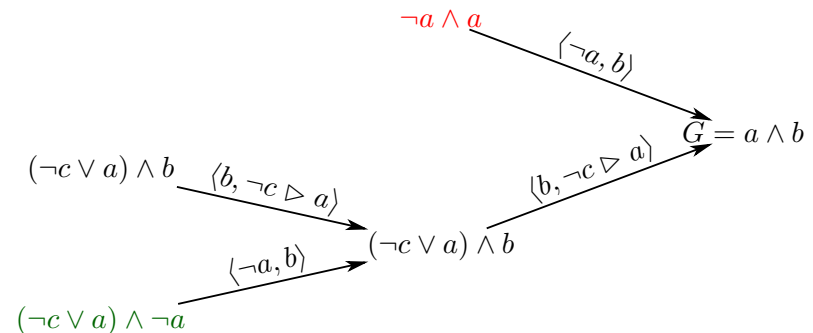
Simplifications

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

Regression: generation of search trees

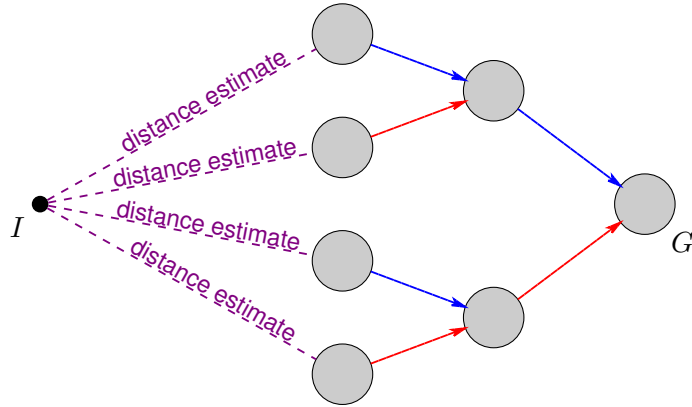
Unrestricted regression (= do nothing about formula size)

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



Planning by heuristic search

Backward search



Images

Images

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

$$img_o(s) = \{s' | sos'\}.$$

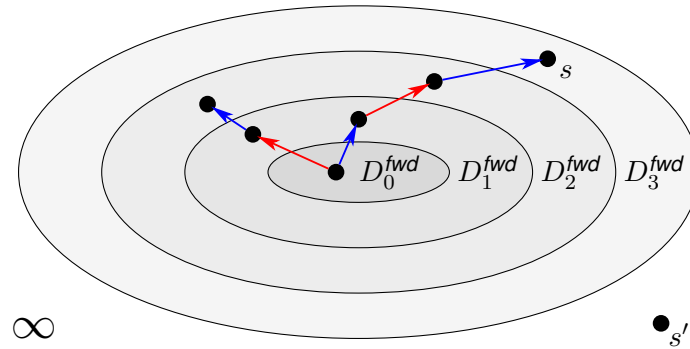
This can be generalized to sets T of states:

$$img_o(T) = \bigcup_{s \in T} img_o(s)$$

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 ∞ .

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

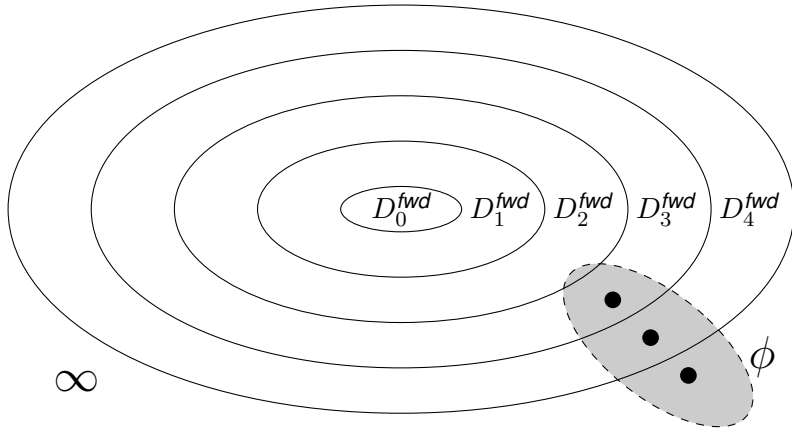
$$D_0^{fwd} = \{I\}$$

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

Distances

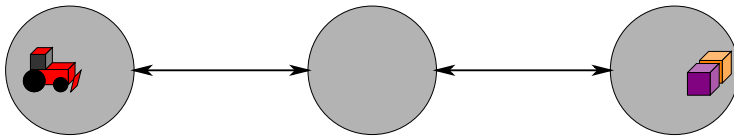
of formulas

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



Distance estimation with max-distances

Tractor example



1. Tractor moves:

- ▶ from 1 to 2: $T_{12} = \langle T_1, T_2 \wedge \neg T_1 \rangle$
- ▶ from 2 to 1: $T_{21} = \langle T_2, T_1 \wedge \neg T_2 \rangle$
- ▶ from 2 to 3: $T_{23} = \langle T_2, T_3 \wedge \neg T_2 \rangle$
- ▶ from 3 to 2: $T_{32} = \langle T_3, T_2 \wedge \neg T_3 \rangle$

2. Tractor pushes A:

- ▶ from 2 to 1: $A_{21} = \langle T_2 \wedge A_2, T_1 \wedge A_1 \wedge \neg T_2 \wedge \neg A_2 \rangle$
- ▶ from 3 to 2: $A_{32} = \langle T_3 \wedge A_3, T_2 \wedge A_2 \wedge \neg T_3 \wedge \neg A_3 \rangle$

3. Tractor pushes B:

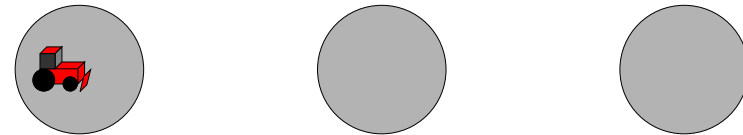
- ▶ from 2 to 1: $B_{21} = \langle T_2 \wedge B_2, T_1 \wedge B_1 \wedge \neg T_2 \wedge \neg B_2 \rangle$
- ▶ from 3 to 2: $B_{32} = \langle T_3 \wedge B_3, T_2 \wedge B_2 \wedge \neg T_3 \wedge \neg B_3 \rangle$

Distance estimation with max-distances

- ▶ Computation of exact distances is as hard as planning itself: only their approximations are useful as heuristics.
- ▶ 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.

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 $A_1 \wedge B_1$ is 4.

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**.

$$\begin{aligned}
 D_0^{max} &= \{T1, \neg T2, \neg T3, \neg A1, \neg A2, A3, \neg B1, \neg B2, B3\} \\
 D_1^{max} &= \{ \quad \quad \quad \neg T3, \neg A1, \neg A2, A3, \neg B1, \neg B2, B3\} \\
 D_2^{max} &= \{ \quad \quad \quad \neg A1, \neg A2, A3, \neg B1, \neg B2, B3\} \\
 D_3^{max} &= \{ \quad \quad \quad \neg A1, \quad \quad \quad \neg B1 \quad \quad \quad \} \\
 D_4^{max} &= \{ \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \}
 \end{aligned}$$

The sets D_i^{max} approximate **forward distance sets**:
 $D_i^{fwd} \subseteq \{s \in S \mid s \models D_i^{max}\}$ for all $i \geq 0$.

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 $\delta_s^{max}(G)$.

Distances of literals

EPC for operator application

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

Computation of literal sets for max-distances

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

$$\begin{aligned}
 D_0^{max} &= \{l \in L \mid I \models l\} \\
 D_i^{max} &= D_{i-1}^{max} \setminus \{l \in L \mid o \in O, SAT(D_{i-1}^{max} \cup \{EPC_l(o)\})\}
 \end{aligned}$$

To make the computation polynomial time we must use a poly-time approximation of $SAT(\cdot)$.

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 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

Sum-distance of $T2 \wedge A2$ is 1+3.

Sum-distance of $A1$ is 1+3+1 = 5 (max-distance is 4.)

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

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 & Geffner 2000) and the sum-heuristic (Haslum, Bonet & Geffner 2005).

Invariants

Motivation

Example

Consider the goal formula

$$A2 \wedge T1$$

regressed with the operator

$$\langle T3 \wedge A3, T2 \wedge A2 \wedge \neg T3 \wedge \neg A3 \rangle$$

giving the new goal

$$T3 \wedge A3 \wedge T1.$$

No legal state satisfies this formula.

Invariants

(Planning graphs)

Goal: Restriction to states that are reachable.

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

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

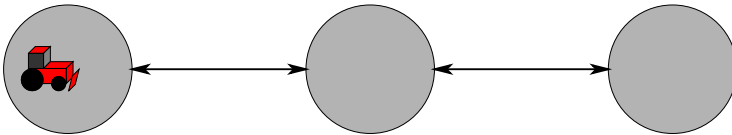
Implementation: Compute in polynomial time **formulas** that characterize a **superset** of the reachable states. Computation of C_0, C_1, C_2, \dots generalizes the computation of $D_0^{max}, D_1^{max}, D_2^{max}, \dots$. Formulas in $\bigcap_{i \geq 0} C_i$ are **invariants**.

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



$$\begin{aligned}
 n &= 2 \text{ (maximum clause length)} \\
 C_0 &= \{T1, \neg T2, \neg T3\} \sim \{100\} \\
 C_1 &= \{T1 \vee T2, \neg T1 \vee \neg T2, \neg T3\} \sim \{010, 100\} \\
 C_2 &= \{\neg T1 \vee \neg T2, \neg T1 \vee \neg T3, \neg T2 \vee \neg T3\} \\
 &\quad \sim \{000, 001, 010, 100\} \\
 C_i &= C_2 \text{ for all } i \geq 3
 \end{aligned}$$

Summary

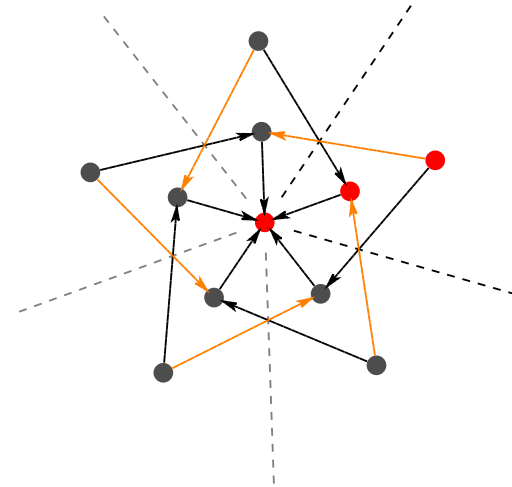
- ▶ Distance heuristics:
 - ▶ Approximations of the **distances** between states.
 - ▶ **max-heuristic**: admissible, not very informative
 - ▶ **sum-heuristic**: more informative, may over and underestimate
 - ▶ **relaxed plans**: more informative, may over and underestimate
- ▶ Invariants:
 - ▶ more exact characterization of reachability
 - ▶ application: pruning search space for backward search & planning as satisfiability
 - ▶ Efficient polynomial time algorithms exist.
 - ▶ **Planning graphs** include 2-literal invariants.

Pruning techniques for state-space search

- ▶ 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 in 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

Example



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.

Planning in the propositional logic

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

Existence of plans of length t

Atomic propositions

Define $A^i = \{a^i | a \in A\}$ for all $i \in \{0, \dots, t\}$.
 a^i expresses the value of $a \in A$ at time i .

Plans of length t in the propositional logic

Plans of length t correspond to satisfying valuations of

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

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

Translating operators into formulas

Operators as formulas

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

$$\begin{aligned} \tau_A(o) &= \bigwedge_{a \in A} (EPC_a(e) \vee (a \wedge \neg EPC_{\neg a}(e))) \leftrightarrow a^c & (1) \\ &\bigwedge_{a \in A} \neg (EPC_a(e) \wedge EPC_{\neg a}(e)) & (3) \end{aligned}$$

(2) says the **new value of a** is 1 if it was assigned 1 **or** the the old value 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**.

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 as **pairs of valuations** $(10, 10)$ and $(11, 11)$ of A .

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 logic

Plans of length t correspond to satisfying valuations of

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

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

Planning as satisfiability

Example

Example

Consider

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

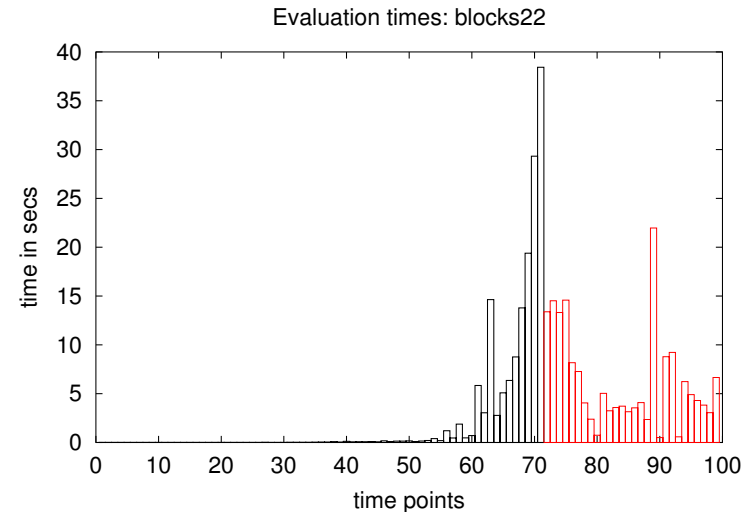
Formula for plans of length 3 is

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

Parallel evaluation strategies

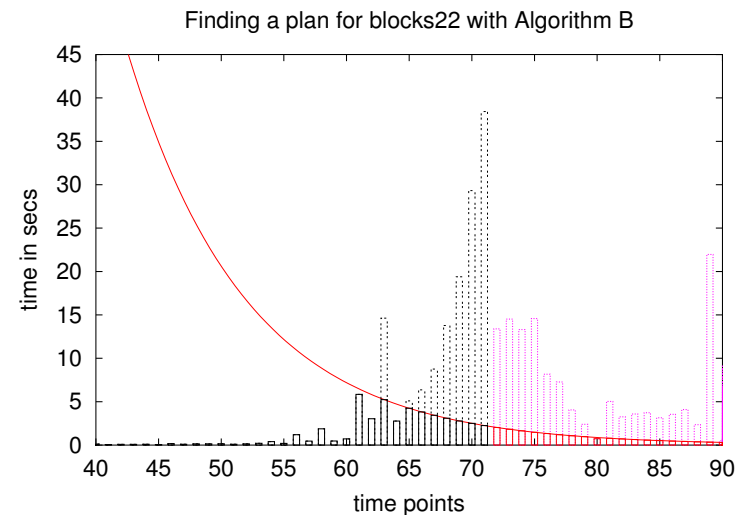
- ▶ The plan search strategy that performs satisfiability tests of formulas for lengths 0,1,2,... until a satisfiable formula (plan) is found always **proves that the plan** has optimal length.
- ▶ Often **proving optimality is much more difficult than finding a plan**.
- ▶ This is the reason why planning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

Profile of runtimes for different plan lengths



A parallel evaluation strategy

Plan found



Planning as satisfiability with parallel plans

- ▶ **Efficiency** of satisfiability planning is strongly **dependent on the plan length** because satisfiability algorithms have worst-case exponential runtime in 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.

Interpretation of parallelism

Example

Operators $\langle a, \neg b \rangle$ and $\langle b, \neg a \rangle$ have non-contradictory effects and preconditions.

However, neither operator sequence

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.

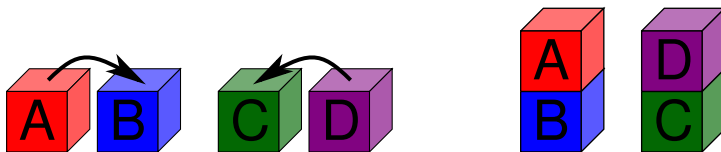
Standard interpretation of parallelism

Operators are **executable in every order**.

Interference

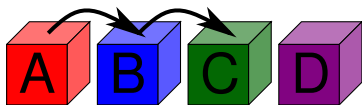
Example

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.

Interference

Interference

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

1. o may make the precondition of o' false, or
2. 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.

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 because plans have less time points.

Planning as satisfiability

Example

	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5
clear(a)	F	F					F	F	T	T			F	F	T	T		
clear(b)	F		F				F	F	T	T	F		F	F	T	T	F	
clear(c)	T	T		F	F		T	T	T	T	F	F	T	T	T	T	F	F
clear(d)	F	T	T	F	F	F	F	T	T	F	F	F	F	T	T	F	F	F
clear(e)	T	T	F	F	F	F	T	T	F	F	F	F	T	T	F	F	F	F
on(a,b)	F	F	F	T			F	F	F	F	T		F	F	F	F	T	
on(a,c)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
on(a,d)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
on(a,e)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
on(b,a)	T	T		F	F		T	T	T	F	F		T	T	T	F	F	
on(b,c)	F	F	T				F	F	F	T			F	F	F	T		
on(b,d)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
on(b,e)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
on(c,a)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
on(c,b)	T		F	F			T	T	F	F			T	T	F	F		
on(c,d)	F	F	F	T	T		F	F	F	T	T		F	F	F	T	T	
on(c,e)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
on(d,a)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
on(d,b)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
on(d,c)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
on(d,e)	F	F	T	T	T		F	F	T	T	T		F	F	T	T	T	
on(e,a)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
on(e,b)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
on(e,c)	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
on(e,d)	T	F	F	F	F	F	T	F	F	F	F	F	T	F	F	F	F	F
ontable(a)	T	T	T		F		T	T	T	T	F		T	T	T	T		F
ontable(b)	F	F		F	F		F	F	F	F			F	F	F	F		F
ontable(c)	F		F	F	F		F	F	F	F			F	F	F	F		F
ontable(d)	T	T	F	F	F	F	T	T	F	F	F	F	T	T	F	F	F	F
ontable(e)	F	T	T	T	T	T	F	T	T	T	T	T	F	T	T	T	T	T

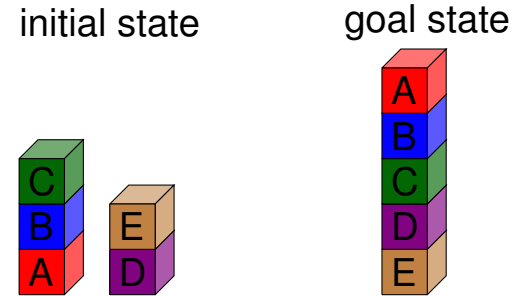
1. State variable values inferred from **initial values** and **goals**.
2. Branch: $\neg \text{clear}(b)^1$.
3. Branch: $\text{clear}(a)^3$.
4. Plan found:


```

0 1 2 3 4
fromtable(a,b) FFFFT
fromtable(b,c) FFFTF
fromtable(c,d) FFFTF
fromtable(d,e) FFFFF
totable(b,a) FFFTF
totable(c,b) FFFTF
totable(e,d) FFFFF
            
```

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.

Applications of Planning as Satisfiability

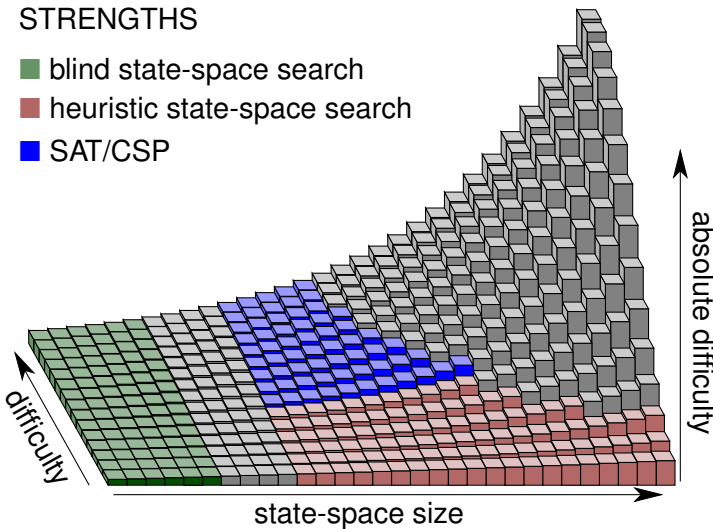
Testing the **existence of a path in a transition system** (a path that satisfies certain properties) is a basic problem that shows up everywhere.

- ▶ Classical planning (Kautz & Selman, 1992/1996)
- ▶ As a **subprocedure** of algorithms for **conditional planning**: Testing 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” ~ “model-checking”

Relative strengths of different approaches

- ▶ “(Heuristic) state-space search stronger when optimality of plan is not required.”
- ▶ “Planning with SAT/CSP is stronger when optimality is required.”
- ▶ State space search is often bad at escaping **local minima**. State space search considers every state explicitly, and cannot group together sets of (similar) states.
- ▶ But:
 - ▶ There are big/difficult local minima in difficult problem instances irrespective of the optimality question.
 - ▶ For many of the standard benchmarks, planners have a problem escaping local minima only when looking for optimal plans.

Relative strengths of different approaches



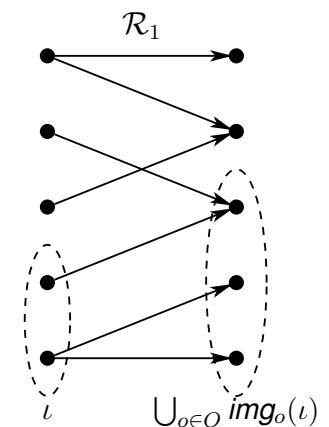
“Symbolic” representation of sets of states

- ▶ Represent **sets** of states without representing every state explicitly.
- ▶ Represent **binary relations** on states without representing every pair of states explicitly.
- ▶ Compute set and relation operations.
- ▶ Implement planning algorithms which do not represent every state explicitly.

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.



Existential and universal abstraction

Definition

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

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

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

Definition

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

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

\forall and \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

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

\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**:

$$\begin{aligned} & \exists \text{mon} \exists \text{tue}((\text{mon} \vee \text{tue}) \wedge (\text{mon} \rightarrow \text{work}) \wedge (\text{tue} \rightarrow \text{work})) \\ &\equiv \exists \text{tue}((\text{work} \wedge (\text{tue} \rightarrow \text{work})) \vee (\text{tue} \wedge (\text{tue} \rightarrow \text{work}))) \equiv \text{work} \end{aligned}$$

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

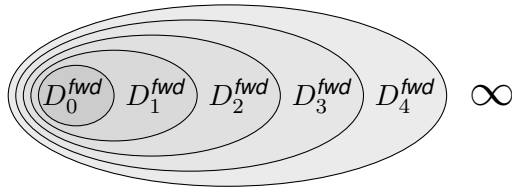
$$\exists A.(\phi \wedge \tau_A(o))$$

which is a formula over A' .

To obtain a formula over A we rename the variables.

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

Forward distances with formulae

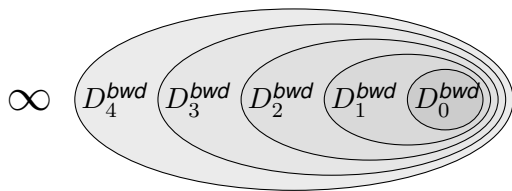


Forward distances with formulae

$$D_0^{fwd} = \bigwedge \{a \in A \mid I \models a\} \wedge \bigwedge \{\neg a \mid a \in A, I \not\models a\}$$

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

Backward distances with formulae



Backward distances with formulae

$$D_0^{bwd} = G$$

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

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

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

which is a formula over A .

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

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

Regression for nondeterministic operators

Example

Example

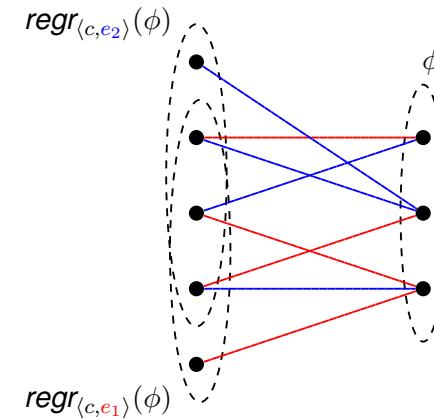
Let $o = \langle d, b | \neg c \rangle$. Then

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

Regression for nondeterministic operators

Illustration

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



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.

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

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 isomorphic (the same DAG.)

Binary decision diagrams

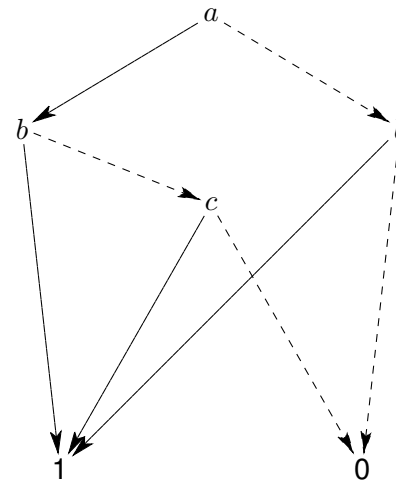
Example

Shannon expansion can be used for reducing any formula to one with the connective *ite* only.

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: $(a \vee b) \wedge (b \vee c)$



Properties of normal forms

Trade-offs between different normal forms

Normal forms that are more efficient to reason about are more expensive to construct and may be much bigger.

Complexity of operations on 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/\wedge 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	less dependent on n

Comparison: application domain

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

Applications of BDDs in planning

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

Other applications of BDDs etc.

- ▶ diagnosis (Huang & Darwiche, AAAA'05) based on DNNFs (Darwiche 01).
- ▶ LTL model-checking (Burch, Clarke, Long, McMillan, Dill, 1993)

Summary

- ▶ state-space search
- ▶ propositional satisfiability: find one path in a transition system
- ▶ other symbolic techniques (BDD, DNNF, ...):
 - ▶ representation of sets and relations as formulas
 - ▶ computation of set and relation operations by formula manipulation