Belief Tracking for Planning with Sensing (Tutorial)

Blai Bonet

Universidad Simón Bolívar

2nd Brazilian Conf. on Intelligent Systems (BRACIS)

Fortaleza, Brazil 2013

(joint work with Hector Geffner)

Recap Early Days of AI: Programming and Methodology

Many of the contributions had to do with:

– programming

– representation of knowledge in programs

It was common to find dissertations in AI that:

– pick up a task and domain X

– analyze how the task is solved

– capture this reasoning in a **program**

The dissertation was

– a **theory** about X, and

– a **program** implementing the theory, **tested** on a few examples

Great ideas came out ... but there was a problem ...

[Intro based on slides by H. Geffner]

Methodological Problem: Generality

Theories expressed as programs are **not falsifiable**:

- when program fails, the blame is always on **'missing knowledge'**

Methodological Problem: Generality

Theories expressed as programs are **not falsifiable**:

- when program fails, the blame is always on **'missing knowledge'**

Three approaches to this problem:

– narrow the domain (expert systems)

  - **problem:** lack of generality

– accept the program as an illustration, a demo

  - **problem:** limited scientific value

– fill up the missing value (intuition, commonsense)

  - **problem:** not clear how to do; not successful so far

AI Research Today

Recent issues of AIJ, JAIR, AAAI or IJCAI shows papers on:

- **SAT and Constraints**
- **Search and Planning**
- **Probabilistic Reasoning**
- **Probabilistic Planning**
- Multi-Agent Systems
- Inference in First-Order Logic
- Machine Learning
- Natural Language
- Vision and Robotics

First four areas often deemed as **techniques**, but it is more accurate to think about them in terms of **models and solvers**

AI Models and Solvers

$$\textit{Problem} \Longrightarrow \boxed{\textit{Solver}} \Longrightarrow \textit{Solution}$$

Some basic models and solvers currently considered in AI:

– **Constraint Satisfaction/SAT:** find state that satisfies constraints
– **Bayesian Networks:** find probability over variable given observations
– **Planning:** find action sequence or policy that produces desired state
– **Answer Set Programming:** find answer set of logic program

- Solvers for these models are general; not **tailored** to specific instances
- Models are all **intractable**, and some very **expressive** (POMDPs)
- Solvers all have a clear and crisp scope
- Challenge is mainly **computational**: how to scale up
- Methodology is **empirical**: benchmarks and competitions

Example: Solvers for SAT and CSPs

SAT is the problem of determining whether there is a **truth assignment** that satisfies a set of clauses

$$x \vee y \vee \neg z \vee \neg w \vee \cdots$$

Problem is **NP-Complete**: in practice, it means worst-case behavior of SAT algorithms is **exponential** in number of variables $(2^{100} = 10^{30})$

Current SAT solvers manage to solve problems with **thousands of variables and clauses**, and are used widely (circuit design, verification, planning, etc)

Constraint Satisfaction Problems (CSPs) generalize SAT by considering non-boolean variables, and constraints that are not clauses

Basic Planning Model and Task

Planning is the **model-based approach** to autonomous behavior:

– a system can be in one of many **states**

– states assign **values** to a set of **variables**

– **actions** change the values of certain variables

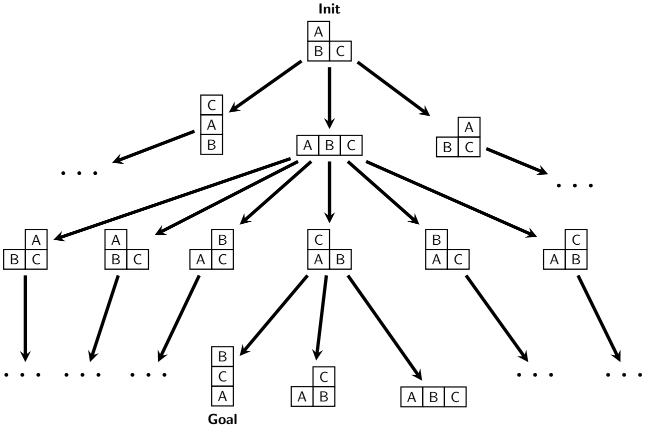
Basic task: find **action sequence** to drive **initial state** into **goal state**

$$\mathit{Model} \Longrightarrow \boxed{\mathit{Box}} \Longrightarrow \mathit{Action\ Sequence}$$

Complexity: NP-hard; i.e., exponential in number of vars in **worst case**

Box is generic: should work on any domain no matter what it is about

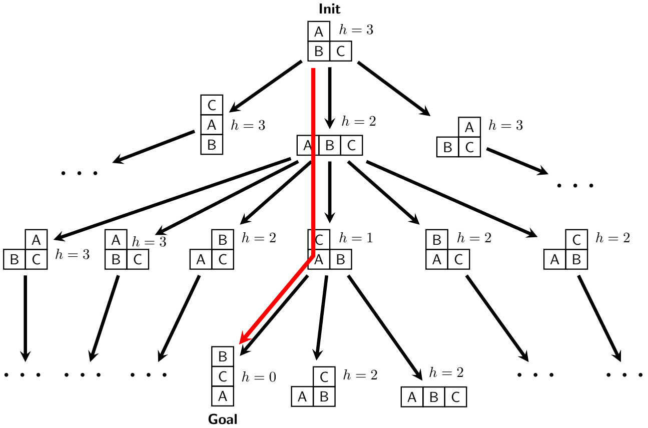
Example: Blocksworld



Task: Given actions that move a 'clear' block to the table or onto another 'clear' block, **find a plan** to achieve given goal

Question: How to find a path in graph of **exponential size** in # blocks?

Plan Found with Heuristics Derived Automatically



Heuristic evaluations $h(s)$ provide 'focus' and 'sense of direction'

Heuristic functions are calculated **automatically** and **efficiently** in a **domain-independent** manner from high-level description of problem

Summary

- Research agenda in AI is clear: **solvers** for a class of **models**
- **Solvers** unlike other programs are **general** as they do not target individual problems but families of problems (**models**)
- The main challenge is **computational**: how to scale up
- Worst-case complexity shouldn't be impediment to meaningful solutions
- **Structure** of problems must be recognized and **exploited**
- Progress is measured **empirically**

Agenda for the Rest of the Talk

- Introduction to planning models and languages
- Planning under uncertainty: non-det actions and incomplete information
- Belief tracking in planning
- Discussion

Planning Models and Languages

How to develop systems or ‘agents’ that make decisions on their own?

Autonomous Behavior in AI: The Control Problem

The key problem is to select the **action to do next**. This is the so-called **control problem**.

Three approaches to this problem:

Autonomous Behavior in AI: The Control Problem

The key problem is to select the **action to do next**. This is the so-called **control problem**.

Three approaches to this problem:

- **Programming-based:** specify control by hand
  - **Advantage:** domain-knowledge easy to express
  - **Disadvantage:** cannot deal with situations not anticipated by programmer

Autonomous Behavior in AI: The Control Problem

The key problem is to select the **action to do next**. This is the so-called **control problem**.

Three approaches to this problem:

- **Programming-based:** specify control by hand

- **Learning-based:** learn control from experience
  - **Advantage:** does not require much knowledge in principle
  - **Disadvantage:** in practice, right features needed, incomplete information is problematic, and unsupervised learning is slow

Autonomous Behavior in AI: The Control Problem

The key problem is to select the **action to do next**. This is the so-called **control problem**.

Three approaches to this problem:

- **Programming-based:** specify control by hand
- **Learning-based:** learn control from experience
- **Model-based:** specify problem by hand, derive control automatically
  - **Advantage:** flexible, clear, and domain-independent
  - **Disadvantage:** need a model; computationally **intractable**

Autonomous Behavior in AI: The Control Problem

The key problem is to select the **action to do next**. This is the so-called **control problem**.

Three approaches to this problem:

– **Programming-based:** specify control by hand

– **Learning-based:** learn control from experience

– **Model-based:** specify problem by hand, derive control automatically

Approaches are not orthogonal; and successes and limitations in each . . .

Model-based approach to intelligent behavior called Planning in AI

Classical Planning: Simplest Model

Model with **deterministic** actions under **complete knowledge**

Characterized by

- a finite **state space** S
- **known** initial state $s_0 \in S$
- subset $S_G \subseteq S$ of **goal states**
- actions $A(s) \subseteq A$ executable at state s
- **deterministic** transition function $f : S \times A \to S$ such that $f(s, a)$ is state after applying action $a \in A(s)$ in state s
- non-negative costs $c(s, a)$ for applying action a in state s

Abstract model that works at 'flat' representation of problem

Solutions (Plans)

Since **known** initial state and action outcomes can be **predicted**, solution is **fixed** action sequence $\pi = \langle a_0, a_1, \ldots, a_n \rangle$

The sequence π defines a **state trajectory** (path) $\langle s_0, s_1, \ldots, s_{n+1} \rangle$:

– s_0 is initial state

– $a_i \in A(s_i)$ is an applicable action at state s_i, $i = 0, \ldots, n$

– $s_{i+1} = f(s_i, a_i)$ is result of applying action a_i at state s_i

– s_{n+1} is a goal state; i.e., $s_{n+1} \in S_G$

Its **cost** is $c(\pi) = c(s_0, a_0) + c(s_1, a_1) + \cdots + c(s_n, a_n)$

It is **optimal** if its cost is minimum among all solutions

Uncertainty but No Feedback: Conformant Planning

Characterized by

- a finite state space S
- **subset of possible initial states** $S_0 \subseteq S$
- subset $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ executable at state s
- **non-deterministic** transition function $F : S \times A \to 2^S$ such that $F(s, a)$ is non-empty subset of possible states after applying action a in state s
- non-negative costs $c(s, a)$ for applying action a in state s

Solution still **action sequence** but must achieve goal from each **possible initial state and transition**

More complex than **classical planning**; checking if given action sequence is solution is **intractable** (for succinctly-described models)

Probabilistic Planning: Markov Decision Processes (MDPs)

Characterized by

- a finite state space S
- known initial state $s_0 \in S$
- subset $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ executable at state s
- **transition probabilities** $P(s'|s,a)$ of reaching state s' after applying action a in state s
- non-negative costs $c(s,a)$ for applying action a in state s

Solution is **function (policy)** that maps states into actions

Cost of solution is **expected cost to reach goal from initial state**

Optimal solution has minimum expected cost to reach goal

Partially Observable MDPs (POMDPs)

POMDPs are probabilistic models that are **partially observable**

Characterized by

- a finite state space S
- initial **distribution (belief)** b_0 over states
- subset $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ executable at state s
- transition probabilities $P(s'|s, a)$ for each $s, s' \in S$ and $a \in A(s)$
- finite set of **observable tokens** O
- **sensor model** given by probabilities $P(o|s', a)$ for observing token $o \in O$ after reaching s' when last action done is a

Solution is **policy** mapping belief states (distributions) into actions

Optimal solution minimizes **expected cost** to reach goal state from b_0

Planners

A planner is a **solver** over a **class of models**

– input is a model description

– output is a controller (solution)

$$\mathit{Model} \Longrightarrow \boxed{\mathit{Planner}} \Longrightarrow \mathit{Controller}$$

Different models and solution forms: uncertainty, feedback, costs, ...

Model described with **planning language** (Strips, PDDL, PPDDL, ...)

Languages

Models specified with representation languages

Expressivity and **succinctness** have impact on complexity (more below)

Flat languages: states and actions have no (internal) structure
(*good for understanding models, solutions and algorithms*)

Factored languages: states and actions are specified with variables
(*good for describing complex problems with few bits*)

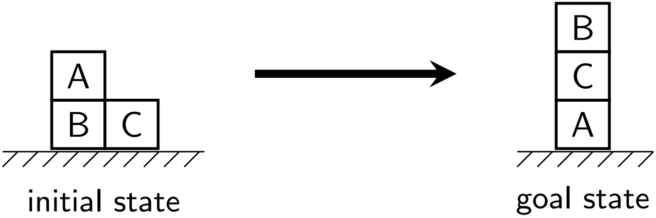
Implicit, through functions: states and actions directly coded
(*good for efficiency, used to deploy solutions*)

Factored Language: Propositional

Model specified in **compact form** using high-level language

- finite set F of propositional variables (atoms)
- an initial state $I \subseteq F$
- a goal description $G \subseteq F$
- finite set A of operators; each operator $a \in A$ given by
  - **precondition** $pre(a) \subseteq F$ (tell states on which action is executable)
  - **conditional effects** $a : C \to C'$ where C, $C' \subseteq Literals(F)$
- non-negative costs $c(a)$ for applying actions $a \in A$

Example: Blocksworld



Atoms: Clear(?x), On(?x,?y), OnTable(?x)

Actions: Move(?x,?y,?z), MoveToTable(?x), MoveFromTable(?x,?y)

Example: Blocksworld in PDDL

```
(define (domain BLOCKS)
  (:requirements :strips)
  (:predicates (clear ?x) (on ?x ?y) (ontable ?x))

  (:action move
     :parameters (?x ?y ?z)
     :precondition (and (clear ?x) (clear ?z) (on ?x ?y))
     :effect (and (not (clear ?z)) (not (on ?x ?y)) (on ?x ?z) (clear ?y)))

  (:action move_to_table
     :parameters (?x ?y)
     :precondition (and (clear ?x) (on ?x ?y))
     :effect (and (not (on ?x ?y)) (clear ?y) (ontable ?x)))

  (:action move_from_table
     :parameters (?x ?y)
     :precondition (and (ontable ?x) (clear ?x) (clear ?y))
     :effect (and (not (ontable ?x)) (not (clear ?y)) (on ?x ?y)))
)

(define (problem BLOCKS_3_1)
  (:domain BLOCKS)
  (:objects A B C)
  (:init (clear A) (clear C) (on A B) (ontable B) (ontable C))
  (:goal (and (on B C) (on C A))))
```

From Language to Model

Problem $P = \langle F, A, I, G, c \rangle$ mapped into model $\mathcal{S}(P) = \langle S, A, f, s_0, S_G, c' \rangle$:

- states S are all the 2^n **truth-assignments** to atoms in F, $|F| = n$
- initial state s_0 assigns **true** to all $p \in I$ and **false** to all $p \notin I$
- goal states $S_G = \{s : s \models G\}$
- same actions A, with $A(s) = \{a : s \models pre(a)\}$
- outcome $f(s, a)$ defined by action's effects (in standard way)
- costs $c'(s, a) = c(a)$

Size of state model is **exponential** in the size of problem P

Factored Language: Multi-valued Variables

Another language based on **multi-valued** variables

A problem is tuple $P = \langle V, A, I, G, c \rangle$ where:

– V is finite set of **variables** X, each with **finite domain** D_X

– initial state I given by **complete valuation of variables**

– goal G that is a **partial valuation over variables**

– A is finite set of operators; each operator $a \in O$ has

  - precondition $pre(a)$ which is a partial valuation of variables
  - conditional effects $a : C \to C'$ where C and C' are partial valuations

– non-negative costs $c(a)$ for actions $a \in A$

Finding Solutions: Algorithms

Solution is path from initial state to goal in an **exponential graph**

State-of-the-art algorithms perform **search in implicit graph** using heuristics to guide the search

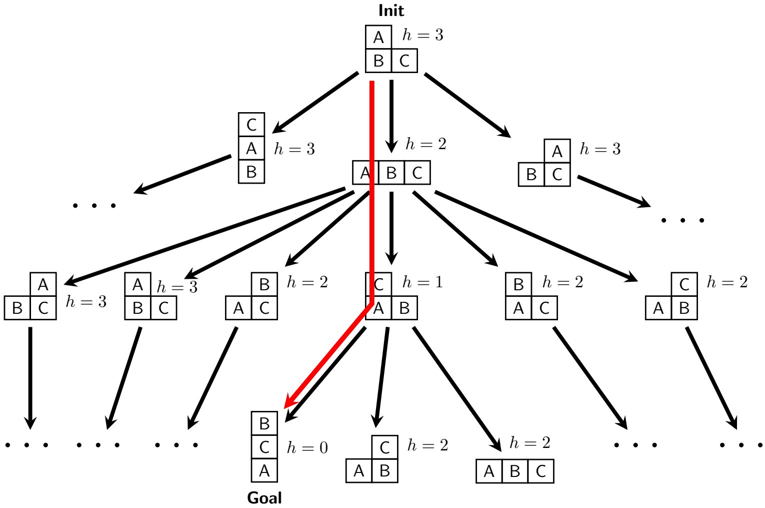
Powerful **heuristics automatically extracted** from problem description

Approach is **general and sucessful**: able to solve large problems quickly

Planners: LAMA-11, FF, ...

Benchmarks: thousands ... IPC repository (over 80 domains / 3,500 problems)

Finding Solutions: Blocksworld



Planning under Uncertainty

Motivation

Classical planning works!

- it is able to solve problems with thousands of atoms and actions fast

Model is simple, but useful:

- operators may be non-primitive; abstractions of policies
- **closed-loop replanning** is able to cope with uncertainty sometimes

There are some limitations:

- can't model **uncertainty on outcome of actions**
- can't deal with **incomplete information** (partial sensing)
- cost structure is very simple
- ...

Motivation

Two ways of handling limitations:

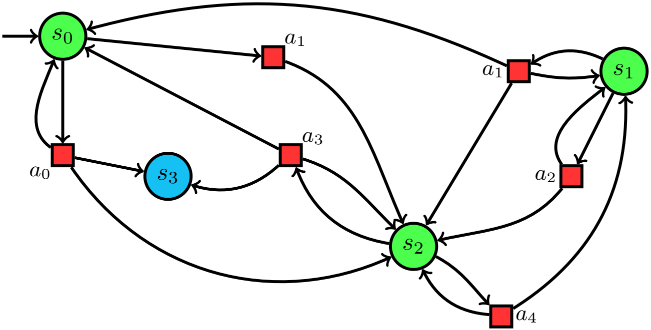
- extend scope of **current classical solvers** (translations / compilation)
- develop **new solvers for extended models**

(Fully Observable) State Model with Non-Det Actions

- finite state space S
- known initial state s_0
- goal states $S_G \subseteq S$
- actions $A(s) \subseteq A$ executable at state s
- **non-deterministic** transition function $F : S \times A \rightarrow 2^S$ such that $F(s, a)$ is subset of states that **may** result after executing a at s
- non-negative costs $c(s, a)$ of applying action a in state s

Current state is always fully observable to agent

Example: Simple Problem (AND/OR Graph)



- 4 states: $S = \{s_0, \ldots, s_3\}$
- 5 actions: $A = \{a_0, a_1, a_2, a_3, a_4\}$
- 1 goal: $S_G = \{s_3\}$
- $A(s_0) = \{a_0, a_1\}$; $A(s_1) = \{a_1, a_2\}$
- $F(s_0, a_0) = \{s_0, s_2, s_3\}$
- $F(s_1, a_1) = \{s_0, s_1, s_2\}$
- $F(s_0, a_1) = \{s_2\}$
- ...

Solutions (Controllers)

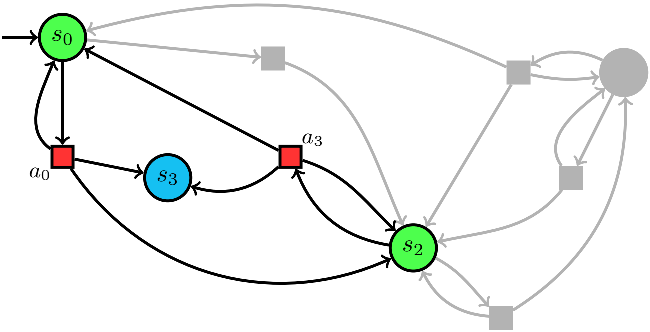
Solution **cannot** be a sequence of actions because agent **cannot predict the outcome of actions**

Since states are fully observable and agent knows model, the agent can be **prepared** for any **possible outcome**

Such controller is called **contingent** (with full observability)

A controller is a **function that maps states into actions**

Example: Solution



Controller π:

- initial state s_0
- $\pi(s_0) = a_0$
- $\pi(s_2) = a_3$

Some executions:

- $\langle s_0, s_0, s_0, s_3 \rangle$ **successful**
- $\langle s_0, s_2, s_0, s_0, s_2, s_2, s_3 \rangle$ **successful**
- $\langle s_0, s_2, s_0, s_2, s_0, s_2, s_0, \ldots \rangle$ **unfair!**

Agent with Partial Information

Agent has **partial information** when it doesn't **fully see current state**

Different ways to model sensing; most frequent is the POMDP model:

- finite set O of **observable tokens**
- **environment produces** one such token **after action is applied**
- agent **receives token** (it doesn't see state directly)
- token may depend on **current state** and **action leading to it**

Partially Observable MDPs (POMDPs)

POMDPs are probabilistic models that are **partially observable**

Characterized by

- a finite state space S
- **initial distribution (belief)** b_0 over states
- subset $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ executable at state s
- transition probabilities $P(s'|s,a)$ for each $s, s' \in S$ and $a \in A(s)$
- finite set of **observable tokens** O
- **sensor model** given by probabilities $P(o|s',a)$ for observing token o after reaching s' when last action done is a

Solution is **policy** mapping belief states (distributions) into actions

Belief States and Belief Tracking (POMDPs)

Agent must keep track of **possible current states** in the form of a **distribution over states**; such distributions are called **belief states**

Belief States and Belief Tracking (POMDPs)

Agent must keep track of **possible current states** in the form of a **distribution over states**; such distributions are called **belief states**

The initial belief state is b_0 (distribution for initial states)

When agent has belief state b, then

– **after executing action a,**

$$b_a(s') \ = \ \sum_s b(s)\, P(s'|s,a) \qquad \text{(progression)}$$

– **after executing action a and receiving token o,**

$$b_a^o(s') \ \propto \ b_a(s')\, P(o|s',a) \qquad \text{(filtering)}$$

Beliefs states depend on history of actions and observations!

Model for Non-Det Planning with Sensing (Logical POMDPs)

Characterized by

– finite state space S

– subset of possible initial states $S_I \subseteq S$

– subset of goal states $S_G \subseteq S$

– actions $A(s) \subseteq A$ executable at state s

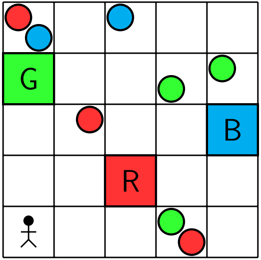
– non-deterministic transition function $F : S \times A \to 2^S$

– finite set of **observable tokens** O

– **sensor model** $O(s,a) \subseteq O$ with $O(s,a) \neq \emptyset$

– non-negative costs $c(s,a)$ for applying action a in state s

Example: Collecting Colored Balls



Agent senses presence of balls (and their colors) in current cell

Observable tokens $O = \{000, 001, 010, \ldots, 111\}$ (i.e., 3 bits of information)

- **First bit** tells whether there is a **red ball** in same cell of agent
- **Second bit** tells whether there is a **green ball** in same cell of agent
- **Third bit** tells whether there is a **blue ball** in same cell of agent

Belief States and Belief Tracking (Logical POMDPs)

Agent must keep track of **possible current states** in the form of a **subset of states**; such subsets are called **belief states**

The initial belief state is $b_0 = S_I$ (possible initial states)

When agent has belief state b, then

– **after executing action a,**

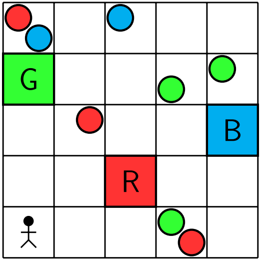
$$b_a = \{s' : s' \in F(s, a) \text{ and } s \in b\} \qquad \text{(progression)}$$

– **after executing action a and receiving token o,**

$$b_a^o = \{s' \in b_a : o \in O(s', a)\} \qquad \text{(filtering)}$$

Beliefs states depend on history of actions and observations!

Example: Belief Tracking on Collecting Colored Balls



- Initial belief $b_0 = \{\text{states w/ agent at } (0,0) \text{ and no balls at } (0,0)\}$ $\quad |b_0| \approx 10^{10}$

- For belief $b = b_0$ and action $a = up$,

$b_a = \{\text{states w/ agent at } (0,1) \text{ and no balls at } (0,0)\}$ $\quad |b_a| \approx 10^{10}$

- Then, agent receives the observation $o = 100$,

$b_a^o = \{\text{states w/ agent at } (0,1)\text{, no balls at } (0,0)\text{, and red balls at } (0,1)\}$ $\quad |b_a^o| \approx 10^9$

POMDPs as Non-Deterministic Planning in Belief Space

From model $P = \langle S, A, F, S_I, S_G, O, c \rangle$, construct **fully observable non-deterministic** model in **belief space** $\mathcal{B}(P) = \langle S', A', F', s'_0, S'_G, c' \rangle$

POMDPs as Non-Deterministic Planning in Belief Space

From model $P = \langle S, A, F, S_I, S_G, O, c \rangle$, construct **fully observable non-deterministic** model in **belief space** $\mathcal{B}(P) = \langle S', A', F', s'_0, S'_G, c' \rangle$

- states S' are all the belief states (distributions or subsets)
- initial state s'_0 is initial belief
- goal states S'_G are beliefs that only deem possible goals in S_G
- actions $A'(b) = \{a : a \in A(s)$ for states s deemed possible by $b\}$
- non-deterministic transitions $F'(b, a) = \{b_a^o : o$ is possible after a in $b\}$
- action costs $c'(b, a) = \max_{s \in b} c(s, a)$

Akin to determinization of Non-det. Finite Automata (NFA)!

Language for Planning with Sensing (Logical POMDPs)

Characterized by:

- V is finite set of variables X, each with finite domain D_X
- initial states given by **clasues** I
- goal description G that is **partial valuation**
- finite set A of actions with prec. and non-deterministic cond. effects
- **observable variables** V' (not necessarily disjoint from V)
- **sensing formulas** $W_a(Y = y)$ for each action a and literal $Y = y$
- non-negative costs $c(a)$ for applying action a

Observable tokens are full valuations over observable variables V'

Construction of Sensing Model

States and transition function constructed in standard way

Sensing model given by:

- observable tokens O are all the **full valuations** over observable vars V'
- possible tokens at state s after applying action a are

$$O(s, a) = \{o : s \models W_a(Y = y) \text{ where } o \models Y = y\}$$

Complexity Issues

With n variables (propositional or multi-valued), there are:

– **exponential** number of states

– **double exponential** number of belief states

Impact on complexity?

Decision problem: Is there a solution (plan) for given problem P?

	deterministic	non-deterministic
full observability	**PSPACE**	EXP
no observability	EXPSPACE	EXPSPACE
partial observability	**EXPSPACE**	**2EXP**

[Bylander, AIJ 1994; Rintanen, ICAPS 2004]

Algorithms: Finding Solutions

Algorithms perform some type of **search** in either

- state space
- belief space

	deterministic	non-deterministic
full obs.	state space / OR graph	state space / AND/OR graph
no obs.	belief space / OR graph	belief space / OR graph
partial obs.	belief space / AND/OR graph	belief space / AND/OR graph

Belief Tracking

Motivation

Two **fundamental tasks** to be solved for **planning with sensing**, both intractable for problems in compact form:

– tracking of belief states (i.e. representation of search space)

– action selection for achieving the goal (i.e. type of search)

We now focus on the belief tracking task

[B & Geffner, AAAI 2012; B & Geffner, IJCAI 2013]

Belief Tracking in Planning (BTP)

Definition (BTP)

Given execution $\tau = \langle a_0, o_0, a_1, o_1, \ldots, a_n, o_n \rangle$ **determine** *whether*

- *the execution* τ *is possible, and*
- *whether* b_τ*, the belief that results of executing* τ*, achieves the goal*

Theorem

BTP is NP-hard and coNP-hard. Indeed, BTP is complete for P^{NP} *with respect to polynomial-time Turing reductions*

Basic Algorithm: Flat Belief Tracking

Explicit representation of beliefs states as sets of states

Definition (Flat Belief Tracking)

Given belief b at time t, and action a (applied) and observation o (obtained), the belief at time $t+1$ is the belief b_a^o given by:

$$b_a = \{s' : s' \in F(s,a) \text{ and } s \in b\}$$

$$b_a^o = \{s' : s' \in b_a \text{ and } s' \models W_a(\ell) \text{ for each } \ell \text{ s.t. } o \models \ell\}$$

- Flat belief tracking is sound and complete for **every formula**
- Time and space complexity is **exponential in** $|V \cap V_U|$ where $V_U = V \setminus V_K$ and V_K are the variables that are **determined**

Other Approaches for Logical POMDPs

Flat belief tracking is **explicit representation** of beliefs as subsets of states

It is called flat because doesn't **exploit structure** in problem and states

Other options for states defined in terms of variables (various authors):

– **as CNF/DNF formulas**:
  - **Advantage:** economic updates, succinct representation
  - **Disadvantage:** intractable query answering

– **as OBDD formulas**:
  - **Advantage:** tractable query answering
  - **Disadvantage:** size of representation may explode quickly

– **knowledge compiled at propositional level**:
  - **Advantage:** tractable inference and representation (for **bounded width**)
  - **Disadvantage:** scope limited to deterministic planning

Belief Tracking in POMDPs: Particle Filters

Probabilistic flat belief tracking is exponential in number of variables:

$$\begin{aligned} b_a(s') &= \textstyle\sum_s b(s)\, P(s'|s,a) \\ b_a^o(s') &\propto b_a(s')\, P(o|s',a) \end{aligned}$$

Particle filter B **approximate** belief b by (multi)set of **unweighted samples**

– probability of $X = x$ approximated by **ratio** of **samples** in B where $X = x$ holds

Next filter B_{k+1} obtained from B_k, action a and observation o in two steps:

– sample s' from S with probability $P(s'|s,a)$ for each s in B_k

– (re)sample new set of samples by sampling each s' with weight $P(o|s',a)$

Two serious problems with particle filters:

– particles **die out** if many probabilities are zero

– **non-unique representation** of beliefs

Complexity of BTP (in Logical POMDPs) (Sketch)

Definition (BTP)

Given execution $\tau = \langle a_0, o_0, a_1, o_1, \ldots, a_n, o_n \rangle$ **determine** *whether*

- *the execution* τ *is possible, and*
- *whether* b_τ*, the belief that results of executing* τ*, achieves the goal*

Theorem

BTP is NP-hard and coNP-hard. Indeed, BTP is complete for P^{NP} *with respect to polynomial-time Turing reductions*

Formally, BTP is the language:

$$\text{BTP} = \{\, \langle P, \tau \rangle : P \text{ is problem, } \tau \text{ is possible execution, } b_\tau \models G \,\}$$

Complexity of BTP (in Logical POMDPs) (Sketch)

BTP $= \{ \langle P, \tau \rangle :$ P is problem, τ is possible execution, $b_\tau \models G \}$

Complexity of BTP (in Logical POMDPs) (Sketch)

$$\text{BTP} = \{ \langle P, \tau \rangle : P \text{ is problem, } \tau \text{ is possible execution, } b_\tau \models G \}$$

Inclusion

Sufficient to give algorithm for BTP that uses SAT **oracle** and that runs in polynomial time

Let n be length of τ

Complexity of BTP (in Logical POMDPs) (Sketch)

$$\text{BTP} = \{ \langle P, \tau \rangle : P \text{ is problem, } \tau \text{ is possible execution, } b_\tau \models G \}$$

Inclusion

Sufficient to give algorithm for BTP that uses SAT **oracle** and that runs in polynomial time

Let n be length of τ

– **To check that τ is a possible execution:** call the SAT solver n times with theories Δ_t that encode the prefix of τ of length t ($t = 0, \ldots, n$) and the satisfaction of preconditions or observation at time t

Complexity of BTP (in Logical POMDPs) (Sketch)

$$\text{BTP} = \{\, \langle P, \tau \rangle : P \text{ is problem, } \tau \text{ is possible execution, } b_\tau \models G \,\}$$

Inclusion

Sufficient to give algorithm for BTP that uses SAT **oracle** and that runs in polynomial time

Let n be length of τ

– **To check that τ is a possible execution:** call the SAT solver n times with theories Δ_t that encode the prefix of τ of length t ($t = 0, \ldots, n$) and the satisfaction of preconditions or observation at time t

– **To check $b_\tau \models G$:** call the SAT solver one more time with theory that encodes τ and the satisfaction of goal G

Complexity of BTP (in Logical POMDPs) (Sketch)

$$\text{BTP} = \{ \langle P, \tau \rangle : P \text{ is problem, } \tau \text{ is possible execution, } b_\tau \models G \}$$

Hardness

P^{NP} is set of decisions problems that can be decided in polytime using a SAT oracle

Complexity of BTP (in Logical POMDPs) (Sketch)

$$\text{BTP} = \{ \langle P, \tau \rangle : P \text{ is problem}, \tau \text{ is possible execution}, b_\tau \models G \}$$

Hardness

P^{NP} is set of decisions problems that can be decided in polytime using a SAT oracle

Therefore, to show hardness, enough to show that UNSAT can be reduced in polytime to BTP since then every call to SAT oracle can be replaced by a call to a BTP oracle

Complexity of BTP (in Logical POMDPs) (Sketch)

$$\mathsf{BTP} = \{ \langle P, \tau \rangle : P \text{ is problem, } \tau \text{ is possible execution, } b_\tau \models G \}$$

Hardness

Let $\Delta = \{C_1, C_2, \ldots, C_m\}$ be a CNF over variables $X_1, \ldots, X_n$

We construct in polytime problem P and execution τ:

Complexity of BTP (in Logical POMDPs) (Sketch)

$$\text{BTP} = \{\, \langle P, \tau \rangle : P \text{ is problem, } \tau \text{ is possible execution, } b_\tau \models G \,\}$$

Hardness

Let $\Delta = \{C_1, C_2, \ldots, C_m\}$ be a CNF over variables $X_1, \ldots, X_n$

We construct in polytime problem P and execution τ:

– variables $V = \{X_1, \ldots, X_n, Q\}$ and obs $V' = \{Z_1, \ldots, Z_m\}$

Complexity of BTP (in Logical POMDPs) (Sketch)

$$\text{BTP} = \{ \langle P, \tau \rangle : P \text{ is problem}, \tau \text{ is possible execution}, b_\tau \models G \}$$

Hardness

Let $\Delta = \{C_1, C_2, \ldots, C_m\}$ be a CNF over variables $X_1, \ldots, X_n$

We construct in polytime problem P and execution τ:

- variables $V = \{X_1, \ldots, X_n, Q\}$ and obs $V' = \{Z_1, \ldots, Z_m\}$
- $I = \emptyset$ and $G = \{Q = true\}$

Complexity of BTP (in Logical POMDPs) (Sketch)

$$\text{BTP} = \{ \langle P, \tau \rangle : P \text{ is problem, } \tau \text{ is possible execution, } b_\tau \models G \}$$

Hardness

Let $\Delta = \{C_1, C_2, \ldots, C_m\}$ be a CNF over variables $X_1, \ldots, X_n$

We construct in polytime problem P and execution τ:

– variables $V = \{X_1, \ldots, X_n, Q\}$ and obs $V' = \{Z_1, \ldots, Z_m\}$

– $I = \emptyset$ and $G = \{Q = true\}$

– empty actions $a_1, \ldots, a_m$ with sensing formulas

- $W_{a_i}(Z_i = true) = C_i \vee Q$
- $W_{a_i}(Z_j = true) = false$

Complexity of BTP (in Logical POMDPs) (Sketch)

$$\text{BTP} = \{ \langle P, \tau \rangle : P \text{ is problem}, \tau \text{ is possible execution}, b_\tau \models G \}$$

Hardness

Let $\Delta = \{C_1, C_2, \ldots, C_m\}$ be a CNF over variables $X_1, \ldots, X_n$

We construct in polytime problem P and execution τ:

– variables $V = \{X_1, \ldots, X_n, Q\}$ and obs $V' = \{Z_1, \ldots, Z_m\}$

– $I = \emptyset$ and $G = \{Q = true\}$

– empty actions $a_1, \ldots, a_m$ with sensing formulas

  - $W_{a_i}(Z_i = true) = C_i \vee Q$
  - $W_{a_i}(Z_j = true) = false$

– execution $\tau = \langle a_1, o_1, \ldots, a_m, o_m \rangle$ where o_i is V'-valuation that makes Z_i true and Z_j false for $j \neq i$

Complexity of BTP (in Logical POMDPs) (Sketch)

$$\text{BTP} = \{ \langle P, \tau \rangle : P \text{ is problem, } \tau \text{ is possible execution, } b_\tau \models G \}$$

Hardness

Analysis:

- initial belief contains all 2^{n+1} valuations over $X_1, \ldots, X_n, Q$
- after o_1, only valuations satisfying $C_1 \vee Q$ remain
- after o_2, only valuations satisfying $(C_1 \,\&\, C_2) \vee Q$ remain
- after o_i, only valuations satisfying $(C_1 \,\&\, \cdots \,\&\, C_i) \vee Q$ remain
- at the end, only valuations satisfying $\Delta \vee Q$ remain

Complexity of BTP (in Logical POMDPs) (Sketch)

$$\text{BTP} = \{ \langle P, \tau \rangle : P \text{ is problem, } \tau \text{ is possible execution, } b_\tau \models G \}$$

Hardness

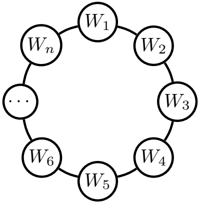
Analysis:

- initial belief contains all 2^{n+1} valuations over $X_1, \ldots, X_n, Q$
- after o_1, only valuations satisfying $C_1 \vee Q$ remain
- after o_2, only valuations satisfying $(C_1 \,\&\, C_2) \vee Q$ remain
- after o_i, only valuations satisfying $(C_1 \,\&\, \cdots \,\&\, C_i) \vee Q$ remain
- at the end, only valuations satisfying $\Delta \vee Q$ remain

Therefore, $b_\tau \models Q$ iff all valuations for $\neg Q$ are gone iff Δ is UNSAT

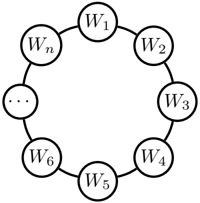
QED

Example: Non-det. Windows with Key (Unobs.)



- windows W_1, ..., W_n that can be open, closed, or locked
- agent doesn't know its position, windows' status, or key position
- goal is to have **all windows locked**
- when unlocked, **windows open/close non-det.** when agent moves
- to lock window: must close and then lock it with **key**
- key must be **grabbed** to lock windows
- **possible plan:** repeat n $\langle$Grab,Fwd$\rangle$; repeat n $\langle$Close,Lock,Fwd$\rangle$

Example: Non-det. Windows with Key (Unobs.)



Variables:

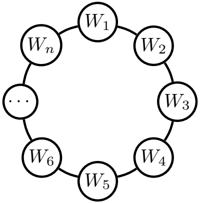
- Windows' status: $W_i \in \{open, closed, locked\}$
- Position of agent Loc $\in \{1, \dots, n\}$ and key KLoc $\in \{1, \dots, n, \text{hand}\}$

Actions:

- Close: Loc $= i$, $W_i = open \longrightarrow W_i = closed$
- Lock: Loc $= i$, $W_i = closed$, KLoc $= hand \longrightarrow W_i = locked$
- Grab: Loc $= i$, KLoc $= i \longrightarrow$ KLoc $= hand$
- Fwd: Loc $= i \longrightarrow$ Loc $= i + 1 \mod n$

  $W_i \neq locked \longrightarrow W_i = open \mid W_i = closed$

Example: Non-det. Windows with Key (Unobs.)



Flat belief tracking:

- single belief that initially contain $n^2 \times 3^n$ states
- each update operation (i.e., compute b_a or b_a^o) takes **exponential time**

Result:

- **intractable belief tracking**
- that likely translates into **intractable action selection**

Want: Factored Algorithm for Belief Tracking

Three key facts about **dynamic of information** in planning:

- don't need completeness for every formula. Only need to check validity of literals '$X = x$' appearing in **preconditions** and **goals**
- not every variable is "correlated" to each other
- uncertainty only propagates through conditional effects of actions

Can we exploit structure and "independence" among variables?

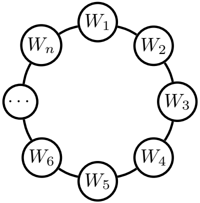
Insight!

Instead of tracking on one big problem P, track on several smaller **subproblems** P_X (simultaneously)

Hopefully, largest subproblem will be **much smaller** than P

Combined complexity: $\#$ subproblems $\times$ complexity largest P_X

Example: Non-deterministic Windows with Key (Unobs.)



Subproblems:

- **One subproblem P_i for each window W_i**
- Subproblem P_i involves only the variables W_i, Loc and KLoc
- Flat belief tracking is done in **parallel and independently** over all subproblems

Usage:

- Queries about window W_i are answered by **inspecting belief for subproblem P_i**

Result:

- **Sound and complete** belief tracking for planning
- **Combined time/space complexity**: $O(n^3)$ for n windows

Decompositions

A decomposition of problem P is pair $D = \langle T, B \rangle$ where

– T is subset of **target** variables, and

– **contexts** $B(X)$ for X in T is a subset of state variables

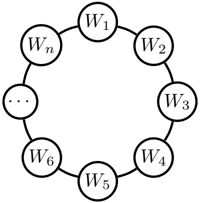
Decomposition $D = \langle T, B \rangle$ decomposes P into subproblems:

– one subproblem P_X for each target variable X in T

– subproblem P_X involves only state variables in $B(X)$

Belief tracking over a decomposition refers to **flat belief tracking** over the subproblems defined by the decomposition

Example: Non-deterministic Windows with Key (Unobs.)



Decomposition $D = \langle T, B \rangle$ where:

- $T = \{W_1, W_2, \ldots, W_n\}$ (target variables are window's status variables)
- $B(W_i) = \{W_i, \mathsf{Loc}, \mathsf{KLoc}\}$ for each $i = 1, \ldots, n$
- that is, total of n subproblems P_i with 3 variables each

Result:

- belief tracking over all subproblems gives **sound and complete algorithm**
- flat belief tracking on original problem has **exponential complexity** $O(n^2 3^n)$
- flat belief tracking on all subproblems has **combined complexity** $O(n^3)$

Soundness and Completeness

A belief tracking algorithm is **sound** with respect to queries $X = x$ if whenever the algorithm says that $X = x$ holds, then $X = x$ holds

A belief tracking algorithm is **complete** with respect to queries $X = x$ if whenever $X = x$ holds, then the algorithm says that $X = x$ holds

Soundness and Completeness

A belief tracking algorithm is **sound** with respect to queries $X = x$ if whenever the algorithm says that $X = x$ holds, then $X = x$ holds

A belief tracking algorithm is **complete** with respect to queries $X = x$ if whenever $X = x$ holds, then the algorithm says that $X = x$ holds

If b denotes the current (global) belief state and b_X denotes the current (local) belief state computed by the algorithm, the properties of soundness and completeness can be expressed as:

– **Sound:** $\Pi_X\, b \;\subseteq\; \Pi_X\, b_X$

– **Complete:** $\Pi_X\, b \;\supseteq\; \Pi_X\, b_X$

– **Sound and Complete:** $\Pi_X\, b \;=\; \Pi_X\, b_X$

How to Compute a Decomposition

Problem: how to automatically obtain decomposition $D = \langle T, B \rangle$ of problem P that gives a sound and complete belief tracking algorithm

- **Target variables** T given by vars appearing precondition and goals
- **Contexts** $B(X)$ defined using notions of relevance
- **Subproblems** P_X defined using projections

Relevance Notions

Different notions of relevance among variables define the contexts $B(X)$ in decompositions $D = \langle T, B \rangle$:

- Causal relevance: X is **causally relevant** to Y
- Evidential relevance: X is **evidentially relevant** to Y
- Relevance: X is **relevant** to Y

Akin to relevance notions in Bayesian networks!

Causal Relevance

X is a **direct cause** of Y

Induced by conditional effects:

$$a : X = x,\, C \;\longrightarrow\; Y = y,\, C'\,,$$

and sensing formulas:

$$W_a(Y = y) = \text{‘some formula involving } X\text{’}$$

Causal relevance is **reflexive-transitive closure** of direct causation

Evidential Relevance

X is **evidentially relevant** to Y

– Y is **causally relevant** to X:

$$Y \rightarrow_{dc} Z_1 \rightarrow_{dc} Z_2 \rightarrow_{dc} \cdots \rightarrow_{dc} X$$

– X is **observable**

Relevance

$\underline{X \text{ is } \textbf{relevant} \text{ to } Y}$

Relevance is **transitive closure** of causal and evidential relevance

I.e., there are variables $Z_1, Z_2, Z_3, \ldots$

$$X \ \rightarrow_c \ Z_1 \rightarrow_e \ Z_2 \ \rightarrow_c \ Z_3 \ \rightarrow_e \ \cdots \ \rightarrow_c \ Y$$

Subproblems P_X

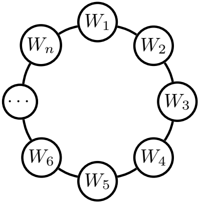
Subproblem P_X is problem P **projected** on the vars in $B(X)$

$P_X = \langle V_X, A_X, I_X, G_X, V'_X, W_X \rangle$ has:

- state variables $B(X)$ but same observables: $V_X = B(X)$, $V'_X = V'$
- only precondition and effects relevant to $B(X)$ are kept
- I_X and G_X are I and G **logically projected** on $B(X)$
- sensing formulas $W_a(Y = y)$ are **logically projected** on $B(X)$

Projection is basically the one used in planning when computing pattern-database (PDB) heuristics!

Example: Non-det. Windows with Key (Unobs.)



Variables:

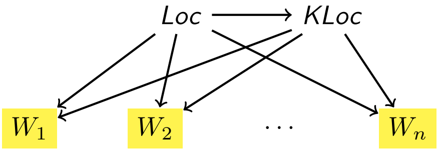
- Windows' status: $W_i \in \{open, closed, locked\}$
- Position of agent Loc $\in \{1, \ldots, n\}$ and key KLoc $\in \{1, \ldots, n, \text{hand}\}$

Actions:

- Close: Loc $= i$, $W_i = open \longrightarrow W_i = closed$
- Lock: Loc $= i$, $W_i = closed$, KLoc $= hand \longrightarrow W_i = locked$
- Grab: Loc $= i$, KLoc $= i \longrightarrow$ KLoc $= hand$
- Fwd: Loc $= i \longrightarrow$ Loc $= i + 1 \mod n$

  $W_i \neq locked \longrightarrow W_i = open \mid W_i = closed$

Example: Non-det. Windows with Key (Unobs.)



Variables:

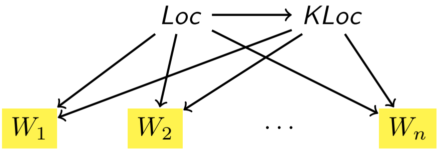
- Windows' status: $W_i \in \{open, closed, locked\}$
- Position of agent Loc $\in \{1, \dots, n\}$ and key KLoc $\in \{1, \dots, n, \text{hand}\}$

Actions:

- Close: Loc $= i$, $W_i = open \longrightarrow W_i = closed$
- Lock: Loc $= i$, $W_i = closed$, KLoc $= hand \longrightarrow W_i = locked$
- Grab: Loc $= i$, KLoc $= i \longrightarrow$ KLoc $= hand$
- Fwd: Loc $= i \longrightarrow$ Loc $= i + 1 \mod n$

  $W_i \neq locked \longrightarrow W_i = open \mid W_i = closed$

Example: Non-det. Windows with Key (Unobs.)



Contexts:

- Yellow variables are those appearing in preconditions and goals
- Variables **relevant** to W_i are W_i, Loc and KLoc
- Context for W_i is $B(W_i) = \{W_i, \mathsf{Loc}, \mathsf{KLoc}\}$

This problem has no observables or evidential relevances!

Factored Decomposition

Decomposition $F = \langle T_F, B_F \rangle$ where:

- target variables T_F are those in preconditions and goal
- contexts $B_F(X)$ given by variables Y **relevant** to X

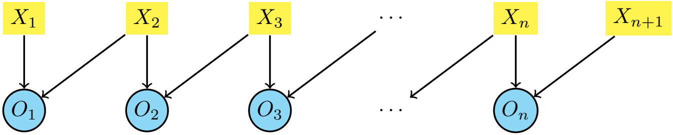
Theorem

Belief tracking over factored decomposition is **sound and complete**, *and exponential in the* **width** *of the problem*

Width of problem:

max number of **unknown** *state variables that are all* **relevant** *to a given precondition or goal variable* X

Example: Two-layer Problem



- $n+1$ state variables $X_1, \ldots, X_{n+1}$
- n observable variables $O_1, \ldots, O_n$ such that O_i is true iff $X_i = X_{i+1}$; i.e.,

$$W_a(O_i = true) = (X_i = X_{i+1})$$
$$W_a(O_i = false) = (X_i \neq X_{i+1})$$

- actions do not create causal relationships between state variables
- every state variable X_i is relevant to another state variable X_j
- **width** is $n+1$ and factored belief tracking is **exponential** ☹

Causal Decompositon

Decomposition $C = \langle T_C, B_C \rangle$ where:

- target variables T_F are precondition, goal **and observable variables**
- contexts $B_C(X)$ given by variables Y **causally relevant** to X

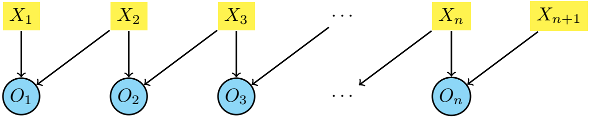
Theorem

Belief tracking over causal decomposition is **sound***, and exponential in the* **causal width** *of the problem*

Causal width of problem:

max number of **unknown** *state variables that are all* **causally relevant** *to a given precondition, goal or observable variable* X

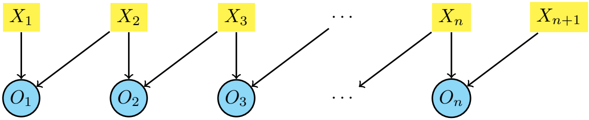
Example: Two-layer Problem



Factored decomposition $F = \langle T_F, B_F \rangle$:

- $T_F = \{X_1, \ldots, X_{n+1}\}$
- $B_F(X_i) = \{X_1, \ldots, X_{n+1}\}$
- **Width is** $n + 1$ ☹

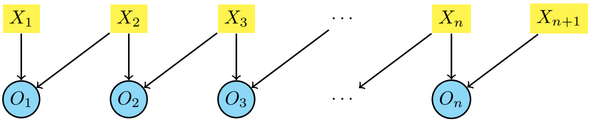
Example: Two-layer Problem



Causal decomposition $C = \langle T_C, B_C \rangle$:

- $T_C = \{X_1, \ldots, X_{n+1}\} \cup \{O_1, \ldots, O_n\}$
- $B_C(X_i) = \{X_i\}$, $B_C(O_i) = \{X_i, X_{i+1}\}$
- **Causal width is** 2 ☺

Example: Two-layer Problem



Causal decomposition $C = \langle T_C, B_C \rangle$:

- $T_C = \{X_1, \ldots, X_{n+1}\} \cup \{O_1, \ldots, O_n\}$
- $B_C(X_i) = \{X_i\}$, $B_C(O_i) = \{X_i, X_{i+1}\}$
- **Causal width is** 2 ☺

Result:

- Belief tracking over causal decomposition is polynomial ☺
- it is sound ☺
- but it is not complete! ☹

Complete Tracking over Causal Decomposition

Tracking over causal decomposition is **incomplete** because:

- two beliefs b_X and b_Y associated with target variables X and Y may interact and are not independent

Algorithm made complete by **enforcing consistency** among local beliefs:

$$b_X^{i+1} \;:=\; \Pi_{B_C(X)} \bowtie \{\, (b_Y^i)_a^o : Y \in T_C \text{ and relevant to } X \,\}$$

Resulting algorithm is:

- complete for the class of **causally decomposable problems** ☺
- space exponential in **causal width** ☺
- time exponential in **width** ☹

Wumpus and Minesweeper are causally decomposable

Causally Decomposable Problems

Large class of meaningful problems: Wumpus, Minesweeper, . . .

Causally Decomposable Problems

Large class of meaningful problems: Wumpus, Minesweeper, ...

Definition (Memory Variable)

State variable X is a memory variable when the value X^k at time point k can in an execution can be determined from an observation of the value X^i of X at any other time point i, the executed actions, and the initial belief

Examples of memory variables:

- static variables (i.e., unknown variables that do not change value)
- known or determined variables
- permutation variables [Amir & Russell, IJCAI 2003]

Causally Decomposable Problems

Definition (Causally Decomposable Problems)

Problem P is causally decomposable when for every pair of beams $B_C(X)$ and $B_C(X')$ with non-empty intersection, where X' is an observation variable, either:

1) *the variables in the intersection are all memory variables, or*

2) *there is target variable Z that is relevant to X or X' such that $B_C(Z) \supseteq B_C(X) \cup B_C(X')$*

– First case: interactions between local beliefs are captured with join

– Second case: interactions are captured by a bigger context (subproblem)

Effective Tracking over Causal Decomposition: Beam Tracking

Replaces costly join (time exponential in width) by effective **local consistency** until **fix point**: for all Y relevant to X

$$b_X^{i+1} \; := \; \Pi_{B_C(X)} \, (\, b_X^{i+1} \bowtie b_Y^{i+1})$$

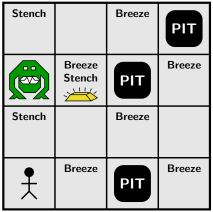
Beam tracking is:

– time and space exponential in **causal width** ☺

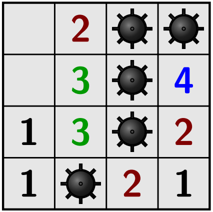
– sound and powerful, but not complete

– **practical algorithm** as it is general and effective ☺

Example: Wumpus and Minesweeper



Wumpus



Minesweeper

Factored belief tracking: exponential in **width** which is $O(n)$ for n cells

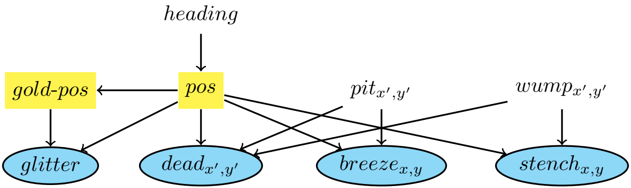
Beam tracking: exponential in **causal width** which is

– Wumpus: **constant** 4 for any number of cells

– Minesweeper: **constant** 9 for any number of cells

[DEMO]

Wumpus



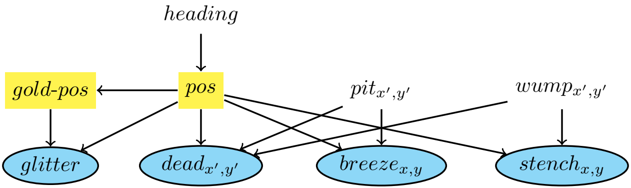
Variables:

- state vars: $heading$, $gold\text{-}pos$, pos, $pit_{x,y}$, $wump_{x,y}$
- observable: $glitter$, $breeze_{x,y}$, $stench_{x,y}$, $dead_{x,y}$

Actions:

- Fwd: $heading = 0$, $pos = (x, y) \longrightarrow pos = (x, y + 1)$
- RotR: $heading = i \longrightarrow heading = i + 1 \mod 4$
- RotL: $heading = i \longrightarrow heading = i - 1 \mod 4$
- Grab(x, y): $gold\text{-}pos = hand$ w/ prec $gold\text{-}pos = (x, y)$ and $pos = (x, y)$

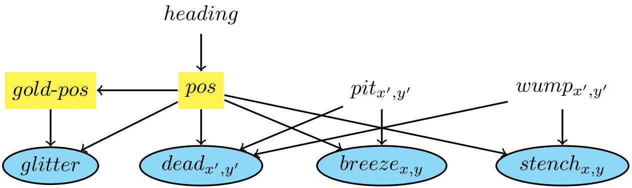
Wumpus



Sensor model:

- $W_a(glitter = true) = \bigvee_{x,y} (pos = (x,y) \wedge \text{gold-pos} = (x,y))$
- $W_a(breeze_{x,y} = true) = \bigvee_{x',y'} (pos = (x,y) \wedge pit_{x',y'})$
- $W_a(stench_{x,y} = true) = \bigvee_{x',y'} (pos = (x,y) \wedge wump_{x',y'})$
- $W_a(dead_{x,y} = true) = [pos = (x,y) \wedge (pit_{x,y} \vee wump_{x,y})$

Wumpus

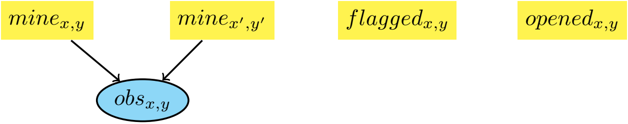


Contexts:

- $B_C(gold\text{-}pos) = \{gold\text{-}pos, pos, heading\}$
- $B_C(pos) = \{pos, heading\}$
- $B_C(glitter) = \{gold\text{-}pos, pos, heading\}$
- $B_C(breeze_{x,y}) = \{pos, heading\} \cup \{pit_{x',y'} : (x', y')$ adj to $(x, y)\}$
- $B_C(stench_{x,y}) = \{pos, heading\} \cup \{wump_{x',y'} : (x', y')$ adj to $(x, y)\}$
- $B_C(dead_{x,y}) = \{pos, heading, pit_{x,y}, wump_{x,y}\}$

Causal width is 4 because $heading$ and pos are **always known to agent**

Minesweeper



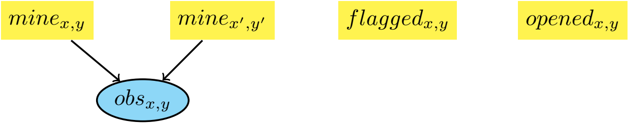
Variables:

- state vars: $mine_{x,y}$, $flag_{x,y}$, $opened_{x,y}$
- observable: $obs_{x,y}$ with domain $D = \{0, \ldots, 9\}$

Actions:

- Open(x, y): $opened_{x,y}$ with precondition $\neg flag_{x,y}$
- Flag(x, y): $flag_{x,y}$ with precondition $\neg mine_{x,y}$

Minesweeper

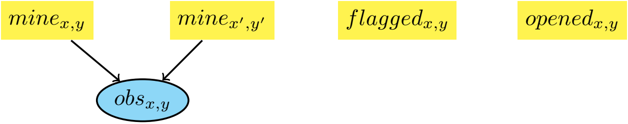


Sensor model:

- $W_{\mathsf{Open}(x,\,y)}(obs_{x,y} = 9) = mine_{x,y}$ (explosion)
- $W_{\mathsf{Open}(x,\,y)}(obs_{x,y} = k) = \neg mine_{x,y} \wedge \bigvee_{t \in N(x,y,k)} t$
- $W_{\mathsf{Open}(x,\,y)}(obs_{x',y'} = k) = true$ (no information)
- $W_{\mathsf{Flag}(x,\,y)}(obs_{x',y'} = k) = true$ (no information)

$N(x,y,k) =$ "terms over 8 cell variables $mine_{x',y'}$ surrounding (x,y) that make exactly k literals true"

Minesweeper



Contexts:

- $B_C(mine_{x,y}) = \{mine_{x,y}\}$
- $B_C(flag_{x,y}) = \{flag_{x,y}\}$
- $B_C(opened_{x,y}) = \{opened_{x,y}\}$
- $B_C(obs_{x,y}) = \{mine_{x',y'} : |x - x'| \leq 1, |y - y'| \leq 1\}$

Causal width is 9

Extensions

Framework supports exensions of the base model:

– defined variables

– (global) state constraints

Defined Variables

Variable Z with domain D_Z can be **defined** as:

– a function of a subset S_Z of state variables, or

– a function of the belief over such variables

For example, Z can be defined as true when $X = Y$, or when W is known

Such variables can be used in preconditions and goals by introducing a context in the decomposition that includes the variables in S_Z

Used in Wumpus because goal is given by set of clauses!

State Constraints

Used to restrict the value combinations of given subsets of state variables

A state constraint C is a formula over a subset of variables

Encoded by means of a **dummy** observable variable Y such that:

- Y is **always observed to be true**
- $W_a(Y = true) = C$ for every action a

Used in Battleship for good placement of ships!

[DEMO OF BATTLESHIP]

Discussion

Related Work

Belief tracking "compiled" at propositional level inside planning problem:

– Det. conformant planning [Palacios & Geffner, JAIR 2009]

– Det. contingent planning [Albore et al., IJCAI 2009; B & Geffner, IJCAI 2011, Shani & Brafman, IJCAI 2011; Brafman & Shani, AAAI 2012]

Belief tracking using non-flat representations:

– logical filtering [Amir & Russell, IJCAI 2003]

– OBDDs [Cimatti et al., AIJ 2004]

– CNF [Hoffmann & Brafman, ICAPS 2005, AIJ 2006]

– DNF/CNF [To et al., IJCAI 2011]

Conclusions

– Main challenge in planning is to achieve **generality and scalability**

– Progress continuosly assessed in **benchmarks and competitions**

– Planning with sensing is **belief tracking** plus **action selection**

– Factored BT is sound and complete, and exponential in **width**

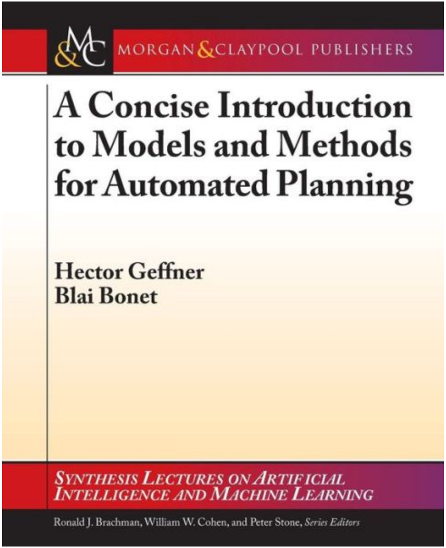
– Causal BT is sound and weak, but exponential in **causal width** which is often much smaller than width

– Beam tracking is sound and effective, and exponential in **causal width**

Challenges

- Effective action selection for planning with sensing isn't clear yet
  - algorithms + heuristics (or base policies)
- Deployment of these methods for other AI models
- Probabilistic belief tracking; applications like robotics; SLAM; . . .

New Book on AI Planning



Thanks. Questions?