AI基础

Lecture 9: Knowledge Representation, Automated Planning, and Quantifying Uncertainty

Bin Yang

School of Data Science and Engineering

byang@dase.ecnu.edu.cn

[Some slides adapted from Philipp Koehn, JHU]

$$\forall s \;\; Breezy(s) \Leftrightarrow \exists r \;\; Adjacent(r,s) \wedge Pit(r)\,.$$

$$R_2 : \quad B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$
$$R_3 : \quad B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2}$$
$$\vdots$$

Lecture 8 ILOs

- Limitations of propositional logic
- First-order logic (FOL)
- Syntax and Semantics
  - Models, symbols, interpretations, terms
  - Atomic and complex sentences
  - Quantifiers, universal and existential quantifiers, equality
  - Database semantics
- Inference
  - Reduce FOL to propositional logic, instantiation and propositionalization
  - Unification
  - Forward chaining, backward chaining, resolution
- Knowledge representation
  - Ontological engineering
  - Categories of objects, substances, and measures

$$\begin{aligned}
\textit{Sentence} &\rightarrow \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} &\rightarrow \textit{Predicate} \mid \textit{Predicate}(\textit{Term},\ldots) \mid \textit{Term} = \textit{Term} \\
\textit{ComplexSentence} &\rightarrow (\textit{Sentence}) \\
&\mid \neg \textit{Sentence} \\
&\mid \textit{Sentence} \wedge \textit{Sentence} \\
&\mid \textit{Sentence} \vee \textit{Sentence} \\
&\mid \textit{Sentence} \Rightarrow \textit{Sentence} \\
&\mid \textit{Sentence} \Leftrightarrow \textit{Sentence} \\
&\mid \textit{Quantifier Variable},\ldots \textit{Sentence} \\
\textit{Term} &\rightarrow \textit{Function}(\textit{Term},\ldots) \\
&\mid \textit{Constant} \\
&\mid \textit{Variable} \\
\textit{Quantifier} &\rightarrow \forall \mid \exists \\
\textit{Constant} &\rightarrow A \mid X_1 \mid \textit{John} \mid \cdots \\
\textit{Variable} &\rightarrow a \mid x \mid s \mid \cdots \\
\textit{Predicate} &\rightarrow \textit{True} \mid \textit{False} \mid \textit{After} \mid \textit{Loves} \mid \textit{Raining} \mid \cdots \\
\textit{Function} &\rightarrow \textit{Mother} \mid \textit{LeftLeg} \mid \cdots \\
\text{OPERATOR PRECEDENCE} &: \neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{aligned}$$

The syntax of first-order logic with equality, specified in Backus–Naur form (see page 1030 if you are not familiar with this notation). Operator precedences are specified, from highest to lowest. The precedence of quantifiers is such that a quantifier holds over everything to the right of it.

$$\text{UNIFY}(p,q) = \theta \text{ where } \text{SUBST}(\theta,p) = \text{SUBST}(\theta,q).$$

Lecture 9 ILOs

- Knowledge representation
  - Events
  - Reasoning systems for categories
    - Semantic networks
    - Description logic
- Automated planning
  - Classic planning and Planning Domain Definition Lagrange (PDDL)
  - Algorithms for classic planning
    - Forward search (progression) and backward search (regression)
    - Heuristics: adding shortcut edges, reducing states
  - Hierarchical planning
  - Planning in nondeterministic domains
  - Scheduling
- Quantifying uncertainty
  - Acting under uncertainty
  - Probabilities

Outline

- Knowledge representation
  - Events
  - Reasoning systems for categories: Semantic networks, Description logic
- Automated planning
  - Classic planning and Planning Domain Definition Lagrange (PDDL)
  - Forward search (progression) and backward search (regression)
  - Heuristics: adding shortcut edges, reducing states
  - Hierarchical planning
  - Planning in nondeterministic domains
  - Scheduling
- Quantifying uncertainty
  - Acting under uncertainty
  - Probabilities

Events

- So far, facts were treated as true independent of time
- Events: need to describe what is true and when something is happening.
- Event calculus
  - Event, fluents, and time points
- For instance: Flying event

$$E_1 \in Flyings \wedge Flyer(E_1, Shankar) \wedge Origin(E_1, SF) \wedge Destination(E_1, DC) .$$

  - Flying is the category of all flying events.
  - Reifying events makes it possible to add arbitrary information as attributes
- The event may or may not be ongoing during a specific time
- In general, facts that are true only at specific time points are called fluents, e.g., At(Shankar, Berkeley)

Predicates of events

$T(f, t_1, t_2)$	Fluent f is true for all times between t_1 and t_2
$Happens(e, t_1, t_2)$	Event e starts at time t_1 and ends at t_2
$Initiates(e, f, t)$	Event e causes fluent f to become true at time t
$Terminates(e, f, t)$	Event e causes fluent f to cease to be true at time t
$Initiated(f, t_1, t_2)$	Fluent f become true at some point between t_1 and t_2
$Terminated(f, t_1, t_2)$	Fluent f cease to be true at some point between t_1 and t_2
$t_1 < t_2$	Time point t_1 occurs before time t_2

- Examples
  - T(At(Shankar, Berkeley), t_1, t_2)
  - Happens(E_1, t_1, t_2)
- Describe the effects of a flying event

$$E = Flyings(a, here, there)\ and\ Happens(E, t_1, t_2) \Rightarrow$$
$$Terminates(E, At(a, here), t_1) \wedge Initiates(E, At(a, there), t_2)$$

Time

- Time points and time intervals
  - Two kinds of time intervals: moments and extended intervals
  - Only movements have zero duration

$$Partition(\{Moments, ExtendedIntervals\}, Intervals)$$
$$i \in Moments \;\Leftrightarrow\; Duration(i) = Seconds(0)\,.$$

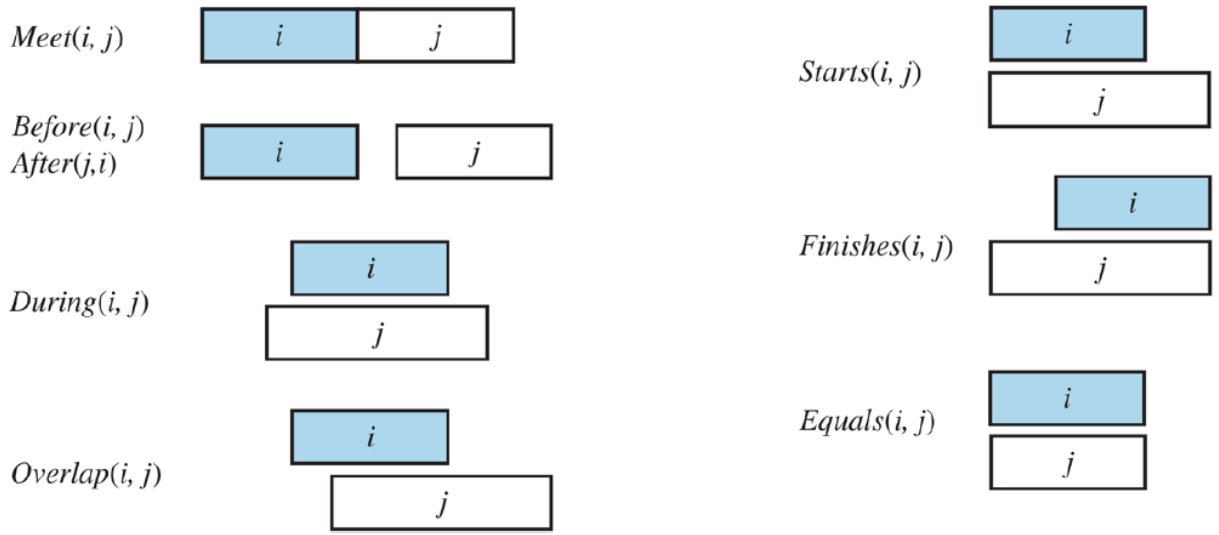
- Absolute time
  - We invent a time scale and associate points on that scale with moments, giving us absolute times.

$$Interval\,(i) \Rightarrow Duration\,(i) = (Time\,(End\,(i)) - Time\,(Begin\,(i)))\,,$$
$$Time\,(Begin\,(AD1900)) = Seconds\,(0)\,.$$
$$Time\,(Begin\,(AD2001)) = Seconds\,(3187324800)\,.$$
$$Time\,(End\,(AD2001)) = Seconds\,(3218860800)\,.$$
$$Duration\,(AD2001) = Seconds\,(31536000)\,.$$

Interval relations

$$
\begin{aligned}
Meet\,(i,j) &\Leftrightarrow End\,(i) = Begin\,(j)\\
Before\,(i,j) &\Leftrightarrow End\,(i) < Begin\,(j)\\
After\,(j,i) &\Leftrightarrow Before\,(i,j)\\
During\,(i,j) &\Leftrightarrow Begin\,(j) < Begin\,(i) < End\,(i) < End\,(j)\\
Overlap\,(i,j) &\Leftrightarrow Begin\,(i) < Begin\,(j) < End\,(i) < End\,(j)\\
Starts\,(i,j) &\Leftrightarrow Begin\,(i) = Begin\,(j)\\
Finishes\,(i,j) &\Leftrightarrow End\,(i) = End\,(j)\\
Equals\,(i,j) &\Leftrightarrow Begin\,(i) = Begin\,(j) \wedge End\,(i) = End\,(j)
\end{aligned}
$$

Figure 10.2



Predicates on time intervals.

Outline

- Knowledge representation
  - Events
  - Reasoning systems for categories: Semantic networks, Description logic
- Automated planning
  - Classic planning and Planning Domain Definition Lagrange (PDDL)
  - Forward search (progression) and backward search (regression)
  - Heuristics: adding shortcut edges, reducing states
  - Hierarchical planning
  - Planning in nondeterministic domains
  - Scheduling
- Quantifying uncertainty
  - Acting under uncertainty
  - Probabilities

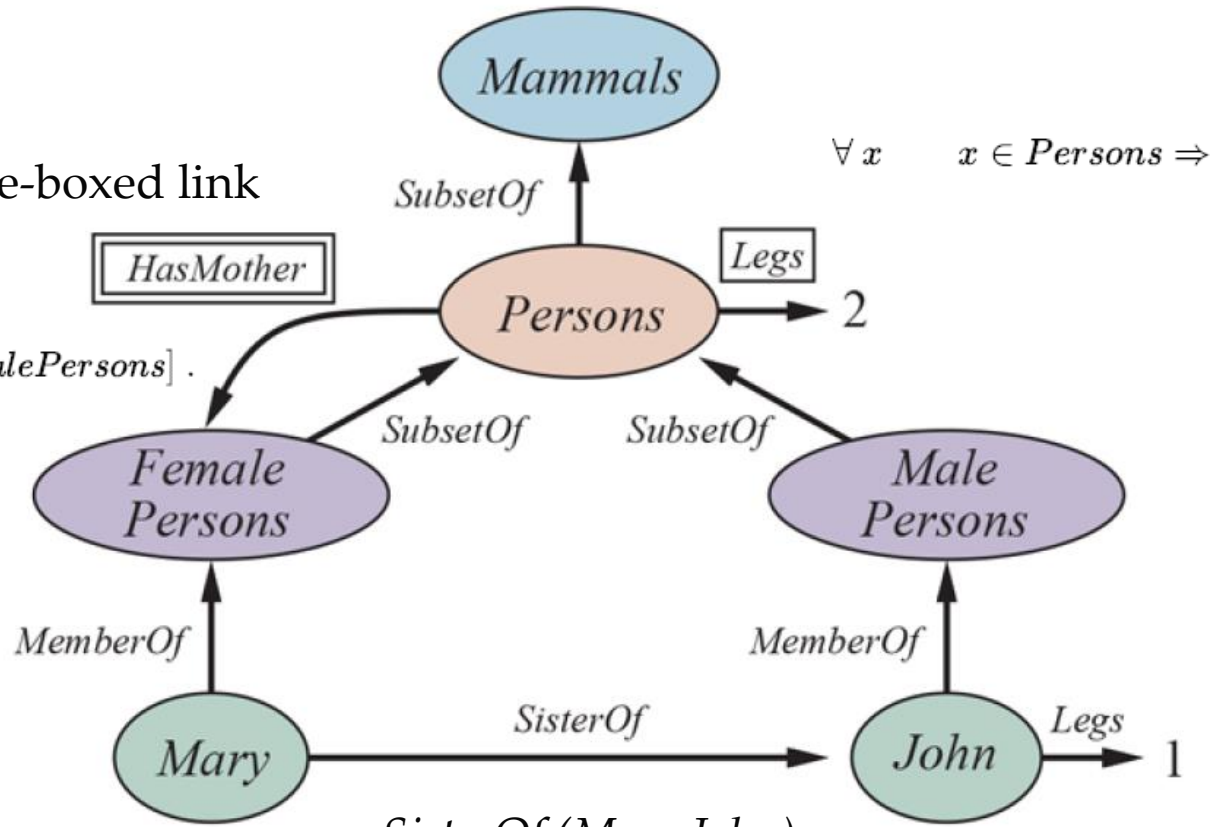
Reasoning systems for categories

- Categories are the primary building blocks of large-scale knowledge representation schemes.
- Systems specially designed for organizing and reasoning with categories. There are two closely related families of systems:
  - **Semantic networks:** provide graphical aids for visualizing a knowledge base and efficient algorithms for inferring properties of an object on the basis of its category membership.
  - **Description logics:** provide a formal language for constructing and combining category definitions and efficient algorithms for deciding subset and superset relationships between categories.

Semantic networks

- There are many variants of semantic networks, but all are capable of representing
  - individual objects, categories of objects, and relations among objects.
- A typical graphical notation displays object or category names in ovals or boxes, and connects them with labeled links.

Figure 10.4



A semantic network with four objects (John, Mary, 1, and 2) and four categories. Relations are denoted by labeled links.

double-boxed link

$\forall x \quad x \in Persons \Rightarrow Legs(x, 2)$.

$\forall x \quad x \in Persons \Rightarrow [\forall y \quad HasMother(x, y) \Rightarrow y \in FemalePersons]$.

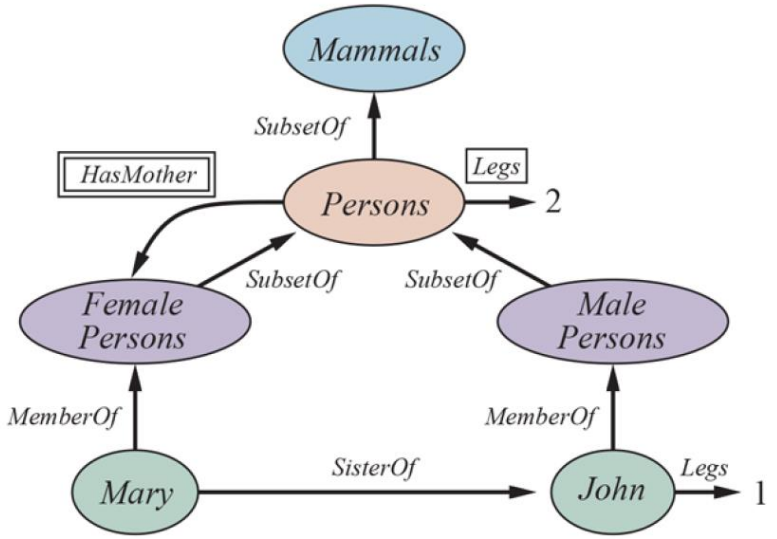
Mary ∈ FemalePersons

SisterOf (Mary,John).

Inheritance in semantic networks

- The semantic network notation makes it convenient to perform inheritance reasoning.
  - Mary inherits the property of having two legs from Persons.
  - Simplicity and efficiency of the **inheritance** reasoning is one of the main advantageous of semantic networks.
- Multiple inheritance
  - A category can be a subset of more than one other category.
  - May have conflicting values.
  - Often banned in object-oriented programming (OOP). But often allowed in semantic networks.

Figure 10.4



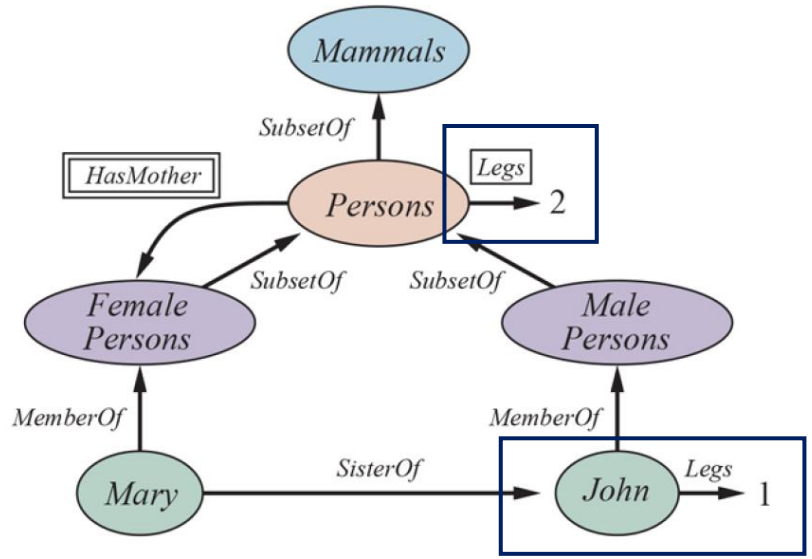
A semantic network with four objects (John, Mary, 1, and 2) and four categories. Relations are denoted by labeled links.

Default values

- One of the most important aspects of semantic networks is their ability to represent default values for categories.
- John has 1 leg but all persons should have two legs.
- In a strictly logical KB, this would be a contradiction.
- In a semantic network, the assertion that all persons have two legs has only default status; that is, a person is assumed to have two legs unless this is contradicted by more specific information.
- Default is **overriden**

$$\forall\, x \qquad x \in Persons \wedge x \neq John \Rightarrow Legs(x, 2)\ .$$

Figure 10.4

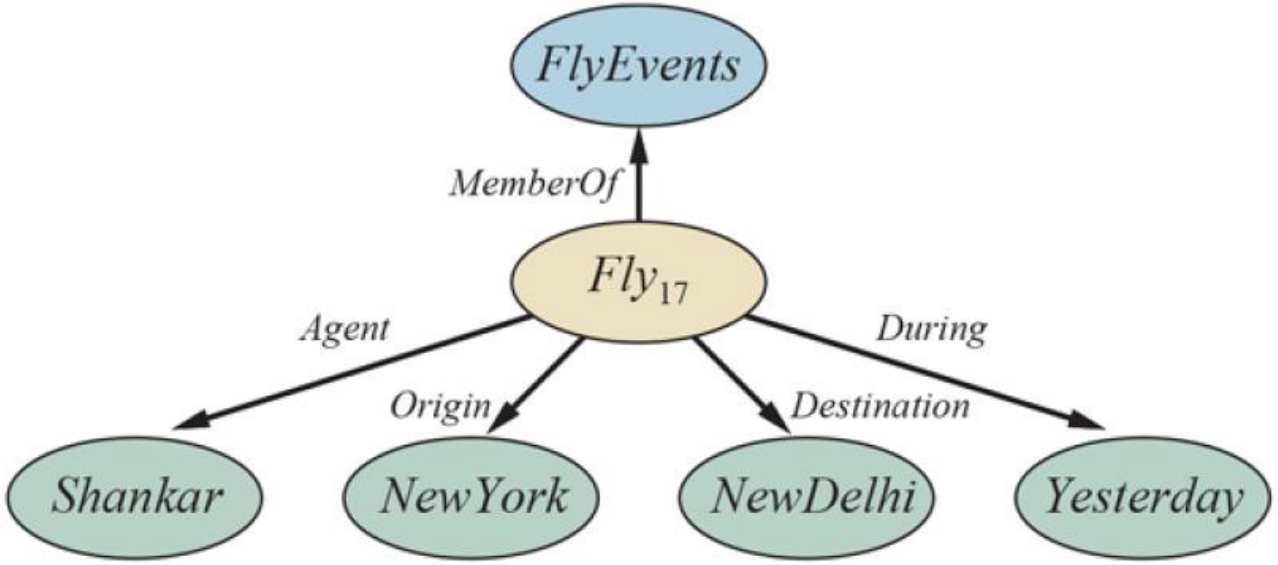


A semantic network with four objects (John, Mary, 1, and 2) and four categories. Relations are denoted by labeled links.

n-ary assertions
Fly(Shankar, NewYork, NewDelhi, Yesterday)

The restriction to binary relations forces the creation of a rich ontology of reified concepts.

Figure 10.5



A fragment of a semantic network showing the representation of the logical assertion *Fly(Shankar,NewYork,NewDelhi,Yesterday)*.

Description logic

- The syntax of first-order logic is designed to make it easy to say things about objects.
- Description logics are notations that are designed to make it easier to describe definitions and properties of categories.
- Description logic systems evolved from semantic networks in response to formalize what the networks meanwhile retaining the emphasis on taxonomic structure as an organizing principle.
- The principal inference tasks for description logics are
  - **Subsumption:** checking if one category is a subset of another by comparing their definitions
  - **Classification:** checking whether an object belongs to a category
  - **Consistency** of a category definition: whether the membership criteria are logically satisfiable.

The CLASSIC language

- A typical description logic

$$Bachelor = And(Unmarried, Adult, Male).$$ Categories

Equivalent FOL Objects

$$Bachelor(x) \Leftrightarrow Unmarried(x) \wedge Adult(x) \wedge Male(x) .$$

To describe the set of men with at least three sons who are all unemployed and married to doctors, and at most two daughters who are all professors in physics or math departments:

$$And(Man, AtLeast\,(3, Son)\,, AtMost\,(2, Daughter)\,,$$
$$All\,(Son, And\,(Unemployed, Married, All\,(Spouse, Doctor)))\,,$$
$$All\,(Daughter, And\,(Professor, Fills\,(Department, Physics, Math)))).$$

Figure 10.6

Concept	→	**Thing** | *ConceptName*
	|	**And**(*Concept*, ...)
	|	**All**(*RoleName*, *Concept*)
	|	**AtLeast**(*Integer*, *RoleName*)
	|	**AtMost**(*Integer*, *RoleName*)
	|	**Fills**(*RoleName*, *IndividualName*, ...)
	|	**SameAs**(*Path*, *Path*)
	|	**OneOf**(*IndividualName*, ...)
Path	→	[*RoleName*, ...]
ConceptName	→	*Adult* | *Female* | *Male* | ...
RoleName	→	*Spouse* | *Daughter* | *Son* | ...

The syntax of descriptions in a subset of the CLASSIC language.

Outline

- Knowledge representation
  - Events
  - Reasoning systems for categories: Semantic networks, Description logic
- Automated planning
  - Classic planning and Planning Domain Definition Lagrange (PDDL)
  - Forward search (progression) and backward search (regression)
  - Heuristics: adding shortcut edges, reducing states
  - Hierarchical planning
  - Planning in nondeterministic domains
  - Scheduling
- Quantifying uncertainty
  - Acting under uncertainty
  - Probabilities

Automated Planning

- Factored representation language for planning problems
- Efficient algorithms and heuristics for planning
- Hierarchical actions
- Partially observable and nondeterministic domains
- Scheduling with resource constraints
- Effectiveness

Definition of classical planning

- The task of finding a sequence of actions to accomplish a goal in a discrete, deterministic, static, fully observable environment.
- Existing solutions we learned so far
  - Problem solving agents with search
    - ad hoc heuristics for each new domain: a heuristic evaluation function for search
  - Propositional logical agents
    - moving a step forward had to be repeated for all four agent orientations, T time steps, and n^2 current locations.
- Factored representation with a family of languages called PDDL (Planning Domain Definition Language)

States

- A state is represented as a conjunction of ground atomic fluents.
  - “ground” means no variables
  - “fluent” means an aspect of the world that changes over time,
  - “ground atomic” means there is a single predicate, and if there are any arguments, they must be constants, i.e., no variables.
- Examples: States in a package delivery problem
  - At($Truck_1$, Melbourne) $\wedge$ At($Truck_2$, Sydney)
  - At(x, y), not ground
  - At(Spouse(Ali), Sydney), not ground atomic
- PDDL uses database semantics:
  - Unique-names assumption
    - A distinct symbol refers to a distinct object. E.g., two different trucks.
  - Closed world assumption
    - Any fluents that are not mentioned are false

Action

- An action schema
  - A family of ground actions
- Action name
- A list of variables in the schema
- A precondition: conjunctions of literals (positive or negated atomic sentences).
- An effect: conjunctions of literals (positive or negated atomic sentences).

$$Action(Fly(p, from, to),$$
$$\text{PRECOND:} At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$$
$$\text{EFFECT:} \neg At(p, from) \wedge At(p, to))$$

Choose constants to instantiate the variables, yielding a ground (variable-free) action:

$$Action(Fly(P_1, SFO, JFK),$$
$$\text{PRECOND:} At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK)$$
$$\text{EFFECT:} \neg At(P_1, SFO) \wedge At(P_1, JFK))$$

Applicable actions

- A ground action *a* is **applicable** in state *s* if *s* entails the precondition of *a*; that is, if every positive literal in the precondition is in *s* and every negated literal is not.
- The result of executing applicable action *a* in state *s* is defined as a state *s'*
  - removing the fluents that appear as negative literals in the action's effects (what we call the delete list or DEL(a)),
  - adding the fluents that are positive literals in the action's effects (what we call the add list or ADD(a)):

$$\text{RESULT}(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a) .$$

$$\begin{aligned} &Action(Fly(P_1, SFO, JFK)), \\ &\quad \text{PRECOND:} At(P_1, SFO) \wedge Plane(P_1) \wedge Airport(SFO) \wedge Airport(JFK) \\ &\quad \text{EFFECT:} \neg At(P_1, SFO) \wedge At(P_1, JFK)) \end{aligned}$$

State s: At(P_1, SFO) → Action a: Fly(P_1, SFO, JFK) → State s': At(P_1, JFK)

A specific planning problem

- A set of action schemas serves as a definition of a planning domain.
- A specific problem within the domain is defined with the addition of an **initial** state and a **goal**.
- The initial state is a conjunction of ground fluents. As with all states, the closed-world assumption is used, which means that any atoms that are not mentioned are false.
- The goal is a conjunction of literals (positive or negative) that may contain variables.
- For example, goal: $At(C_1, SFO) \land \neg At(C_2, SFO) \land At(p, SFO)$,
  - refers to any state in which cargo C_1 is at SFO but C_2 is not, and in which there is a plane at SFO.

Example domain: Air Cargo Transport

Figure 11.1

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT: } \neg At(p, from) \land At(p, to))
\end{aligned}
$$

A PDDL description of an air cargo transportation planning problem.

In(c, p) means that cargo c is inside plane p
At(x, a) means that object x (either plane or cargo) is at airport a.

$$[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK),$$
$$Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)].$$

Example domain: The spare tire problem

Figure 11.2

$Init(Tire(Flat) \land Tire(Spare) \land At(Flat, Axle) \land At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(obj, loc),$
PRECOND: $At(obj, loc)$
EFFECT: $\neg At(obj, loc) \land At(obj, Ground))$
$Action(PutOn(t, Axle),$
PRECOND: $Tire(t) \land At(t, Ground) \land \neg At(Flat, Axle) \land \neg At(Spare, Axle)$
EFFECT: $\neg At(t, Ground) \land At(t, Axle))$
$Action(LeaveOvernight,$
PRECOND:
EFFECT: $\neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk)$
$\land \neg At(Flat, Ground) \land \neg At(Flat, Axle) \land \neg At(Flat, Trunk))$

The simple spare tire problem.

[Remove(Flat,Axle),Remove(Spare, Trunk), PutOn(Spare,Axle)]

Example domain: The blocks world

- The domain consists of a set of cube-shaped blocks sitting on an arbitrarily-large table.
- The blocks can be stacked, but only one block can fit directly on top of another.
- A robot arm can pick up a block and move it to another position, either on the table or on top of another block. The arm can pick up only one block at a time, so it cannot pick up a block that has another one on top of it.

Figure 11.3

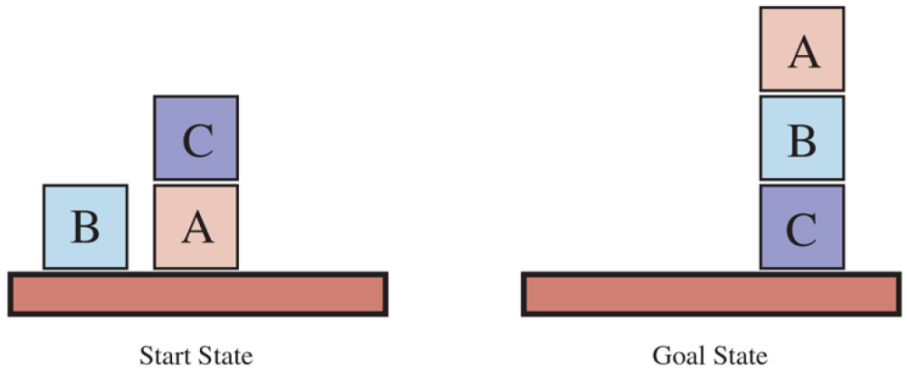


Diagram of the blocks-world problem in Figure 11.4.

Figure 11.4

arbitrarily-large table

$$
\begin{aligned}
&Init(On(A,Table) \land On(B,Table) \land On(C,A) \\
&\quad \land Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C) \land Clear(Table)) \\
&Goal(On(A,B) \land On(B,C)) \\
&Action(Move(b,x,y), \\
&\quad \text{PRECOND}: On(b,x) \land Clear(b) \land Clear(y) \land Block(b) \land Block(y) \land \\
&\qquad (b \neq x) \land (b \neq y) \land (x \neq y), \\
&\quad \text{EFFECT}: On(b,y) \land Clear(x) \land \neg On(b,x) \land \neg Clear(y)) \\
&Action(MoveToTable(b,x), \\
&\quad \text{PRECOND}: On(b,x) \land Clear(b) \land Block(b) \land Block(x), \\
&\quad \text{EFFECT}: On(b,Table) \land Clear(x) \land \neg On(b,x))
\end{aligned}
$$

A planning problem in the blocks world: building a three-block tower. One solution is the sequence $[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]$.

Figure 11.3

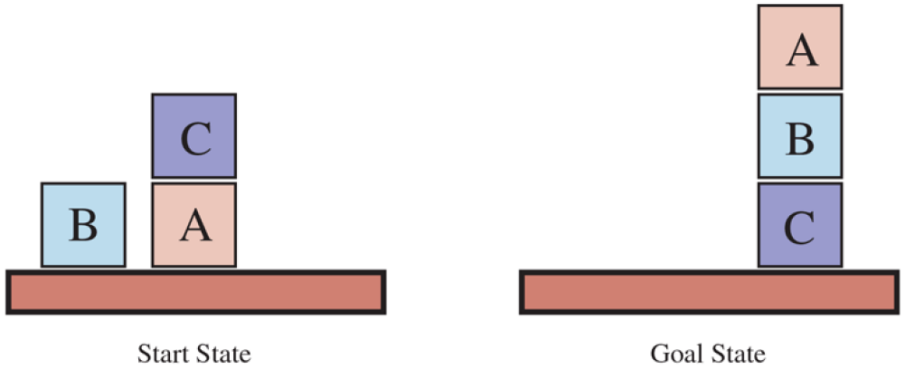


Diagram of the blocks-world problem in Figure 11.4.

- Move(b, x, y): move block b from x to y
- predicate clear(x) that is true when nothing is on .

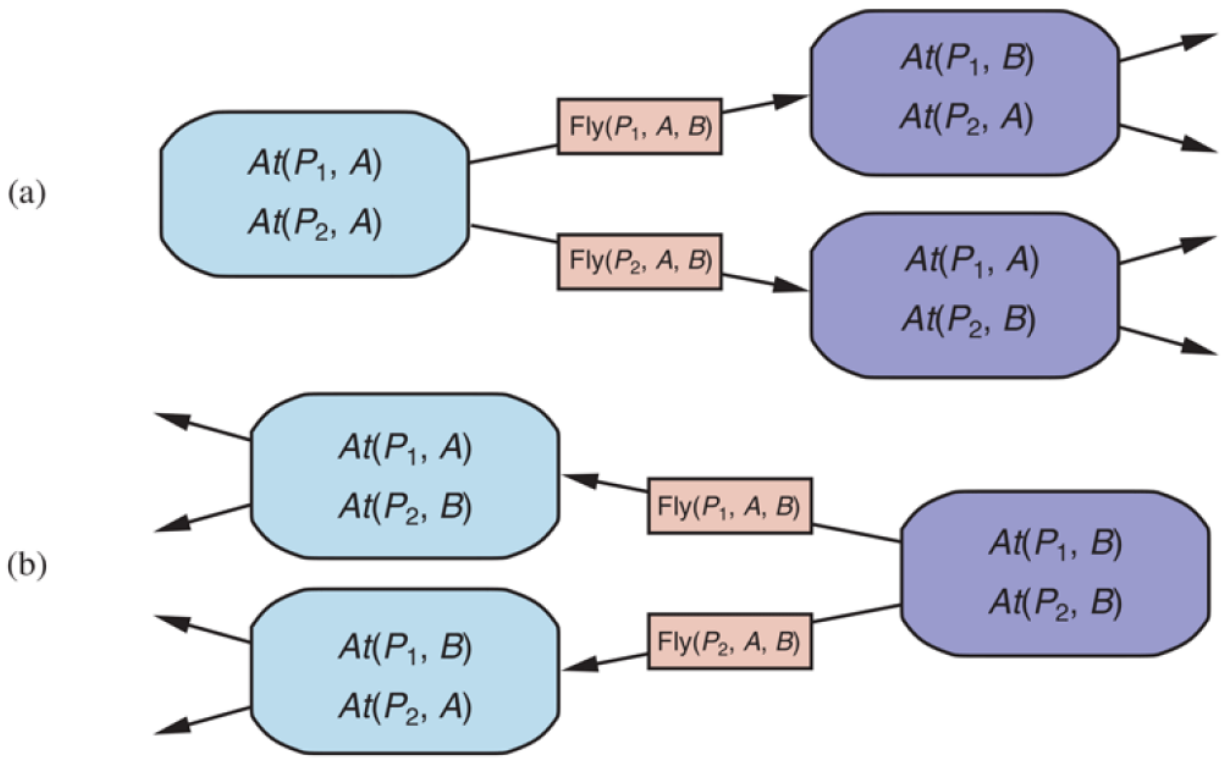
Outline

- Knowledge representation
  - Events
  - Reasoning systems for categories: Semantic networks, Description logic
- Automated planning
  - Classic planning and Planning Domain Definition Lagrange (PDDL)
  - Forward search (progression) and backward search (regression)
  - Heuristics: adding shortcut edges, reducing states
  - Hierarchical planning
  - Planning in nondeterministic domains
  - Scheduling
- Quantifying uncertainty
  - Acting under uncertainty
  - Probabilities

Algorithms for classical planning

- Three possible ways
  - The description of a planning problem provides an obvious way to **search** from the initial state through the space of states, looking for a goal.
  - A nice advantage of the declarative representation of action schemas is that we can also **search backward** from the goal, looking for the initial state.
  - A third possibility is to translate the problem description into a set of logic sentences, to which we can apply a **logical inference** algorithm to find a solution.

Figure 11.5

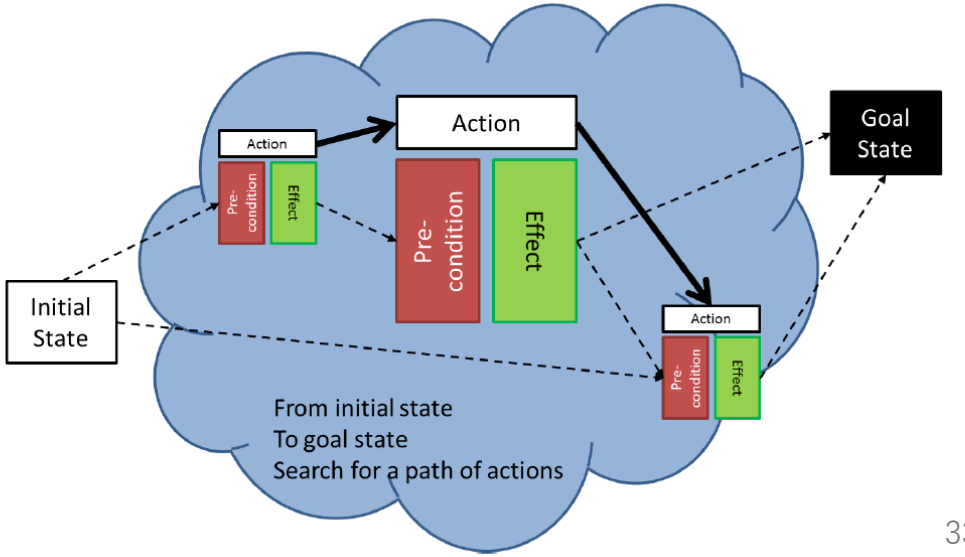


Two approaches to searching for a plan. (a) Forward (progression) search through the space of ground states, starting in the initial state and using the problem's actions to search forward for a member of the set of goal states. (b) Backward (regression) search through state descriptions, starting at the goal and using the inverse of the actions to search backward for the initial state.

Forward state-space search

- Search algorithms
- The states in this search state space are ground states, where every fluent is either true or not.
- The goal is a state that has all the positive fluents in the problem's goal and none of the negative fluents.
- State-space search considers actions that are **applicable**

A ground action *a* is **applicable** in state *s* if *s* entails the precondition of *a*; that is, if every positive literal in the precondition is in *s* and every negated literal is not.



Backward search

- In backward search (also called regression search) we start at the goal and apply the actions backward until we find a sequence of steps that reaches the initial state.
- At each step we consider **relevant** actions (in contrast to forward search, which considers actions that are applicable).
- This reduces the branching factor significantly, particularly in domains with many possible actions.
- A **relevant action** is one with an effect that unifies with one of the goal literals, but with no effect that negates any part of the goal.
  - With the goal ¬Poor ∧ Famous, an action with the sole effect Famous would be relevant, but one with the effect Poor ∧ Famous is not considered relevant: because then Poor would appear in the final state.

Other planning approaches

- Planning as **Boolean satisfiability**
  - Translating a PDDL problem description into propositional form.
- Graphplan uses a **planning graph** to encode constraints on how actions are related to their preconditions and effects.
- **Situation calculus** describes planning problems in FOL.
  - Has not made a big impact in practical applications
- **Partial-order planning**
  - each action is a node in the graph, and for each precondition of the action there is an edge from another action (or from the initial state) that indicates that the predecessor action establishes the precondition.
  - We search in the space of plans rather than world-states, inserting actions to satisfy conditions.
  - Often used in domains where it is important for humans to understand the plans.

Outline

- Knowledge representation
  - Events
  - Reasoning systems for categories: Semantic networks, Description logic
- Automated planning
  - Classic planning and Planning Domain Definition Lagrange (PDDL)
  - Forward search (progression) and backward search (regression)
  - Heuristics: adding shortcut edges, reducing states
  - Hierarchical planning
  - Planning in nondeterministic domains
  - Scheduling
- Quantifying uncertainty
  - Acting under uncertainty
  - Probabilities

Heuristics for planning

- Neither forward nor backward search is efficient without a good heuristic function.
- Recall that a heuristic function h(s) estimates the distance from a state to the goal, and that if we can derive an admissible heuristic for this distance—one that does not overestimate—then we can use A* search to find optimal solutions.
- By definition, there is no way to analyze an atomic state, and thus it requires some ingenuity by an analyst (usually human) to define good **domain-specific** heuristics for search problems with atomic states.
  - Generating heuristics from relaxed problems
  - Generating heuristics from subproblems, pattern databases
  - Generating heuristics from with landmarks
  - Learning heuristics from experience
- But planning uses a factored representation for states and actions, which makes it possible to define good **domain-independent** heuristics.

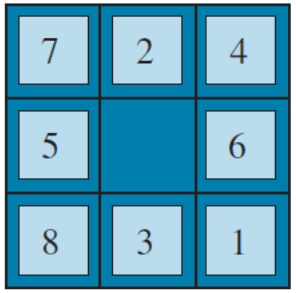
Relaxed problem

- Recall that an admissible heuristic can be derived by defining a relaxed problem that is easier to solve.
- The exact cost of a solution to this easier problem then becomes the heuristic for the original problem.
- A search problem is a graph where the nodes are states and the edges are actions. The problem is to find a path connecting the initial state to a goal state.
- There are two main ways we can relax this problem to make it easier:
  - By **adding more edges** to the graph, making it strictly easier to find a path; (adding short-cuts)
  - By **grouping multiple nodes together**, forming an abstraction of the state space that has fewer states, and thus is easier to search.

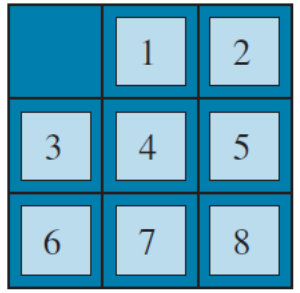
Adding edges, ignore-preconditions heuristics

- Drops all preconditions from actions.
- Every action becomes applicable in every state, and any single goal fluent can be achieved in one step (if there are any applicable actions—if not, the problem is impossible).
- This almost implies that the number of steps required to solve the relaxed problem is the number of unsatisfied goals
  - Almost but not quite, because (1) some action may achieve multiple goals and (2) some actions may undo the effects of others.

Ignore-preconditions heuristics



Start State



Goal State

- We relax the actions by removing all preconditions and all effects except those that are literals in the goal.
- Then, we count the minimum number of actions required such that the union of those actions' effects satisfies the goal.
  - Set-cover problem
- It is also possible to ignore only selected preconditions of actions.

$$
\begin{aligned}
&Action(Slide\,(t, s_1, s_2)\,,\\
&\quad \textsc{Precond}{:}On\,(t, s_1) \wedge Tile\,(t) \wedge Blank\,(s_2) \wedge Adjacent\,(s_1, s_2)\,,\\
&\quad \textsc{Effect}{:}On\,(t, s_2) \wedge Blanks\,(s_1) \wedge \neg On\,(t, s_1) \wedge \neg Blank\,(s_2))
\end{aligned}
$$

  - Remove Blank(s_2) and Adjacent(s_1, s_2): number of misplaced titles
  - Remove Blank(s_2): Manhattan distance

Ignore delete-lists heuristic

- Assume that all goals and preconditions contain only positive literals.
  - replace every negative literal ¬ P in a goal or precondition with a new positive literal p', and modify the initial state and the action effects accordingly.
- We want to create a relaxed version of the original problem that will be easier to solve, and where **the length** of the solution will serve as a good heuristic.
- We can do that by removing the delete lists from all actions (i.e., removing all negative literals from effects).

Reducing states: State abstraction

- Relaxing the actions does nothing to reduce the number of states, which means that it may still be expensive to compute the heuristic.
- We now look at relaxations that decrease the number of states by forming a state abstraction—a many-to-one mapping from states in the ground representation of the problem to the abstract representation.

Outline

- Knowledge representation
  - Events
  - Reasoning systems for categories: Semantic networks, Description logic
- Automated planning
  - Classic planning and Planning Domain Definition Lagrange (PDDL)
  - Forward search (progression) and backward search (regression)
  - Heuristics: adding shortcut edges, reducing states
  - Hierarchical planning
  - Planning in nondeterministic domains
  - Scheduling
- Quantifying uncertainty
  - Acting under uncertainty
  - Probabilities

Hierarchical planning

- A high-level plan for a Hawaii vacation might be “Go to San Francisco airport; take flight HA 11 to Honolulu; do vacation stuff for two weeks; take HA 12 back to San Francisco; go home.”
- Given such a plan, the action “Go to San Francisco airport” can be viewed as a planning task in itself, with a solution such as “Choose a ride-hailing service; order a car; ride to airport.”
- Each of these actions, in turn, can be decomposed further, until we reach the low-level motor control actions like a button-press.
- **Hierarchical decomposition**: an idea that pervades almost all attempts to manage complexity.

Outline

- Knowledge representation
  - Events
  - Reasoning systems for categories: Semantic networks, Description logic
- Automated planning
  - Classic planning and Planning Domain Definition Lagrange (PDDL)
  - Forward search (progression) and backward search (regression)
  - Heuristics: adding shortcut edges, reducing states
  - Hierarchical planning
  - Planning in nondeterministic domains
  - Scheduling
- Quantifying uncertainty
  - Acting under uncertainty
  - Probabilities

Planning and Acting in Nondeterministic Domains

- **Sensorless planning** (also known as conformant planning) for environments with no observations;
  - Belief state planning
- **Contingency planning** for partially observable and nondeterministic environments;
  - AND-OR search
- **Online planning** and replanning for unknown environments.

Outline

- Knowledge representation
  - Events
  - Reasoning systems for categories: Semantic networks, Description logic
- Automated planning
  - Classic planning and Planning Domain Definition Lagrange (PDDL)
  - Forward search (progression) and backward search (regression)
  - Heuristics: adding shortcut edges, reducing states
  - Hierarchical planning
  - Planning in nondeterministic domains
  - Scheduling
- Quantifying uncertainty
  - Acting under uncertainty
  - Probabilities

Time, Schedules, and Resources

- Classical planning talks about what to do, in what order, but does not talk about time: how long an action takes and when it occurs.
- For example, in the airport domain we could produce a plan saying what planes go where, carrying what, but could not specify departure and arrival times. This is the subject matter of **scheduling**.
- The real world also imposes resource constraints: an airline has a limited number of staff, and staff who are on one flight cannot be on another at the same time.
  - Techniques for planning and scheduling problems with resource constraints.
- The approach we take is "**plan first, schedule later**": divide the overall problem into
  - A planning phase in which actions are selected, with some ordering constraints, to meet the goals of the problem, and
  - A scheduling phase, in which temporal information is added to the plan to ensure that it meets resource and deadline constraints.
- Solutions can be evaluated according to a cost function
  - For simplicity, cost function is just the total duration of the plan, called the makespan.

Representing temporal and resource constraints

- A typical job-shop scheduling problem, consists of a set of **jobs**, each of which has a collection of **actions** with ordering constraints among them.
- Each action has a duration and a set of resource constraints required by the action. A constraint
  - specifies a type of resource (e.g., bolts, wrenches, or pilots)
  - the number of that resource required
  - whether that resource is consumable (e.g., the bolts are no longer available for use) or reusable (e.g., a pilot is occupied during a flight but is available again when the flight is over).
- Actions can also produce resources (e.g., manufacturing and resupply actions).

Example: the assembly of two cars

- The representation of resources as numerical quantities, such as Inspectors(2), rather than as named entities, such as I_1 and I_2, is an example of a technique called **aggregation.**
- Aggregation is essential for reducing complexity.
  - Consider what happens when a proposed schedule has 10 concurrent Inspect actions but only 9 inspectors are available. With inspectors represented as quantities, a failure is detected immediately and the algorithm backtracks to try another schedule.
  - With inspectors represented as individuals, the algorithm would try all 9! ways of assigning inspectors to actions before noticing that none of them work.

Figure 11.13

$$Jobs(\{AddEngine1 \prec AddWheels1 \prec Inspect1\}, \{AddEngine2 \prec AddWheels2 \prec Inspect2\})$$

$$Resources(EngineHoists(1), WheelStations(1), Inspectors(e2), LugNuts(500))$$

$$Action(AddEngine1, \text{DURATION}:30, \text{USE}:EngineHoists(1))$$
$$Action(AddEngine2, \text{DURATION}:60, \text{USE}:EngineHoists(1))$$

reusable

$$Action(AddWheels1, \text{DURATION}:30, \text{CONSUME}:LugNuts(20), \text{USE}:WheelStations(1))$$
$$Action(AddWheels2, \text{DURATION}:15, \text{CONSUME}:LugNuts(20), \text{USE}:WheelStations(1))$$

consumable

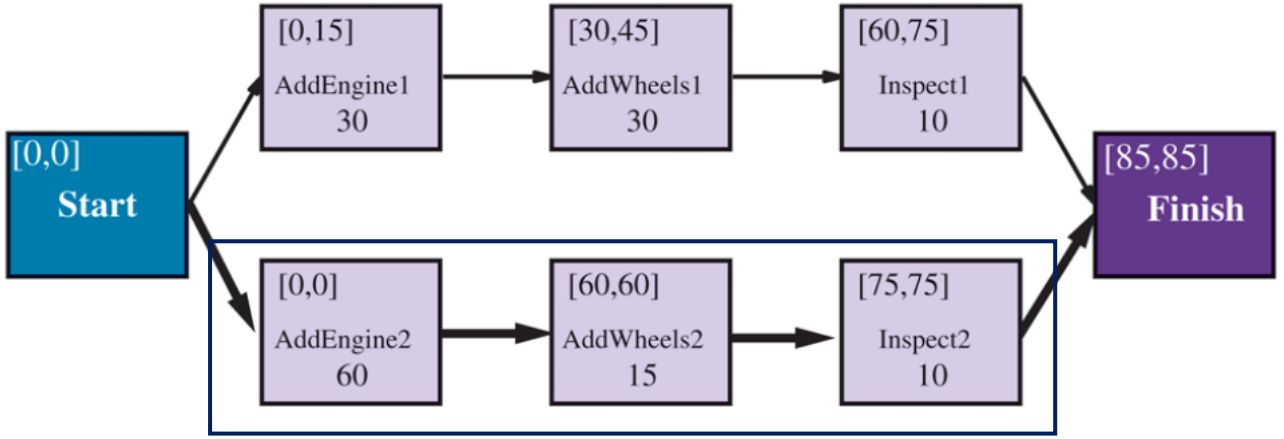
$$Action(Inspect_i, \text{DURATION}:10, \text{USE}:Inspectors(1))$$

A job-shop scheduling problem for assembling two cars, with resource constraints. The notation $A \prec B$ means that action A must precede action B.

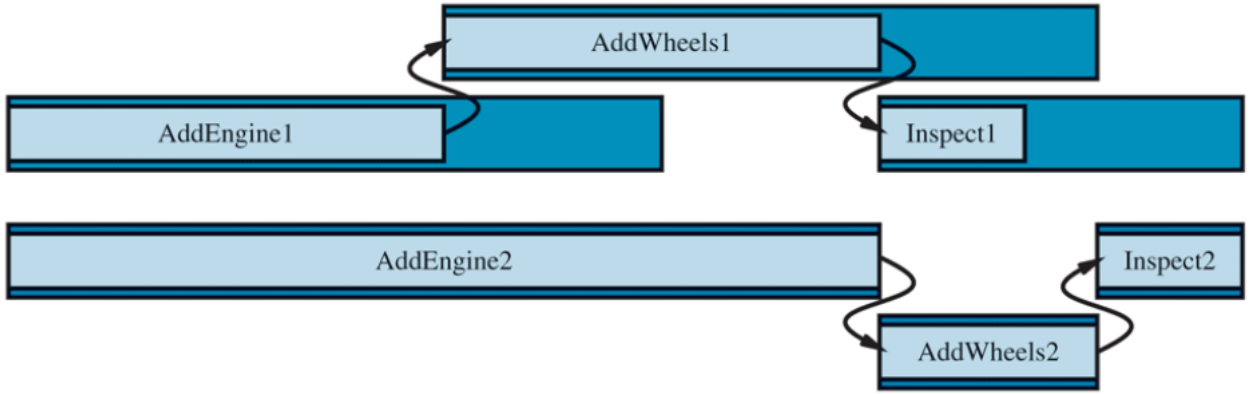
Solving scheduling problem

- For now, considering just the temporal scheduling problem, but ignoring resource constraints.
- To minimize makespan (plan duration), we must find the **earliest start times** for all the actions consistent with the ordering constraints supplied with the problem. It is helpful to view these ordering constraints as a **directed graph** relating the actions.
- We can apply the critical path method (CPM) to this graph to determine the possible start and end times of each action. A path through a graph representing a partial-order plan is a linearly ordered sequence of actions beginning with Start and ending with Finish.

- The critical path is that path whose total duration is longest; the path is “critical” because it determines the duration of the entire plan—shortening other paths doesn’t shorten the plan as a whole, but delaying the start of any action on the critical path slows down the whole plan.
- Actions that are off the critical path have a window of time in which they can be executed. The window is specified in terms of an earliest possible start time, ES, and a latest possible start time, LS. The quantity LS–ES is known as the slack of an action.
- We can see that the whole plan will take 85 minutes, that each action in the top job has 15 minutes of slack, and that each action on the critical path has no slack (by definition).
- Together the ES and LS times for all the actions constitute a schedule for the problem.



15 min slack



Defining ES and LS

A and B are actions, and $A \prec B$ means that A precedes B:

$$ES(Start) = 0$$
$$ES(B) = \max_{A \prec B} ES(A) + Duration(A)$$
$$LS(Finish) = ES(Finish)$$
$$LS(A) = min_{B \prec A} LS(B) - Duration(A).$$

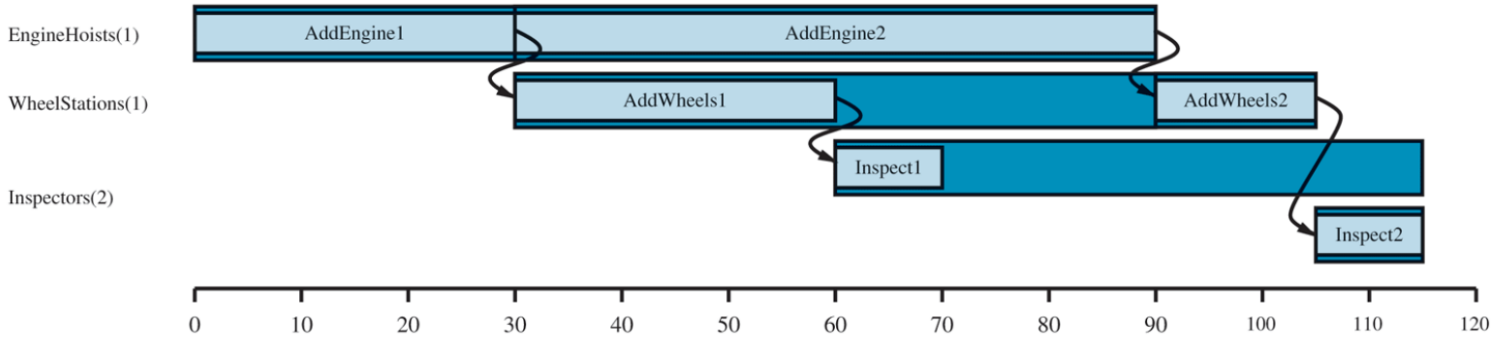
The maximum of the earliest finish times of those immediately preceding actions.
The earliest finish time = the earliest start time + the duration.

Often use dynamic-programming algorithms

The complexity of the critical path algorithm is just $O(Nb)$, where N is the number of actions and b is the maximum branching factor into or out of an action. (To see this, note that the LS and ES computations are done once for each action, and each computation iterates over at most b other actions.) Therefore, finding a minimum-duration schedule, given a partial ordering on the actions and no resource constraints, is quite easy.

Resource constraints

- More complicated



- One popular heuristic is the **minimum slack** heuristic: on each iteration, schedule for the earliest possible start whichever unscheduled action has all its predecessors scheduled and has the least slack.
  - The heuristic often works well in practice. But in our example,
  - AddEngine2 will be selected first, thus resulting in a 130-min solution
    - Optimal solution: 115-minute

Analysis of Planning Approaches

- Planning combines the two major areas of AI we have covered so far: search and logic.
- A planner can be seen either as a program that searches for a solution or as one that (constructively) proves the existence of a solution.
- Unfortunately, we do not yet have a clear understanding of which techniques work best on which kinds of problems.
- New techniques will emerge, perhaps providing a synthesis of highly expressive first-order and hierarchical representations with the highly efficient factored and propositional representations that dominate today.
- Portfolio planning systems, where a collection of algorithms are available to apply to any given problem.
  - This can be done selectively (the system classifies each new problem to choose the best algorithm for it).
  - or in parallel (all the algorithms run concurrently, each on a different CPU).
  - or by interleaving the algorithms according to a schedule.

Outline

- Knowledge representation
  - Events
  - Reasoning systems for categories: Semantic networks, Description logic
- Automated planning
  - Classic planning and Planning Domain Definition Lagrange (PDDL)
  - Forward search (progression) and backward search (regression)
  - Heuristics: adding shortcut edges, reducing states
  - Hierarchical planning
  - Planning in nondeterministic domains
  - Scheduling
- Quantifying uncertainty
  - Acting under uncertainty
  - Probabilities

Acting under uncertainty

- Agents in the real world need to handle uncertainty, whether due to partial observability, nondeterminism, or adversaries.
- Let action A_t = leave for airport t minutes before flight.
  - Will A_t get me there on time?
- Problems
  - partial observability (road state, other drivers' plans, etc.)
  - noisy sensors (traffic reports)
  - uncertainty in action outcomes (flat tire, etc.)
  - immense complexity of modelling and predicting traffic
- Hence a purely logical approach either
  - risks falsehood: "A_{90} will get me there on time"
  - leads to conclusions that are too weak for decision making:
    - "A_{90} will get me there on time if there's no accident on the bridge and it doesn't rain and my tires remain intact, and no meteorite hits the car, and…"
    - None of these conditions can be deduced for sure, so we can't infer that the plan succeeds.

$$Toothache \Rightarrow Cavity.$$

Methods for handling uncertainty

$$Toothache \Rightarrow Cavity \vee GumProblem \vee Abscess \ldots$$

- Logic fails to cope uncertainty
  - Laziness: It is too much work to list the complete set of antecedents or consequents needed to ensure an exceptionless rule and too hard to use such rules.
  - Ignorance:
    - Theoretical ignorance: Medical science has no complete theory for the domain.
    - Practical ignorance: Even if we know all the rules, we might be uncertain about a particular patient because not all the necessary tests have been or can be run.
- The theory of probability provides a way of summarizing the uncertainty that comes from our laziness and ignorance, thereby solving the qualification problem.

Uncertainty and rational decisions

$$P(A_{25} \text{ gets me there on time}|\ldots) = 0.04$$
$$P(A_{90} \text{ gets me there on time}|\ldots) = 0.70$$
$$P(A_{120} \text{ gets me there on time}|\ldots) = 0.95$$
$$P(A_{1440} \text{ gets me there on time}|\ldots) = 0.9999$$

- Consider again the plan A_{90} for getting to the airport.
- Suppose A_{90} gives us a 70% chance of catching our flight.
- Does this mean it is a rational choice?
  - Not necessarily: there might be other plans, such as A_{120}, with higher probabilities.
  - If it is vital not to miss the flight, then it is worth risking the longer wait at the airport.
- What about A_{1440}, a plan that involves leaving home 24 hours in advance?
  - In most circumstances, this is not a good choice, because although it almost guarantees getting there on time, it involves an intolerable wait—not to mention a possibly unpleasant diet of airport food.

Simon Aagaard Pedersen, Bin Yang, Christian S. Jensen, Jesper Møller: Stochastic Routing with Arrival Windows. ACM Trans. Spatial Algorithms Syst. 9(4): 30:1-30:48 (2023)

(1) ***Stochastic Routing with an Arrival Window*** Given source and destination vertices s and d and an arrival window AW, return a path P from s to d and a departure time T_D that satisfy the following: When departing from s at T_D, path P has the maximum probability of arriving at d within AW across all paths $\mathbb{P}$ between s and d and across all possible departure times $\mathbb{T}$. Finally, any path with an earlier departure time has a lower such probability. Formally:

$$SRAW(s, d, AW) = (P, T_D) = \max_{el2}(\arg\max_{(P,T_D)\in\mathbb{P}\times\mathbb{T}} P.Prob(AW, T_D)),$$

where $P.Prob(AW, T_D)$ returns the probability of arriving within arrival window AW when departing at time T_D for path P, and $\max_{eli}$ returns a tuple in the argument set with the maximum value for the i'th element.

(2) ***Stochastic Routing with an Arrival Window and a Threshold*** Given source and destination vertices s and d, an arrival window AW, and a threshold t, return a path P from s to d and a departure time T_D that satisfy the following: When departing from s at T_D, path P yields a probability above t of arriving at d within AW. Finally, any path, across all paths $\mathbb{P}$ between s and d and across all possible departure times $\mathbb{T}$ with a departure time later than T_D, has arrival probability at most t. Formally,

$$SRAWT(s, d, AW, t) = \max_{el2}(\{(P, T_D) \in \mathbb{P} \times \mathbb{T} | P.Prob(AW, T_D) > t\}).$$

Preferences

- To make such choices, an agent must first have preferences among the different possible outcomes of the various plans. An outcome is a completely specified state, including such factors as whether the agent arrives on time and the length of the wait at the airport.
- We use utility theory to represent preferences and reason quantitatively with them. (The term utility is used here in the sense of "the quality of being useful.")
- Utility theory says that every state (or state sequence) has a degree of usefulness, or utility, to an agent and that the agent will prefer states with higher utility.
- Preferences, as expressed by utilities, are combined with probabilities in the general theory of rational decisions called **decision theory**
  - **Decision theory = probability theory + utility theory.**

Rational agent

Jilin Hu, Bin Yang, Chenjuan Guo, Christian S. Jensen: Risk-aware path selection with time-varying, uncertain travel costs: a time series approach. VLDB J. 27(2): 179-200 (2018)

- The fundamental idea of **decision theory** is that an agent is rational if and only if it chooses the action that yields the highest expected utility, averaged over all the possible outcomes of the action.
- This is called the principle of **maximum expected utility** (MEU).

Table 1 Uncertain travel times for P_1, P_2, and P_3, [8:00, 8:15)

Travel time (min)	70	80	90	100	110	120
P_1	0	0.25	0.50	0	0	0.25
P_2	0	0	0.50	0.50	0	0
P_3	0	0	0	0.50	0	0.50

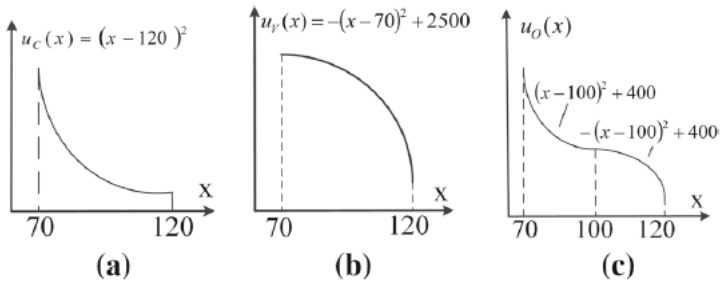


Fig. 2 Categorization of utility functions. **a** Convex, **b** concave and **c** other

Table 3 Risk preferences, utility functions, and stochastic dominance

Risk attitudes	Utility functions	Stochastic dominance
Risk-neutral	Non-increasing	First order
Risk-loving	Non-incr., convex	Second convex order
Risk-averse	Non-incr., concave	Second concave order

Table 4 Expected utilities for paths P_1, P_2, and P_3

	P_1	P_2	P_3	Optimal
u_C	**850**	650	200	P_1
u_V	1650	**1850**	800	P_2
u_O	**450**	**450**	200	P_1, P_2

Outline

- Knowledge representation
  - Events
  - Reasoning systems for categories: Semantic networks, Description logic
- Automated planning
  - Classic planning and Planning Domain Definition Lagrange (PDDL)
  - Forward search (progression) and backward search (regression)
  - Heuristics: adding shortcut edges, reducing states
  - Hierarchical planning
  - Planning in nondeterministic domains
  - Scheduling
- Quantifying uncertainty
  - Acting under uncertainty
  - Probabilities

Probability theory

- Begin with a set Ω—the sample space
  e.g., 6 possible rolls of a die.
  $\omega \in \Omega$ is a sample point/possible world/atomic event

- A probability space or probability model is a sample space with an assignment $P(\omega)$ for every $\omega \in \Omega$ s.t.
  $0 \leq P(\omega) \leq 1$
  $\sum_{\omega} P(\omega) = 1$
  e.g., $P(1) = P(2) = P(3) = P(4) = P(5) = P(6) = 1/6$.

- An event A is any subset of Ω

$$P(A) = \sum_{\{\omega \in A\}} P(\omega)$$

- E.g., $P(\text{die roll} \leq 3) = P(1) + P(2) + P(3) = 1/6 + 1/6 + 1/6 = 1/2$

- In logic, a set of worlds corresponds to a proposition in a formal language.
  - For each proposition, the corresponding set contains just those possible worlds in which the proposition holds.

$$\text{For any proposition } \phi, P(\phi) = \sum_{\omega \in \phi} P(\omega).$$

  - Rolling two fair dice
    - P(Total=11)=P((5, 6))+P((6,5))=1/36+1/36=1/18

Priors and posterior

- Unconditional/prior probability, priors
  - Degrees of belief in propositions **in the absence of any other information**
- Condition/posterior probability, posterior, denoted P(a|b)
  - Total is 11, given the first die is a 5.

$$P(a \mid b) = \frac{P(a \wedge b)}{P(b)}, \qquad P(a \wedge b) = P(a \mid b)P(b).$$

  - P(total=11|5)=P(total=11 and one die is 5)/P(one die is 5) = (1/36)/(1/6)=6/36=1/6
  - P(Total=11)=P((5, 6))+P((6,5))=1/36+1/36=1/18

Random variables

- Variables in probability theory are called random variables.
- The range of Total for two dice is the set {2, 3, ⋯, 12}
- Weather={sun, rain, cloud, snow}
- Probability distribution $\mathbf{P}(Weather) = \langle 0.6, 0.1, 0.29, 0.01 \rangle$
  - categorical distribution, probability mass function (pmf)

$$P(Weather = sun) = 0.6$$
$$P(Weather = rain) = 0.1$$
$$P(Weather = cloud) = 0.29$$
$$P(Weather = snow) = 0.01\,,$$

- For continuous variables, probability density function (pdf)

$$P(NoonTemp = x) = Uniform(x; 18C, 26C) = \begin{cases} \frac{1}{8C} & \text{if } 18C \leq x \leq 26C \\ 0 & \text{otherwise} \end{cases}$$

Distributions on multiple variables

- Joint probability function
- Weather={sun, rain, cloud, snow}
- Cavity={true, false}
- P(Weather, Cavity), 4 X 2=8 possible worlds.

$$\mathbf{P}(Weather, Cavity) = \mathbf{P}(Weather \mid Cavity)\mathbf{P}(Cavity)\,,$$

$$\begin{aligned}
&P(W = sun \wedge C = true) = P(W = sun|C = true)\,P(C = true)\\
&P(W = rain \wedge C = true) = P(W = rain|C = true)\,P(C = true)\\
&P(W = cloud \wedge C = true) = P(W = cloud|C = true)\,P(C = true)\\
&P(W = snow \wedge C = true) = P(W = snow|C = true)\,P(C = true)\\
&P(W = sun \wedge C = false) = P(W = sun|C = false)\,P(C = false)\\
&P(W = rain \wedge C = false) = P(W = rain|C = false)\,P(C = false)\\
&P(W = cloud \wedge C = false) = P(W = cloud|C = false)\,P(C = false)\\
&P(W = snow \wedge C = false) = P(W = snow|C = false)\,P(C = false)\,.
\end{aligned}$$

Lecture 9 ILOs

- Knowledge representation
  - Events
  - Reasoning systems for categories
    - Semantic networks
    - Description logic
- Automated planning
  - Classic planning and Planning Domain Definition Lagrange (PDDL)
  - Algorithms for classic planning
    - Forward search (progression) and backward search (regression)
    - Heuristics: adding shortcut edges, reducing states
  - Hierarchical planning
  - Planning in nondeterministic domains
  - Scheduling
- Quantifying uncertainty
  - Acting under uncertainty
  - Probability