Principles of AI Problem Solving
Tutorial IJCAI-05

Adnan Darwiche (UCLA, Los Angeles)
Rina Dechter (UCI, Irvine)
H. Geffner (UPF, Barcelona)

Problems

Problem solving in AI is about representation and automated solution of a wide variety of problems

diagnosis, planning, scheduling, logistics, control
games: sokoban, mastermind, 15-puzzle, n-queens, chess
robot navigation, traveling salesman, map coloring, . . .

Models

Common **structure** of certain classes of problems can be abstracted and expressed in terms of **mathematical model**; e.g.,

- **Constraint Satisfaction Problems (CSP)** are models of the form $\langle X, D, C\rangle$ where X, D, and C are sets of variables, domains, and constraints

- A **solution** to a CSP assigns to each variable a value from its domain such that all constraints satisfied

  Key point:

  – Many problems can be formulated as CSPs
  – If we know how to solve CSPs, we know to solve those problems
  – Same for other models . . .

Example: Linear Equations Model

- John's age is three times Peter's age
- In 10 years, John's age will be twice Peter's age
- How old are John and Peter now?

Formulate problem as **2-linear equations with 2 unknowns:**

$$\begin{aligned} J &= 3P \\ J + 10 &= 2(P + 10) \end{aligned}$$

Solve **model** using **general method**; e.g. variable elimination

$$3P + 10 = 2P + 20$$

Then

$$\begin{aligned} P &= 20 - 10 = 10 \\ J &= 3P = 30 \end{aligned}$$

Problem Solving in AI

- define models of interest
- develop effective methods for solving them

In this tutorial:

-- We'll cover a wide range of models, including State Models, SAT, CSPs, Bayesian Networks, Markov Decision Processes (MDPs), . . .

-- Focus on key principles underlying current solution methods:

- Search Space
- Pruning
- Learning
- Decomposition
- Compilation
- Variable Elimination

Plan for the tutorial

- Introduction (Hector)
  - *Models based on States*
  - *Models based on Variables*
  - *Overview of Techniques*
- Solving models with Search and Inference
  - *State-based Models* (Hector)
  - *Variable-based [Factored or Graphical] Models* (Rina)
- Solving models with Pure Inference and No Search (Adnan)
- Hybrid Methods (Rina)
- Wrap up

More about Tutorial

- assumes basic course in AI
- focuses on principles; not exhaustive (e.g., no approx. methods)
- conceptual but also technical

Please ask questions along the way . . .

Part 1: Introduction

Principles of AI Problem Solving: Introduction

- **Contents**
  - *Models based on States*
  - *Models based on Variables*
  - *Overview of Techniques*

- **Format; Style**
  - general, high-level view of field
  - emphasize intuitions and coherence
  - raise questions that will be addressed in detail later on

Models

- Models define **what** is to be solved
- Algorithms define **how** to solve models

E.g, we understand **what** $\sqrt{43}$ **is** without necessarily knowing **how to compute** value

Same with models: they define the solutions we are looking for, without commitment about their computation

State Models

- Basic State Model characterized by

  - finite and discrete state space S
  - an initial state $s_0 \in S$
  - a set $G \subseteq S$ of goal states
  - actions $A(s) \subseteq A$ applicable in each state $s \in S$
  - a state transition function $f(s, a)$ for $s \in S$ and $a \in A(s)$
  - action costs $c(a, s) > 0$

- A **solution** is a sequence of applicable actions a_i, $i = 0, \ldots, n$, that maps the initial state s_0 into a goal state $s \in S_G$; i.e.,

$$s_{i+1} = f(a_i, s_i) \text{ and } a_i \in A(s_i) \text{ for } i = 0, \ldots, n \text{ and } s_{n+1} \in S_G$$

- **Optimal** solutions minimize total cost $\sum_{i=0}^{i=n} c(a_i, s_i)$

Problems mapping naturally into State Models

- Grid Navigation
- 15-puzzle
- Rubik
- Route Finding in Map
- TSP (Traveling Salesman Problem)
- Jug Puzzles (e.g., 4 & 3 liter jars, have 2 liters in 4 lit. jar)
- ⋮

This is the model underlying **Classical Planning** . . .

Languages

State-Models often represented implicitly in terms of (planning) **languages**

E.g.: **Strips** is a simple language for representing the Basic State Models

- A **problem** in Strips is a tuple $\langle A, O, I, G \rangle$:
  - A stands for set of all **atoms** (boolean vars)
  - O stands for set of all **operators** (actions)
  - $I \subseteq A$ stands for **initial situation**
  - $G \subseteq A$ stands for **goal situation**

- Operators $o \in O$ **represented** by three lists

  -- the **Add** list $Add(o) \subseteq A$
  -- the **Delete** list $Del(o) \subseteq A$
  -- the **Precondition** list $Pre(o) \subseteq A$

Strips: From Language to Model

Strips problem $P = \langle A, O, I, G \rangle$ determines **state model** $\mathcal{S}(P)$ where

- the states $s \in S$ are **collections of atoms**
- the initial state s_0 is I
- the goal states s are such that $G \subseteq s$
- the actions a in $A(s)$ are s.t. $Prec(a) \subseteq s$
- the next state is $s' = s - Del(a) + Add(a)$
- action costs $c(a, s)$ are all 1

The (optimal) **solution** of problem P is the (optimal) **solution** of State Model $\mathcal{S}(P)$

Later on we'll see how **Strips descriptions** can play a **computational role** as well . . .

Model with Incomplete Information and Non-Determinsm

- finite and discrete state space S
- a **set** of possible initial states $S_0 \subseteq S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a **non-deterministic** transition function F s.t. $F(a, s)$ is a **set** of states, $a \in A(s)$
- action costs $c(a, s) > 0$

-- A **solution** is a sequence of actions that lead to S_G for **any** possible initial state and transition

-- An **optimal** solution miminizes the sum of action costs

-- Planning over this class of models called **Conformant Planning**

Model with Non-determinism and Full Feedback

- finite and discrete state space S
- a **set** of possible initial states $S_0 \subseteq S$
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- a **non-deterministic** transition function F ...
- action costs $c(a, s) > 0$
- states **fully observable**

-- **Solutions** become **functions** mapping states into actions (**closed-loop control policies**)

-- This is because dynamics is **Markovian** and past history of system is not relevant

-- **Optimal** solutions minimize cost in **worst case** (**min-max state policies**)

Stochastic Model with Full Feedback (Markov Decision Process - MDP)

- finite and discrete state space S
- a set $S_G \subseteq S$ of goal states
- actions $A(s) \subseteq A$ applicable in each $s \in S$
- **transition probabilities** $P_a(s'|s)$ for s and $a \in A(s)$
- action costs $c(a, s) > 0$
- states **fully observable**

-- **Solutions** like before are **functions** mapping states into actions (**closed-loop control policies**)

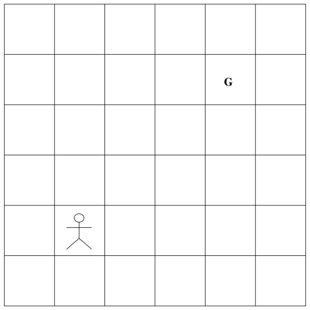
-- **Optimal** solutions minimize **expected** cost

-- This model underlies **Probabilistic Planning**; although variations possible (e..g, partial observability, achieving goal with certain probability, discounted formulations, . . .).

Example: Navigation Problems

Consider robot that has to reach target G when

- *initial state is known and actions are deterministic*
- *initial state is unknown and actions are deterministic*
- *states are fully observable and actions are stochastic*
- *states are partially observable and actions are stochastic . . .*



-- How do these problems map into the models considered?

-- What is the form of the solutions?

Models based on Variables: Factored or Graphical Models

- Constraint Satisfaction Problems (CSP)
- Satisfiability (SAT)
- Bayesian Networks
- Influence Diagrams, . . .

Several **tasks** associated with these models . . .

Constrain Satisfaction Problems (CSPs)

- CSPs are triplets $\langle \text{Variables,Domains,Constraints} \rangle$
- A **solution** assigns values to the variables from their corresponding domains satisfying all constraints
- A CSP is **consistent** if it has one or more solutions

E.g., first CSP below is consistent; second is not

$$\langle \{X, Y\}, \{D_X, D_Y = [1..10]\}, \{X + Y > 10, X - Y > 7\} \rangle$$

$$\langle \{A, B, C\}, \{D_A, D_B, D_C = [a, b]\}, \{A \neq B, B \neq C, A \neq C\} \rangle$$

Problems that Map into CSPs

- Scheduling
- Planning
- Resource allocation
- Map coloring
- N-queens
- $\vdots$

Satisfiability (SAT)

- SAT is special type of CSP where variables x, y, . . . are **boolean** and constraints are **clauses**

$$x \vee \neg y \vee z \ldots$$

- A set of clauses denotes a formula in **Conjunctive Normal Form (CNF)**: a conjunction of disjunctions of **literals**

- Current SAT solvers are very powerful, and used in Planning and Verification

- Any CSP can be mapped into SAT and vice versa, and solving techniques have a lot in common

The Graph Underlying Graphical Models

- SAT and CSP are both NP-Complete, yet complexity bounded by **treewidth of interaction graph**

- The **interaction graph** of a problem is an undirected graph where
  - the vertices are the variables, and
  - two variables are connected iff if they appear in same constraint/clause

- The **treewidth** measures how 'tree-like' is the interaction graph
  - treewidth $= 1$ implies **linear complexity**, while
  - **bounded treewidth** implies **polynomial complexity**

Bayesian Networks (BNs)

BNs are graphical models that express a **joint probability distribution** over a set of variables X_1, . . . , X_n by means of

- a **directed acyclic graph** over the variables
- **conditional probability tables** $P(X_i|pa(X_i))$ of each variable X_i given its parents $pa(X_i)$ in the graph

The joint distribution is the product of the tables:

$$P(X_1, \ldots, X_n) = \prod_{i=1,n} P(X_i|pa(X_i))$$

BNs and CSPs are similar; they specify joint **probability** and joint **consistency** through local factors that define the interaction graph

Models and Tasks: SAT and CSPs

- **Consistency** is basic task in SAT and CSPs: *find a satisfying assignment o that no one exists*

Yet other tasks common:

- **Optimization:** find **best** satisfying assigment according to some **function** (COP)
- **Enumeration:** find **number** of satisfying assignments (**Model Counting**); find **all solutions**, . . .

All tasks are NP-hard; **Consistency** and **(Bounded) Optimization** are NP-Complete while **Enumeration** is #P.

Models and Tasks: Bayesian Networks

- **Enumeration:** find probability given evidence: $P(X = x|Y = y, Z = z, ...)$ (Bel)
- **Optimization:** find most probable instantiation given evidence (MPE)
- Other: find most probable instantiation of **subset** of variables given evidence (MAP)

All tasks are NP-hard; (Bounded) MPE is NP-Complete, and Bel is #P

Map

- Introduction
  - *Models based on States*
  - *Models based on Variables*
  - ▷ *Overview of Techniques*
    - ∗ Search Space
    - ∗ Pruning
    - ∗ Learning
    - ∗ Decomposition
    - ∗ Compilation
    - ∗ Variable Elimination
- Solving models with Search and Inference
  - *State-based Models*
  - *Variable-based [Factored or Graphical] Models*
- Solving models with Pure Inference and No Search
- Hybrid Methods

Search

- Basic tasks can be formulated as **search problem** in suitable **problem space**

- Problem or Search space is a Directed Graph given by
  - root node n_0 of the search
  - set of terminal nodes; either dead-ends or goals
  - branching rule generating children n' of non-terminal nodes n
  - costs $c(n, n') \geq 0$

- A solution is a directed path that connects the root node with a goal node. It is optimal if it minimizes the sum of the edge costs.

Direct Problem Space for Basic State Models

The **nodes** correspond to the **states** and

- *root node is initial state* s_0
- *goal nodes are the goal states*
- *dead ends are states* s *s.t no action applies in* s
- *branching rule:* $s \rightarrow s'$ *if* $s' = f(a, s)$ *for some* a *applicable in* s
- *cost is then* $c(s, s') = c(a, s)$

In spite of direct mapping from Basic State Model to Search Graph, it's good to keep in mind that first is a description of the **problem**, while second is the structure explored for finding a **solution**

When the State Model is described in **Strips**, this is the so-called **progression space**, as alternative spaces are possible . . .

Alternative Problem Spaces from Strips Encodings

- **regression space:** branch by applying actions backward from goal til finding conditions that hold in initial state

- **plan space:** branch by refining partial plan, removing its flaws

In certain cases, these alternative branching schemas/problem spaces more suitable (e.g., plan space seems best for optimal temporal planning)

Strips problems with fixed planning horizon can also be mapped into SAT, which works very well when **optimal parallel plans** are sought

Problem Space for Non-Deterministic State models

Conformant Planning can be formulated as Search Problem over **belief space**, where nodes are **belief states**, i.e., sets of states deemed possible

- *root node is set of possible initial states*
- *goal nodes are sets of goal states*
- *edge* $n \rightarrow n'$ *if for some action,* n' *is the set of states that may follow the states in* n
- . . .

-- This is most common formulation for Conformant Planning currently

-- Belief states represented often by **propositional formula** in suitable **'compiled'** form (e.g., OBDDs, d-DNNF, . . . ; more about this later on)

Problem Space for Graphical Models: OR Space

Consistency and **Optimization Problems** for SAT, CSP, and Bayesian Networks are formulated as **search problems** over suitable Search Graph.

In the standard formulations, the nodes are **partial assignments**:

- *root node is empty assignment*
- *dead-ends are partial assignments that violate a constraint*
- *'goal' nodes are complete assignments*
- *children n' of node n obtained* **by picking up unassigned variable in n, and assigning it a value**
- *costs $c(n, n')$ uniform in consistency problems, and dependent on local functions in COP and BNets.*

-- Choice of **branching variable** affects size of Search Tree and critical for performance

Problem Space for Graphical Models: AND/OR Space

- Solution to **enumeration problems**, like **model counting** over CNFs and **belief updating** over BNets, do not correspond to solution **paths** in graph but can be computed from it

- We will also see an **alternative formulation** of all these tasks in terms of **AND/OR Graphs** rather than (OR) Graphs that exploit **decomposition**

- This AND/OR space is (almost) explicit in some algorithms (e.g., Recursive Conditioning) and implicit in others (e.g., non-chronological backtracking).

Map

- Introduction
  - *Models based on States*
  - *Models based on Variables*
  - *Overview of Techniques*
    - ∗ Search Space
    - ▷ Pruning
    - ∗ Learning
    - ∗ Decomposition
    - ∗ Compilation
    - ∗ Variable Elimination
- Solving models with Search and Inference
  - *State-based Models*
  - *Variable-based [Factored or Graphical] Models*
- Solving models with Pure Inference and No Search
- Hybrid Methods

Pruning in Depth-First Search

- Search graph can be solved by Depth-First Search (DFS) and variations
- More effective DFS obtained by **pruning** nodes that cannot lead to acceptable solutions; e.g.,

  Consistency: *prune node n if it can only lead to dead-ends*

  State Models/Optimization: *prune node n if it can only lead to solutions with cost $>$ than given $Bound$*

- By playing with $Bound$ one can get Bounded DFS, IDA*, DFS Branch & Bound
- **Key issue:** how to **predict** when paths up to node n
  - can only lead to dead-ends? [consistency]
  - can only lead to solutions with cost $> Bound$? [optimization]

Pruning (2)

- Pruning criterion has to be **sound** and **cost-effective**
- Two ideas: **lower bounds** (LBs) and **constraint propagation** (CP)
  - **LBs:** *prune* n *if* $f(n) > Bound$ *where* $f(n)$ *is LB of cost of best solution that extends* n
  - **CP:** *prune value* x *from variable* X *domain, if* $X = x$ **proved inconsistent** *with constraints and commitments in* n*; prune node* n *itself if some domain becomes empty*
- LBs and CP mechanisms can both be obtained as **inference in relaxed model**; e.g.
  - in Strips: forget 'deletes' and assume (relaxed) actions can be done in parallel ('simple reachability heuristic')
  - in CSPs: e.g., solve each constraint in isolation ('arc consistency')

We will say more about LBs and CP mechanisms . . .

Learning during Search (State Models)

- Results of (partial) search can be used to improve pruning in rest of the search

- E.g., if node shown not to lead to solution found again in the search, it can be pruned right away

- However one can do better; e.g., in IDA*, for example, right after all sons n' of node n return without a solution (for given $Bound$), **heuristic value** $h(n)$ can be **increased** to:

$$h(n) := min_{n':n \rightarrow n'} \ c(n, n') + h(n')$$

- Resulting algorithm known as IDA* + **Transposition Tables**

- Exactly same update rule used in **Learning Real Time A* (LRTA*)**

- Actually **updates** can be done without search at all and they eventually yield h^*! This is what **Value Iteration** does, which also applies to MDPs

Learning during Search (Factored Models)

- For consistency tasks (SAT, CSP), one can actually do even better
- Rather than **updating** the **value** of a node, update the **theory** itself!
- Use structure for identifying and ruling out **cause of the inconsistency**
- This is the idea of **no-good learning** in SAT and CSPs
- Learned information
  - applies to other nodes as well
  - results in non-chronological backtracking
  - enables further inferences and pruning
- It is a key idea in current SAT solvers

Decomposition

- Consider solving a SAT problem T made up of two independent sub-problems T' and T'' with n variables each, none in common

- By **decomposing** the problem in two as

$$SAT(T) = SAT(T') \And SAT(T'')$$

  worst case complexity is reduced from $2^n * 2^n$ to $2^n + 2^n$

- Interestingly, if T' and T'' overlap over single variable X, T can still be decomposed by **conditioning** on X as

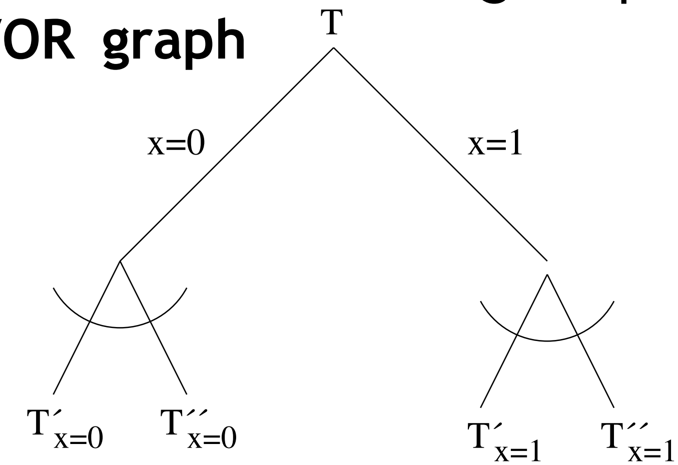
$$SAT(T) = \bigvee_x SAT(T'_{X=x}) \And SAT(T''_{X=x})$$

  where $T_{X=x}$ means T with variable X replaced by value x.

- This idea can be applied recursively, even if T' and T'' overlap over **set** of variables

Decomposition and AND/OR Search Graph

- Decomposition by **recursive conditioning** maps search over **OR-graph** into search over **AND/OR graph**



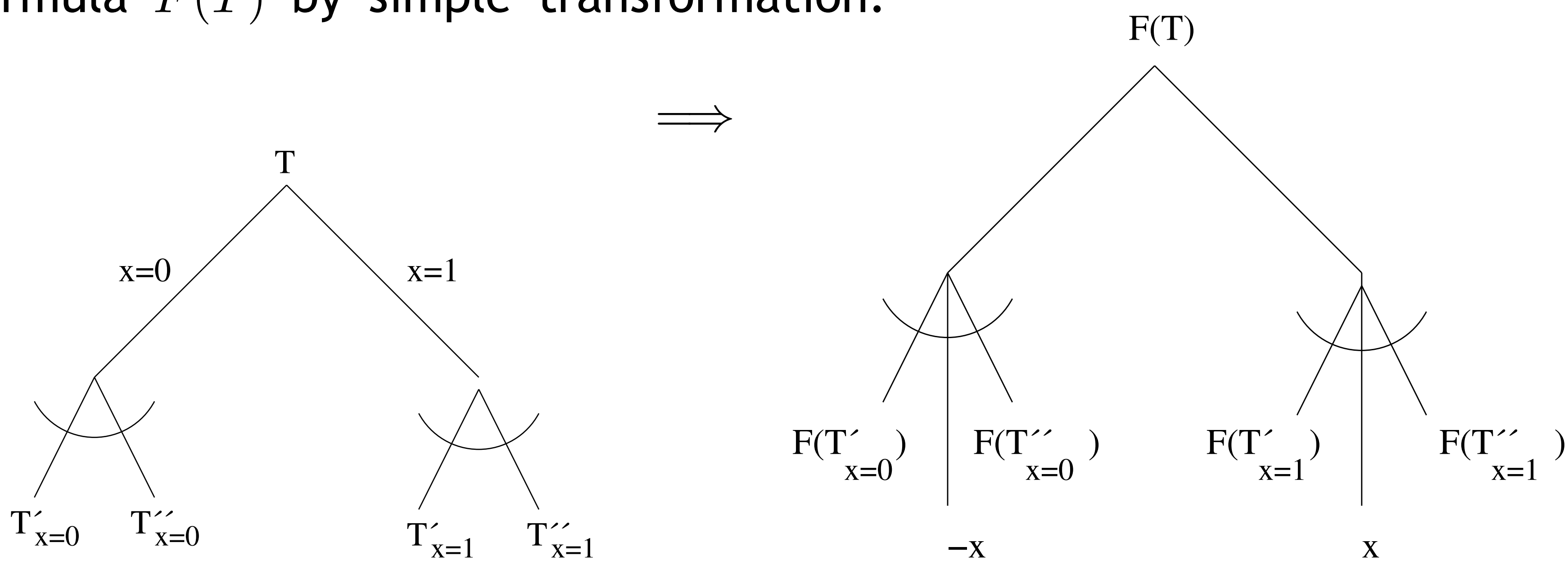
- By suitable choice of decompositions and caching, worst-case complexity can be reduced from $O(Exp(n))$ to $O(Exp(w^*))$, where $w^* \leq n$ is theory **treewidth** (e.g., linear for trees)

- Similar decomposition methods can be used (and are used!) for enumeration tasks like **Model Counting** (MC) and **Belief Update** but with different agregation operators; e.g.,

$$MC(T) = \sum_{x} MC(T'_{X=x}) \ * \ MC(T''_{X=x})$$

From Decomposition to Knowledge Compilation (1)

- Suitable 'trace' of AND/OR Search can be used to perform a large class of **intractable boolean operations** in time linear in size of 'trace'
- Indeed, just map AND/OR Graph for P into **logically equivalent** AND/OR formula $F(T)$ by simple transformation:



- AND/OR formula $F(T)$ is in Deterministic Decomposable Negation Normal Form (d-DNNF): **disjuncts** are exclusive, **conjuncts** share no variables, and **negations** affect vars only; closely related to **OBDDs**

Map

- Introduction
  - *Models based on States*
  - *Models based on Variables*
  - *Overview of Techniques*
    - ∗ Search Space
    - ∗ Pruning
    - ∗ Learning
    - ∗ Decomposition
    - ∗ Compilation
    - ▷ Variable Elimination
- Solving models with Search and Inference
  - *State-based Models*
  - *Variable-based [Factored or Graphical] Models*
- Solving models with Pure Inference and No Search
- Hybrid Methods

Variable Elimination

- Gaussian Elimination used to solve n linear equations T_n with n unknowns $X_1, \ldots, X_n$
  - *Eliminate X_n obtaining $n-1$ equations T_{n-1} with $n-1$ unknowns*
  - *Iterate til obtaining 1 equation T_1 with 1 unknown (X_1)*
  - *Solve T_1 for X_1 and plug result into T_2*
  - *Solve T_2 for X_2 and plug result x_2 for X_2 into T_3, etc*
- Method can be be generalized to **graphical models**; only change is way for **eliminating variables**; e.g.,
  - *in CNF, X_i eliminated by* **resolving upon** X_i
  - *in CSPs, X_i eliminated by* **join** *and* **project** *DB operations*
  - *in Belief Update (BNets), X_i eliminated by* **sum** *and* **products**, . . .

Variable Elimination, Inference, and AND/OR Search

- Variable Elimination solves problems by **inference and no search**
- Yet same complexity bounds ($O(Exp(w^*))$) as Decomposition Methods with Caching that search over AND/OR graphs, and furthermore . . .
- Variable Elimination can be understood as **bottom up search of same AND/OR graph!**

Few powerful ideas that span a large terrain and have a lot of connections and ramifications. This is what the tutorial is about . . .

Map

- Introduction (Hector)
  - *Models based on States*
  - *Models based on Variables*
  - *Overview of Techniques*
    - ∗ Search Space
    - ∗ Pruning
    - ∗ Learning
    - ∗ Decomposition
    - ∗ Compilation
    - ∗ Variable Elimination
- Solving models with Search and Inference
  - ▷ *State-based Models* (Hector)
  - *Variable-based [Factored or Graphical] Models* (Rina)
- Solving models with Pure Inference and No Search (Adnan)
- Hybrid Methods (Rina)

Part 2: Search and Inference

Techniques for Solving State Models: Focus

- **Models**
  - *Basic State Models: Complete Knowledge, Deterministic Actions*
  - *Markov Decision Processes: Stochastic Actions and Full Feedback*
- **Techniques**
  - *Problem Space: Branching Schemes*
  - *Pruning: Admissible heuristics or LBs* $h(s) \leq h^*(s)$
  - *Learning: Improving* $h(s)$ *while Searching*
- **Language**
  - *We assume* **Models** *specified in a* **Strips**-*like language*
  - *This takes us into what is called* **Planning** *in AI*

State Models Reminder

- Basic State Model characterized by
  - finite and discrete state space S
  - an initial state $s_0 \in S$
  - a set $G \subseteq S$ of goal states
  - actions $A(s) \subseteq A$ applicable in each state $s \in S$
  - a state transition function $f(s, a)$ for $s \in S$ and $a \in A(s)$
  - action costs $c(a, s) > 0$

- A **solution** is a sequence of applicable actions that map initial state s_0 into goal state

- **Optimal** solutions minimize total cost . . .

Strips Reminder

- A **problem** in Strips is a tuple $\langle A, O, I, G \rangle$:
  - A stands for set of all **atoms** (boolean vars)
  - O stands for set of all **operators** (actions)
  - $I \subseteq A$ stands for **initial situation**
  - $G \subseteq A$ stands for **goal situation**

- Operators $o \in O$ **represented** by three lists

  -- the **Add** list $Add(o) \subseteq A$
  -- the **Delete** list $Del(o) \subseteq A$
  -- the **Precondition** list $Pre(o) \subseteq A$

Strips: From Language to Model

Strips problem $P = \langle A, O, I, G \rangle$ determines **state model** $\mathcal{S}(P)$ where

- the states $s \in S$ are **collections of atoms**
- the initial state s_0 is I
- the goal states s are such that $G \subseteq s$
- the actions a in $A(s)$ are s.t. $Prec(a) \subseteq s$
- the next state is $s' = s - Del(a) + Add(a)$
- action costs $c(a, s)$ are all 1

The (optimal) **solution** of problem P is the (optimal) **solution** of State Model $\mathcal{S}(P)$

Pruning: Getting Lower Bounds for Strips Problems

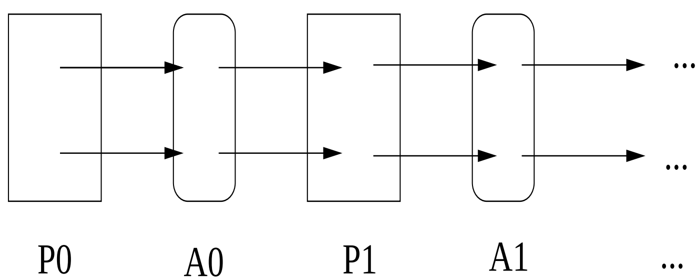
Admissible Heuristics (LBs) for Strips obtained by solving **relaxed models**

- ignore delete-lists
- ignore certain atoms
- decompose goals sets into smaller subsets
  - e.g., assume cost of achieving set given by cost of achieving most costly **pair** in the set . . .

Lower Bounds for Strips: Reachability Graph

Ignore deletes and apply all actions in parallel:

$$\begin{aligned} P_0 &= \{p \in s\} \\ A_i &= \{a \in O \mid Prec(a) \subseteq P_i\} \\ P_{i+1} &= \{p \in Add(a) \mid a \in A_i\} \end{aligned}$$



Define then **admissible heuristic**

$$h^1_G(s) \stackrel{\text{def}}{=} \text{min } i \text{ such that } G \subseteq P_i$$

Need No-op(p) action for each p: $Prec = Add = \{p\}$

Planning Graph = Reachability Graph + Mutexes

- **Better relaxation:** assume **no deletes** and that all actions can be done in parallel **except** certain **incompatible action pairs**

- This relaxation not tractable but good LB approximation exists, in particular for **parallel planning** (where only diff is that deletes are not ignored)

  – **action** pair **mutex** at i if incompatible or preconditions **mutex** at i
  – **atom** pair **mutex** at $i+1$ if all supporting action pairs **mutex** at i

- Mutex x, y at i implies that no valid plan can have both x and y at i but not the converse

- Define then more informed **admissible heuristic** $h^2_G(s)$ as:

$$h^2_G(s) \stackrel{\text{def}}{=} \min\ i \text{ such that } G \subseteq P_i \ \& \text{ not mutex at } i$$

- Graph and resulting heuristic $h^2_G(s)$ computed for one state s only, but valid for **any** goal G . . .

How to use Strips Heuristics? Problem Spaces in Planning

- **Option 1: Progression Space:** search forward from s_0
  - *recompute graph and $h^2_G(s)$ for every s; this is costly*
- **Option 2: Regression Space:** search backward from Goal (Graphplan)
  - *no need to recompute graph which encodes $h^2_{G'}(s)$ for all goals G'! but high branching factor when lots of parallel actions*
- **Option 3: Action Space**: non-directional search
  - *Branch by picking an action a and time point i and trying the two possibilities: a in the plan at i ; a not in the plan at i*
  - *At each node n* **recompute planning graph** *from s_0* **respecting commitments in** n, *and* **prune** n *if Goals pushed beyond horizon*

Current State of the Art in Planning

No single best **Problem Space** or **Pruning Criterion** for all types of planning tasks:

- **Sequential (Classical) Planning: Heuristic Search** in Progression Space currently best with both admissible and non-admissible h's
- **Optimal Parallel Planning: SAT formulation** fed to state-of-the-art solvers (Siege) currenty best
- **Optimal Temporal Planning:** Search in **Plan Space (POCL)** best with pruning scheme based on **Constraint Propagation**

Learning while Searching in State Models

A number of algorithms combine **search** with **state value updates**:

- Learning Real time A* (LRTA*)
- IDA* + Memory (Tranposition Tables)
- Real Time Dynamic Programming (RTDP)
- MTD (algorithm for Game Trees better than Alpha-Beta)
- . . .

Other algorithms do **updates with no search**

- Value Iteration

Understanding Value Updates: Dynamic Programming

- Solutions to wide range of models can be expressed in terms of solution of so-called **Bellman equation:**

$$V(s) = min_{a \in A(s)} \; Q_V(a, s)$$

where cost-to-go term $Q_V(a, s)$ depends on model ($F(a, s)$: next states)

$c(a, s) + V(s'), \; s' \in F(a, s)$	for OR Graphs
$c(a, s) + \max_{s' \in F(a,s)} V(s')$	for Max AND/OR Graphs
$c(a, s) + \sum_{s' \in F(a,s)} V(s')$	for Additive AND/OR Graphs
$c(a, s) + \sum_{s' \in F(a,s)} P_a(s'\|s)V(s')$	for MDPs
$\max_{s' \in F(a,s)} V(s')$	for Game Trees

- The **greedy policy** π_V is **optimal** when $V = V^*$ solves Bellman

$$\pi_V(s) = \operatorname{argmin}_{a \in A(s)} \; Q_V(a, s)$$

- **Question:** how to get V^*?

Updates with No Search: Value Iteration

- Value Iteration finds V^* by successive approximations

- Starting with an arbitrary V, uses Bellman equation to update V; e.g. for Basic State Models (OR Graphs)

$$V(s) := min_{a \in A(s)} \; [c(a,s) + V(s_a)]$$

- As long as **all states updated** sufficiently often (and certain general conditions hold), left and right hand sides converge, and $V = V^*$

- VI is simple and general but also **exhaustive**

- Can the updates be restricted to **subset of states** preserving optimality?

- Yes: like in Heuristic Search, use **Lower Bounds** and **Initial State**

Focusing Value Iteration using LBs and Initial State

- Say that a state s is
  - **inconsistent** if $V(s) < \min_{a \in A(s)} Q_V(a, s)$, and
  - **greedy** if reachable from s_0 using greedy policy π_V
- Then starting with an **admissible**, follow loop:
  - **Find** *an* **inconsistent greedy** *state* s *and* **Update** *it*
- Loops delivers greedy policy that is **optimal even if some states not updated or visited at all!**
- Unlike DP method, both LBs ($V \leq V^*$) and Initial State (s_0) used

Learning During Search in State Models

- Convergence of all Learning Algorithms in State Models (LRTA*, MTD, IDA*+TT, . . .) can be understood in these terms: *update an inconsistent greedy state in all iterations til no more such states*
- Speed up obtained by updating **multiple** such states in every iteration
- This can be done by implementing **Find** as a **DFS** over greedy states with inconsistent states as terminals (*Learning in Depth-First Search*)
- This is what **IDA*** $+$ **Trans. Tables** (Basic State Models) and **MTD** (Game Trees) actually do
- Same idea underlies current **heuristic-search methods for solving MDPs:** LAO*, RTDP, HDP, . . .

Bibliography: Techniques for Solving State Models Part

- Heuristics for Strips Planning: [23, 6, 16, 14, 30]; admissible heuristics [13, 9, 12]; heuristics and the planning graph [6, 13, 24]; Graphplan [4].
- Planners searching in Regression Space [4, 5, 13].
- Planners searching in 'Plan Space': [22, 18, 32].
- Planners searching in (non-directional) 'Action Space': SAT-formulation and CSP formulations like [19, 28] (and in particular [10]), and [15].
- Temporal Planning: [21, 25, 17], Optimal Temporal Planning [31], Planning and Scheduling [29].
- Learning while Searching in State Models: IDA* with Transposition Tables [27], LRTA* [20], RTDP [1], MTD for Game Trees [26]; general LDFS framework [8].
- Value Iteration and Dynamic Programming [2, 3].
- Focusing Value Iteration: General Find-and-Revise procedure [7, 8]
- Learning in DFS [8], Heuristic Search Algorithms for MDPs [1, 11, 7]

References

[1] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81--138, 1995.

[2] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[3] D. Bertsekas. *Dynamic Programming and Optimal Control, Vols 1 and 2*. Athena Scientific, 1995.

[4] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, pages 1636--1642. Morgan Kaufmann, 1995.

[5] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proceedings of ECP-99*, pages 359--371. Springer, 1999.

[6] B. Bonet and H. Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1--2):5--33, 2001.

[7] B. Bonet and H. Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proc. IJCAI-03*, pages 1233--1238, 2003.

[8] B. Bonet and H. Geffner. Learning in DFS: A unified approach to heuristic search in deterministic, non-deterministic, probabilistic, and game tree settings. 2005.

[9] S. Edelkamp. Planning with pattern databases. In *Proc. ECP 2001*, 2001.

[10] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In *Proc. AAAI-98*, pages 948--953, 1998.

[11] E. Hansen and S. Zilberstein. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35--62, 2001.

[12] P. Haslum, B. Bonet, and H. Geffner. New admissible heuristics for optimal planning. In *Proc. AAAI-05*, 2005. To appear.

[13] P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Proc. of the Fifth International Conference on AI Planning Systems (AIPS-2000)*, pages 70--82, 2000.

[14] M. Helmert. A planning heuristic based on causal graph analysis. In *Proc. ICAPS-04*, pages 161--170, 2004.

[15] J. Hoffmann and H. Geffner. Branching matters: Alternative branching in graphplan. In E. Giunchiglia, N. Muscettolla, and D. Nau, editors, *Proc. 13th Int. Conf. on Automated Planning and Scheduling (ICAPS-2003)*, pages 22--31. AAAI Press, 2003.

[16] J. Hoffmann and B. Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253--302, 2001.

[17] A. Jonsson, P. Morris, N. Muscettola, and K. Rajan. Planning in interplanetary space: Theory and practice. In *Proc. AIPS-2000*, pages 177--186, 2000.

[18] S. Kambhampati, C. Knoblock, and Q. Yang. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence*, 76(1-2):167--238, 1995.

[19] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of AAAI-96*, pages 1194--1201. AAAI Press / MIT Press, 1996.

[20] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189--211, 1990.

[21] P. Laborie and M. Ghallab. Planning with sharable resources constraints. In C. Mellish, editor, *Proc. IJCAI-95*, pages 1643--1649. Morgan Kaufmann, 1995.

[22] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of AAAI-91*, pages 634--639, Anaheim, CA, 1991. AAAI Press.

[23] D. McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence*, 109(1-2):111--159, 1999.

[24] X. Nguyen, S. Kambhampati, and R. Sanchez Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence*, 135(1-2):73--123, 2002.

[25] J. Penberthy and D. Weld. Temporal planning with continous change. In *Proc. AAAI-94*, pages 1010--1015, 1994.

[26] A. Plaat, J. Schaeffer, W. Pijls, and A.de Bruin. Best-first fixed-depth minimax algorithms. *Artificial Intelligence*, 87(1-2):255--293, 1996.

[27] A. Reinefeld and T. Marsland. Enhanced iterative-deepening search. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(7):701--710, 1994.

[28] J. Rintanen. A planning algorithm not based on directional search. In *Proceedings KR'98*, pages 617--624. Morgan Kaufmann, 1998.

[29] D. Smith, J. Frank, and A. Jonsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1), 2000.

[30] V. Vidal. A lookahead strategy for heuristic search planning. In *Proc. ICAPS-04*, pages 150--159, 2004.

[31] V. Vidal and H. Geffner. Branching and pruning: An optimal temporal POCL planner based on constraint programming. In D. McGuiness and G. Ferguson, editors, *Proceedings of 19th Nat. Conf. on Artificial Intelligence (AAAI-04)*, pages 570--577. AAAI Press/MIT Press, 2004.

[32] Daniel S. Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27--61, 1994.