ARTIFICIAL INTELLIGENCE-SCSX1021
UNIT IV
ACTING LOGICALLY

PLANNING

The task of coming up with a sequence of actions that will achieve a goal is called planning. Environments that are fully observable, deterministic, finite, static and discrete are called classical planning environments.

PLANNING PROBLEM

The **representation of planning** problems—states, actions,and goals should make it possible for planning algorithms to take advantage of the logical structure of the problem. The basic representation language of classical planners, known as the STRIPS language. Some of the possible variations in STRIPS-like languages.

Representation of states: Planners decompose the world into logical conditions and represent a state as a conjunction of positive literals. for example, Poor ^ Unknown might represent the state of a hapless agent.

Representation of goals: A goal is a partially specified state, represented as a conjunction of positive ground literals, such as Rich ^ Famous.

Representation of actions: An action is specified in terms of the preconditions that must hold before it can be executed and the effects that ensue when it is executed. For example, an action for flying a plane from one location to another is: Action(Fly(p; from; to)

Action Schema means it represents a number of different actions that can be derived by instantiating the variables p, from, and to to different constants. In general, an action schema consists of three parts:

i)The action name and parameter list.

ii) The **precondition** is a conjunction of function-free positive literals stating what must be true in a state before the action can be executed. Any variables in the precondition must also appear in the action's parameter list.

iii) The **effect** is a conjunction of function-free literals describing how the state changes when the action is executed. A positive literal P in the effect is asserted to be true in the state resulting from the action, whereas a negative literal :P is asserted to be false.Variables in the effect must also appear in the action's parameter list.

STRIPS Assumption

Assumption: Every literal not mentioned in the effect remains unchanged in the resulting state when the action is executed.

Solution for the planning problem: An action sequence that, when executed in the initial state, results in a state that satisfies the goal.

Comparision of STRIPS and ADL (Action Description Language)

STRIPS Language	ADL Language
Only positive literals in states: $Poor \wedge Unknown$	Positive and negative literals in states: $\neg Rich \wedge \neg Famous$
Closed World Assumption: Unmentioned literals are false.	Open World Assumption: Unmentioned literals are unknown.
Effect $P \wedge \neg Q$ means add P and delete Q.	Effect $P \wedge \neg Q$ means add P and $\neg Q$ and delete $\neg P$ and Q.
Only ground literals in goals: $Rich \wedge Famous$	Quantified variables in goals: $\exists x At(P_1, x) \wedge At(P_2, x)$ is the goal of having P_1 and P_2 in the same place.
Goals are conjunctions: $Rich \wedge Famous$	Goals allow conjunction and disjunction: $\neg Poor \wedge (Famous \vee Smart)$
Effects are conjunctions.	Conditional effects allowed: when P: E means E is an effect only if P is satisfied.
No support for equality.	Equality predicate $(x = y)$ is built in.
No support for types.	Variables can have types, as in $(p : Plane)$.

Example: Air cargo transport

Air cargo transport problem involving loading and unloading cargo onto and off of planes and flying it from place to place. The problem can be defined with three actions: Load, Unload, and Fly. The actions affect two predicates: In(c; p) means that cargo c is inside plane p, and At (x; a) means that object x (either plane or cargo) is at airport a. Note that cargo is not At anywhere when it is In a plane, so At really means “available for use at a given location.” It takes

some experience with action definitions to handle such details consistently. The following plan is a solution to the problem:

[Load(C1; P1; SFO); Fly(P1; SFO; JFK);
Load(C2; P2; JFK); Fly(P2; JFK; SFO)] :

Our representation is pure STRIPS. In particular, it allows a plane to fly to and from the same airport.

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

Example: The spare tire problem

Consider the problem of changing a flat tire. More precisely, the goal is to have a good spare tire properly mounted onto the car's axle, where the initial state has a flat tire on the axle and a good spare tire in the trunk. To keep it simple, our version of the problem is a very abstract one, with no sticky lug nuts or other complications. There are just four actions: removing the spare from the trunk, removing the flat tire from the axle, putting the spare on the axle, and leaving the car unattended overnight.

$$
\begin{aligned}
&Init(At(Flat, Axle) \wedge At(Spare, Trunk))\\
&Goal(At(Spare, Axle))\\
&Action(Remove(Spare, Trunk),\\
&\quad \textsc{Precond}: At(Spare, Trunk)\\
&\quad \textsc{Effect}: \neg At(Spare, Trunk) \wedge At(Spare, Ground))\\
&Action(Remove(Flat, Axle),\\
&\quad \textsc{Precond}: At(Flat, Axle)\\
&\quad \textsc{Effect}: \neg At(Flat, Axle) \wedge At(Flat, Ground))\\
&Action(PutOn(Spare, Axle),\\
&\quad \textsc{Precond}: At(Spare, Ground) \wedge \neg At(Flat, Axle)\\
&\quad \textsc{Effect}: \neg At(Spare, Ground) \wedge At(Spare, Axle))\\
&Action(LeaveOvernight,\\
&\quad \textsc{Precond}:\\
&\quad \textsc{Effect}: \neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)\\
&\qquad \wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))
\end{aligned}
$$

Example: The blocks world

This domain consists of a set of cube-shaped blocks sitting on a table.3 .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 it. The goal will always be to build one or more stacks of blocks, specified in terms of what blocks are on top of what other blocks. For example, a goal might be to get block A on B and block C on D.

$$
\begin{aligned}
&Init(On(A, Table) \wedge On(B, Table) \wedge On(C, Table)\\
&\quad \wedge Block(A) \wedge Block(B) \wedge Block(C)\\
&\quad \wedge Clear(A) \wedge Clear(B) \wedge Clear(C))\\
&Goal(On(A, B) \wedge On(B, C))\\
&Action(Move(b, x, y),\\
&\quad \textsc{Precond}: On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge\\
&\qquad (b \neq x) \wedge (b \neq y) \wedge (x \neq y),\\
&\quad \textsc{Effect}: On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))\\
&Action(MoveToTable(b, x),\\
&\quad \textsc{Precond}: On(b, x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x),\\
&\quad \textsc{Effect}: On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))
\end{aligned}
$$

PLANNING WITH STATE-SPACE SEARCH

The descriptions of actions in a planning problem specify both preconditions and effects, it is possible to search in either direction: either forward from the initial state or backward from the goal.

Forward state-space search

Planning with forward state-space search is similar to the problem-solving approach. It is sometimes called **progression** planning, because it moves in the forward direction. The problem's starts from initial state, considering sequences of actions until we find a sequence that reaches a goal state. The formulation of planning problems as state-space search problems is as follows:

_ The **initial state** of the search is the initial state from the planning problem. In general, each state will be a set of positive ground literals; literals not appearing are false.

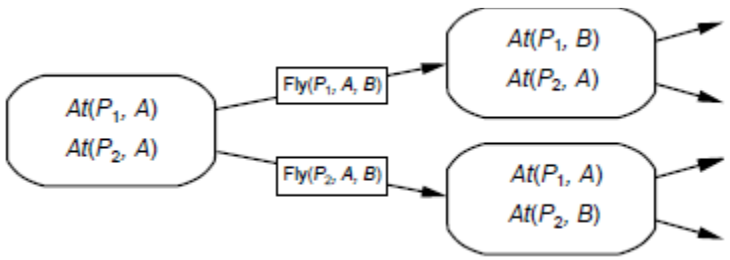
-The **actions** that are applicable to a state are all those whose preconditions are satisfied. The successor state resulting from an action is generated by adding the positive effect literals and deleting the negative effect literals.

_ The **goal test** checks whether the state satisfies the goal of the planning problem.

_ The **step cost** of each action is typically 1. Although it would be easy to allow different costs for different actions, this is seldom done by STRIPS planners.

Example

Consider an air cargo problem with 10 airports, where each airport has 5 planes and 20 pieces of cargo.The goal is to move all the cargo at airport A to airport B. There is a simple solution to the problem: load the 20 pieces of cargo into one of the planes at A, fly the plane to B, and unload the cargo. But finding the solution can be difficult because the average branching factor is huge: each of the 50 planes can fly to 9 other airports, and each of the 200 packages can be either unloaded (if it is loaded), or loaded into any plane at its airport (if it is unloaded).



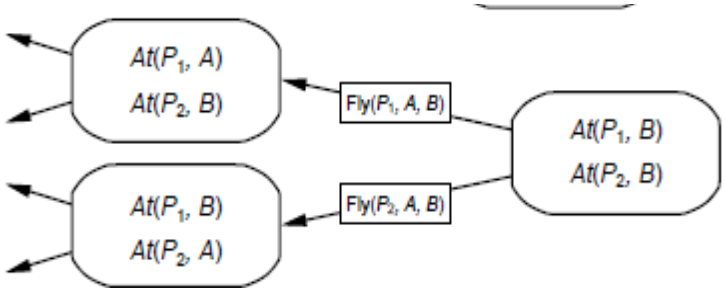
Backward state-space search

Backward search can be difficult to implement when the goal states are described by a set of constraints rather than being listed explicitly. In particular, it is not always obvious how to generate a description of the possible **predecessors** of the set of goal states.

The main advantage of backward search is that it allows us to consider only **relevant** actions. An action is relevant to a conjunctive goal if it achieves one of the conjuncts of the goal. For example, the goal in our 10-airport air cargo problem is to have 20 pieces of cargo at airport B, or more precisely,

At (C1;B) ^ At (C2;B) ^ : : : ^ At (C20;B) :

Now consider the conjunct At (C1;B). Working backwards, we can seek actions that have this as an effect. There is only one: Unload(C1; p;B), where plane p is unspecified.



PARTIAL ORDER PLANNING

Forward and backward state-space search are particular forms of *totally ordered* plan search.

Consider,the simple problem of putting on a pair of shoes. The formal planning problem as follows:

Goal (RightShoeOn ^ LeftShoeOn)
Init()
Action(RightShoe; PRECOND:RightSockOn; EFFECT:RightShoeOn)
Action(RightSock; EFFECT:RightSockOn)
Action(LeftShoe; PRECOND:LeftSockOn; EFFECT:LeftShoeOn)
Action(LeftSock; EFFECT:LeftSockOn)

A planner should be able to come up with the two-action sequence RightSock followed by RightShoe to achieve the first conjunct of the goal and the sequence LeftSock followed by LeftShoe for the second conjunct. Then the two sequences can be combined to yield the final plan.

In doing this, the planner will be manipulating the two subsequences independently, without committing to whether an action in one sequence is before or after an action in the other. Any planning algorithm that can place two actions into a plan without specifying which comes first is called a **partial-order planner**. Figure 11.6 shows the partial-order plan that is the solution to the shoes and socks problem. Note that the solution is represented as a *graph* of actions, not a sequence.

Note also the "dummy" actions called Start and Finish, which mark the beginning and end of the plan. The partial-order solution corresponds to six possible total-order plans; each of these is called a **linearization** of the partial-order plan.

Each plan has the following four components, where the first two define the steps of the plan and the last two serve a bookkeeping function to determine how plans can be extended:

_ A set of **actions** that make up the steps of the plan. These are taken from the set of actions in the planning problem. The "empty" plan contains just the Start and Finish actions. Start has no preconditions and has as its effect all the literals in the initial state of the planning problem. Finish has no effects and has as its preconditions the goal literals of the planning problem.

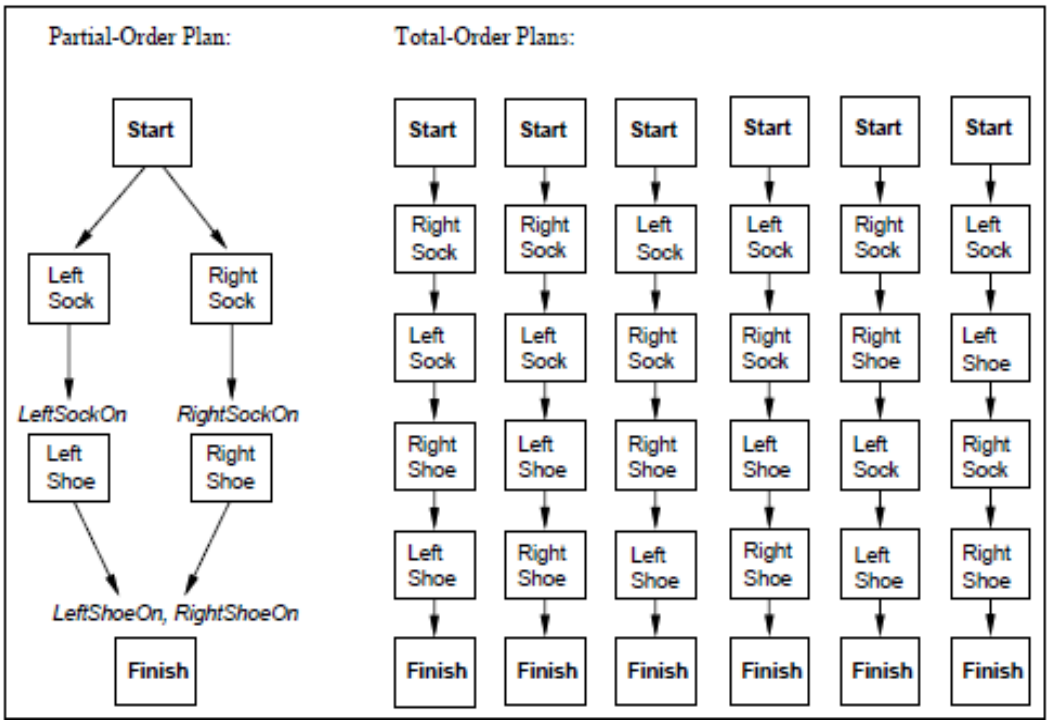
A set of **ordering constraints**. Each ordering constraint is of the form A_B, which is read as "A before B" and means that action A must be executed sometime before action B, but not necessarily immediately before. The ordering constraints must describe a proper partial order. Any cycle—such as A_B and B _A—represents a contradiction, so an ordering constraint cannot be added to the plan if it creates a cycle.

A set of **causal links**. A causal link between two actions A and B in the plan is written as A $\xrightarrow{p}$B and is read as "A **achieves** p for B." For example, the causal link

$$\text{RightSock} \xrightarrow{RightSockOn} \text{RightShoe}$$

asserts that RightSockOn is an effect of the RightSock action and a precondition of RightShoe.

A set of **open preconditions**. A precondition is open if it is not achieved by some action PRECONDITIONS in the plan. Planners will work to reduce the set of open preconditions to the empty set,without introducing a contradiction.



Partial Order Planning Example

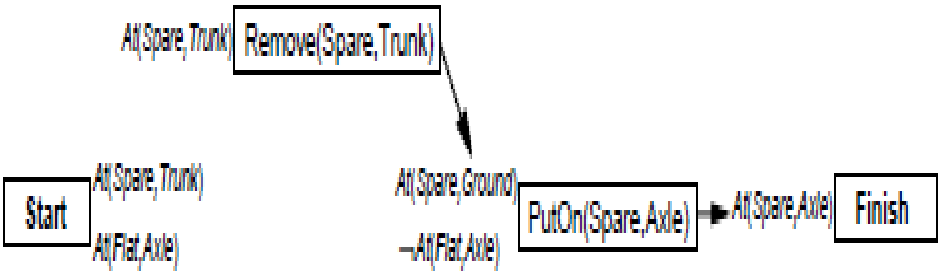
$$
\begin{aligned}
&Init(At(Flat, Axle) \wedge At(Spare, Trunk))\\
&Goal(At(Spare, Axle))\\
&Action(Remove(Spare, Trunk),\\
&\quad \text{PRECOND}: At(Spare, Trunk)\\
&\quad \text{EFFECT}: \neg At(Spare, Trunk) \wedge At(Spare, Ground))\\
&Action(Remove(Flat, Axle),\\
&\quad \text{PRECOND}: At(Flat, Axle)\\
&\quad \text{EFFECT}: \neg At(Flat, Axle) \wedge At(Flat, Ground))\\
&Action(PutOn(Spare, Axle),\\
&\quad \text{PRECOND}: At(Spare, Ground) \wedge \neg At(Flat, Axle)\\
&\quad \text{EFFECT}: \neg At(Spare, Ground) \wedge At(Spare, Axle))\\
&Action(LeaveOvernight,\\
&\quad \text{PRECOND}:\\
&\quad \text{EFFECT}: \neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)\\
&\qquad \wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))
\end{aligned}
$$

The search for a solution begins with the initial plan, containing a Start action with the effect At (Spare;Trunk) ^ At (Flat; Axle) and a Finish action with the sole precondition At (Spare; Axle). Then we generate successors by picking an open precondition to work and choosing among the possible actions to achieve it.

The sequence of events is as follows:

1. Pick the only open precondition, At (Spare; Axle) of Finish. Choose the only applicable action, PutOn(Spare; Axle).

2. Pick the At (Spare; Ground) precondition of PutOn(Spare; Axle). Choose the only applicable action, Remove(Spare;Trunk) to achieve it.

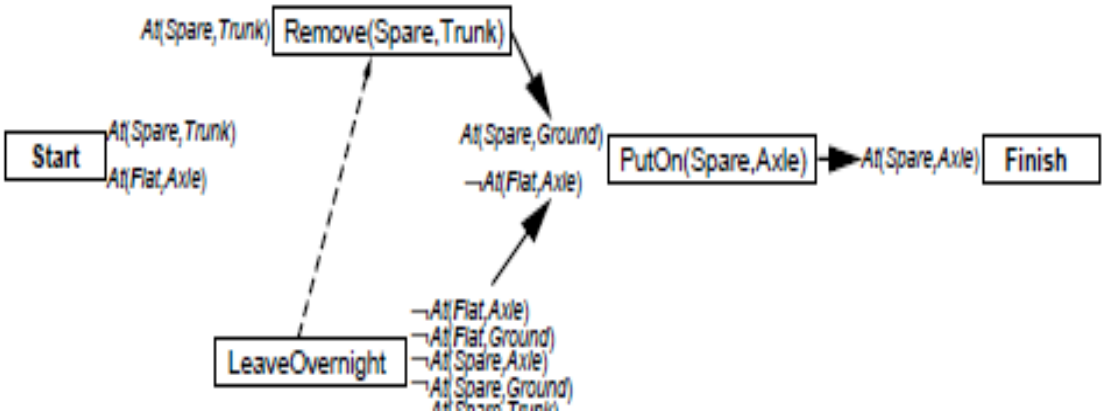


3. Pick the :At (Flat; Axle) precondition of PutOn(Spare; Axle). Just to be contrary,choose the LeaveOvernight action rather than the Remove(Flat; Axle) action. Notice that LeaveOvernight also has the effect :At (Spare; Ground), which means it conflicts with the causal link

Remove(Spare;Trunk) At(Spare;Ground)

PutOn(Spare; Axle) :

To resolve the conflict we add an ordering constraint putting LeaveOvernight before Remove(Spare;Trunk).



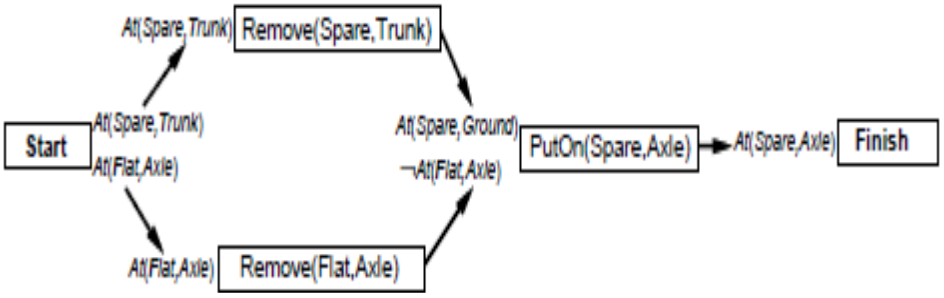
4. The only remaining open precondition at this point is the At (Spare;Trunk) precondition of the action Remove(Spare;Trunk). The only action that can achieve it is the existing Start action, but the causal link from Start to Remove(Spare;Trunk) is in conflict with the :At (Spare;Trunk) effect of LeaveOvernight. This time there is no way to resolve the conflict with LeaveOvernight:

we cannot order it before Start (because nothing can come before Start), and we cannot order it after Remove(Spare;Trunk) (because there is already a constraint ordering it before Remove(Spare;Trunk)). So we are forced to back up, remove the Remove(Spare;Trunk) action and the last two causal links.

5. Consider again the :At (Flat; Axle) precondition of PutOn(Spare; Axle). This time, we choose Remove(Flat; Axle).

6. Once again, pick the At (Spare; Tire) precondition of Remove(Spare;Trunk) and choose Start to achieve it. This time there are no conflicts.

7. Pick the At (Flat; Axle) precondition of Remove(Flat; Axle), and choose Start to achieve it.



PLANNING AND ACTING IN REAL WORLD

Time, Schedules and Resources

The STRIPS representation talks about *what* actions do, but, because the representation is based on situation calculus, it cannot talk about *how long* an action takes or even about *when* an action occurs. Time is of the essence in the general family of appllications called **job shop scheduling.** Such tasks require completing a set of jobs, each of whi.ch consists of a sequence of actions, where each action has a given duration and might require some resources.

The problem is to determine a schedule that minimizes the total time required to complete all the jobs, while respecting the resource constraints.This is a highly simplified automobile assembly problem. There are two jobs: assembling cars C1 and *C2.*

Each job consists of three actions: adding the engine, adding the wheels, and inspecting the results. The engine must be put in first (because having the front wheels on would inhibit access to the engine compartment) and of course the inspection must be done last.

$$
\begin{aligned}
&Init(Chassis(C_1) \wedge Chassis(C_2) \\
&\quad \wedge Engine(E_1, C_1, 30) \wedge Engine(E_2, C_2, 60) \\
&\quad \wedge Wheels(W_1, C_1, 30) \wedge Wheels(W_2, C_2, 15)) \\
&Goal(Done(C_1) \wedge Done(C_2)) \\
&Action(AddEngine(e,c), \\
&\quad \text{PRECOND}: Engine(e, c, d) \wedge Chassis(c) \wedge \neg EngineIn(c), \\
&\quad \text{EFFECT}: EngineIn(c) \wedge Duration(d)) \\
&Action(AddWheels(w, c), \\
&\quad \text{PRECOND}: Wheels(w, c, d) \wedge Chassis(c) \wedge EngineIn(c), \\
&\quad \text{EFFECT}: WheelsOn(c) \wedge Duration(d)) \\
&Action(Inspect(c), \text{PRECOND}: EngineIn(c) \wedge WheelsOn(c) \wedge Chassis(c), \\
&\quad \text{EFFECT}: Done(c) \wedge Duration(10))
\end{aligned}
$$

A *scheduling* problem rather than a *planning* problem,when each action should begin and end, based on the *durations* of actions as well as their ordering. The notation *Duration(d)* in the effect of an action (where *d* must be bound to a number) means that the action takes *d* minutes to complete.

critical path method (CPM) to determine the possible start and end times of each action.

A **path** through 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 have a window of time [ES,LS]

ES - earliest possible start time, LS - latest possible start time

Given A, B actions and $A \prec B$:

ES(Start) = 0

$ES(B) = \max_{A \prec B} ES(A) + Duration(A)$

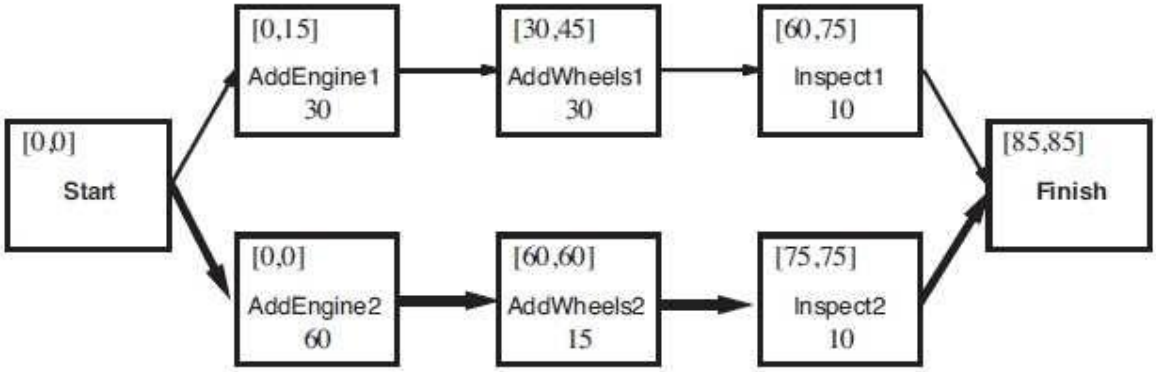
LS(Finish) = ES(Finish)

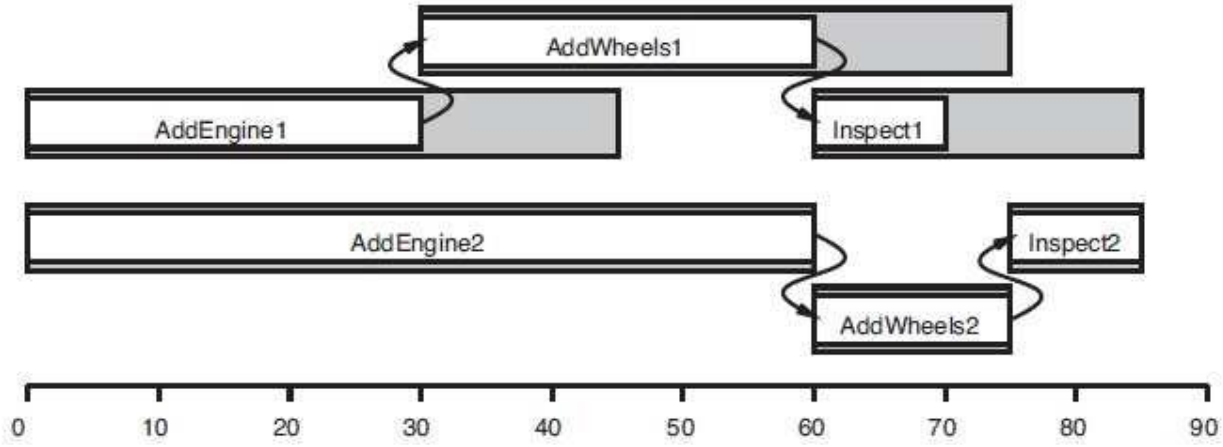
$LS(A) = \min_{B \succ A} LS(B) - Duration(A)$

Scheduling - No resource constraints

Partial order plan produced by e.g. POP

- To create a *schedule*, we must place actions on a timeline
- Can use *critical path* method (CPM): the longest path, no slack – determines total duration
- Shortest duration schedule, given partial-order plan: 85 minutes

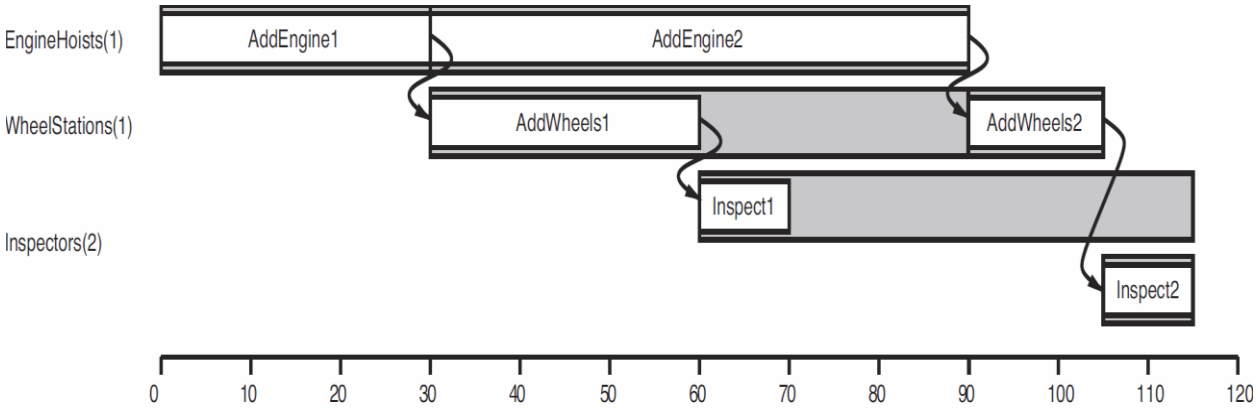




Scheduling with resource constraints

Actions typically require resources

- *Consumable* resources – e.g. *LugNuts*
- *Reusable* resources – e.g. *EngineHoists*
- Resource constraints make scheduling more complex because of interaction between actions
- AI and OR (Operations Research) methods can be used to solve scheduling problems with resources
- Shortest duration gone up from 85 to 115 minutes



Aggregation is essential for reducing complexity. Consider what happens when a schedule is proposed that 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.

Advantages

i) resource constraints make scheduling problems more complicated by introducing additional interactions among actions.

ii) Whereas unconstrained scheduling using the critical-path method is easy, finding a resource-constrained schedule with the earliest possible completion time is NP-hard.

$Init(Chassis(C_1) \wedge Chassis(C_2)$
$\wedge Engine(E_1, C_1, 30) \wedge Engine(E_2, C_2, 60)$
$\wedge Wheels(W_1, C_1, 30) \wedge Wheels(W_2, C_2, 15)$
$\wedge EngineHoists(1) \wedge WheelStations(1) \wedge Inspectors(2))$
$Goal(Done(C_1) \wedge Done(C_2))$

$Action(AddEngine(e, c),$
PRECOND: $Engine(e, c, d) \wedge Chassis(c) \wedge \neg EngineIn(c),$
EFFECT: $EngineIn(c) \wedge Duration(d),$
RESOURCE: $EngineHoists(1))$
$Action(AddWheels(w, c),$
PRECOND: $Wheels(w, c, d) \wedge Chassis(c) \wedge EngineIn(c),$
EFFECT: $WheelsOn(c) \wedge Duration(d),$
RESOURCE: $WheelStations(1))$
$Action(Inspect(c),$
PRECOND: $EngineIn(c) \wedge WheelsOn(c),$
EFFECT: $Done(c) \wedge Duration(10),$
RESOURCE: $Inspectors(1))$

Cannot overlap constraint: Disjunction of two linear inequalities, one for each possible ordering

Minimum slack algorithm:

- ♦ On each iteration, schedule for the earliest possible start whichever unscheduled action has all its predecessors scheduled and has the least slack;
- ♦ Then update the ES and LS times for each affected action and repeat.

Planning and scheduling

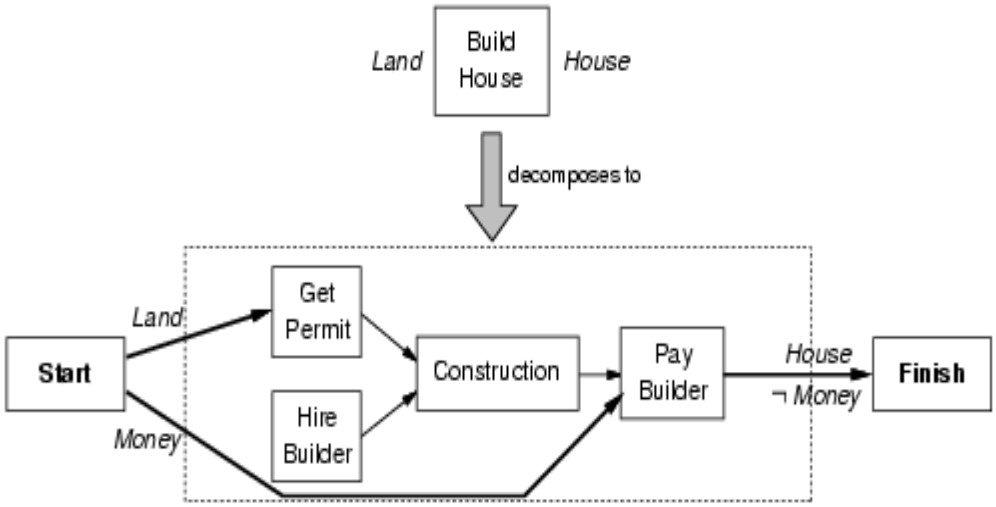
The approach shown here is common in real-world AI applications for manufacturing scheduling, airline scheduling, etc. :

- First generate partial order plan without timing information (*planning*)
- Then use separate algorithm to find optimal (or satisfactory) time behavior (*scheduling*)
- In some cases it may be better to *interleave* planning and scheduling, e.g. to consider temporal constraints already at the planning stage.
- Reduce complexity by decomposition

Often possible to reduce problem complexity by decompose to subproblems, solve independently, and assemble solution

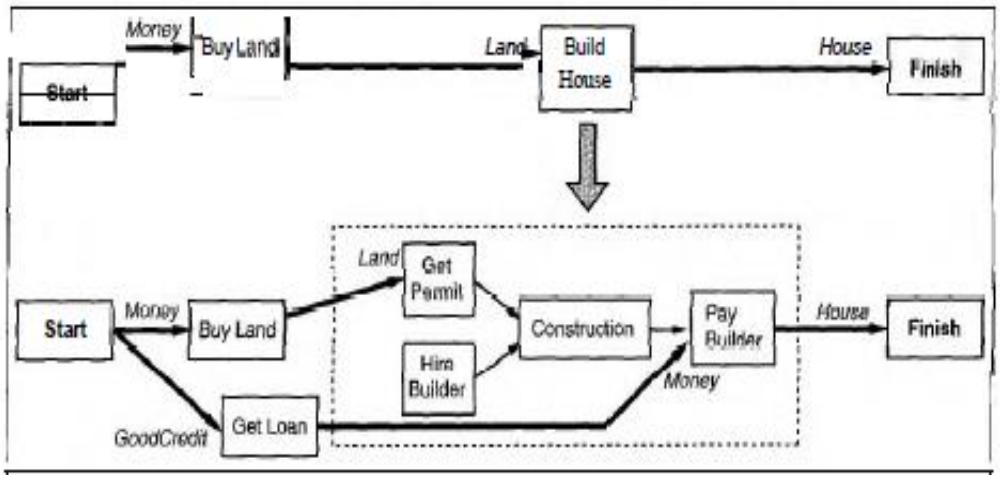
HTN – Hierarchical Task Networks

- Planner keeps library of subplans
- Extend planning algorithm to use subplans
- Can reduce time&space requirements considerably
- Most real-world planners use HTN variants
- Reduce complexity ⇒ hierarchical decomposition
- At each level of the hierarchy a computational task is reduced to a small number of activities at the next lower level.
- The computational cost of arranging these activities is low. ƒ
- Hierarchical task network (HTN) planning uses a refinement of actions through decomposition.
- e.g. building a house = getting a permit + hiring a contractor + doing the construction + paying the contractor.
- Refined until only primitive actions remain. ƒ
- Pure and hybrid HTN planning



Representation Decomposition

- General descriptions are stored in plan library.
- Each method = Decompos(a,d); a= action and d= PO plan. ƒ
- In buildhouse example , Start action supplies all preconditions of actions not supplied by other actions.
- external preconditions ƒ-Finish action has all effects of actions not present in other actions
- external effects - Primary effects (used to achieve goal) vs secondary effects



Properties of Decomposition

- Should be correct implementation of action a
- Correct if plan d is complete and consistent PO plan for the problem of achieving the effects of a given the preconditions of a.
- A decomposition is not necessarily unique.

Performs information hiding

- STRIPS action description of higher-level action hides some preconditions and effects
- Ignore all internal effects of decomposition
- Does not specify the intervals inside the activity during which preconditions and effects must hold.
- Information hiding is essential to HTN planning.

Planning in nondeterministic domains

Nondeterministic worlds

- *Bounded* nondeterminism: Effects can be enumerated, but agent cannot know in advance which one will occur
- *Unbounded* nondeteminism: The set of possible effects is unbounded or too large to enumerate
- Planning for bounded nondeterminism
- Sensorless planning
- Contingent planning
- Planning for unbounded nondeterminism
- Online replanning
- Continuous planning

Sensorless planning

Agent has no sensors to tell which state it is in, therefore each action might lead to one of several possible outcomes

- Must reason about *sets* of states (belief states), and make sure it arrives in a goal state regardless of where it comes from and results of actions
- Nondeterminism of the environment does not matter – the agent cannot detect the difference anyway
- The required reasoning is often not feasible, and sensorless planning is therefore often not applicable

Contingent planning

Constructs conditional plans with branches for each (enumerable) possible situation

- Decides which action to choose based on special sensing actions that become parts of the plan
- Can also tackle partially observable domains by including reasoning about belief states (as in sensorless planning)
- Planning algorithms have been extended to produce conditional branching plan

Online replanning

Monitors situation as plan unfold, detects when things go wrong

- Performs replanning to find new ways to reach goals, if possible by repairing current plan

- Agent proceeds from S, and next expects E following original *whole-plan*
- Detects that it's actually in O
- Creates a *repair* plan that takes it from O to a state P in original plan
- New plan to reach G becomes *repair + continuation*

Contingent planning vs. replanning

Contingent planning

- All actions in the real world have additional outcomes
- Number of possible outcomes grows exponentially with plan size, most of them are highly improbable
- Only one outcome will actually occur
- Replanning
- Basically assumes that no failure occurs
- Tries to fix problems as they occur
- May produce fragile plans, hard to fix if things go wrong

Conditional Planning

- Deal with uncertainty by checking the environment to see what is really happening.
- Used in fully observable and nondeterministic environments.
- The outcome of an action is unknown.
- Conditional steps will check the state of the environment. How to construct a conditional plan?
- Actions: left, right, suck
- Propositions to define states: AtL, AtR, CleanL, CleanR
- How to include nondeterminism?
- Actions can have more than one effect
- E.g. moving left sometimes fails - Action(Left, PRECOND: AtR, EFFECT: AtL)
- Becomes : Action(Left, PRECOND: AtR, EFFECT: AtL∨AtR)

- Actions can have conditional effects - Action(Left, PRECOND:AtR, - EFFECT: AtL∨(AtL∧when cleanL: ¬cleanL) - Both disjunctive and conditional

Conditional plans require conditional steps

- If then plan_A else plan_B if AtL∧CleanL then Right else Suck
- Plans become trees ƒ Games against nature
- Find conditional plans that work regardless of which action outcomes actually occur.
- Assume vacuum-world Initial state = AtR ∧ CleanL ∧ CleanR

Double murphy: possibility of desposit dirt when moving to other square and possibility of despositing dirt when action is Suck.

Continuous Planning:

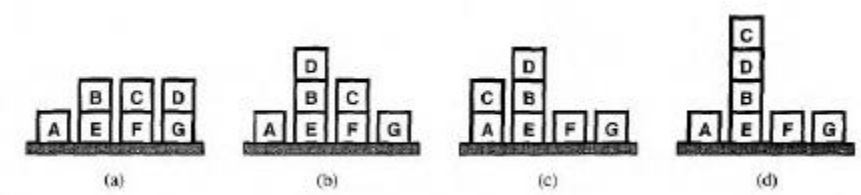
ƒ Agent persists indefinitely in an environment

- Phases of goal formulation, planning and acting ƒ Execution monitoring + planner as one continuous process ƒ

Example:Blocks world

- Assume a fully observable environment
- Assume partially ordered plan

Block World Example ƒ



Initial state

(a) Action(Move(x,y), PRECOND: Clear(x) ∧ Clear(y) ∧ On(x,z)

(b) EFFECT: On(x,y) ∧ Clear(z) ∧ ¬On(x,z) ∧ ¬Clear(y) ƒ The agent first needs to formulate a goal: On(C,D) ∧ On(D,B) ƒ Plan is created incrementally, return NoOp and check

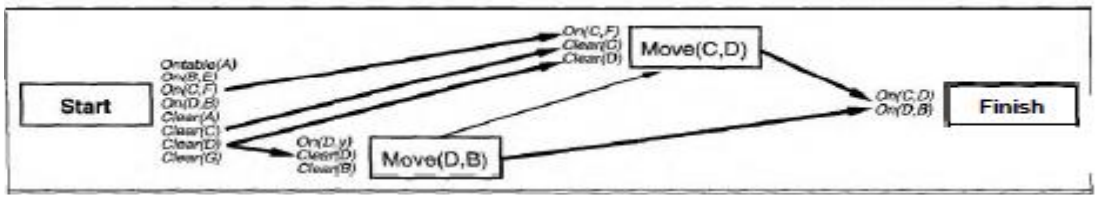
ƒ Assume that percepts don't change and this plan is constructed ƒ Ordering constraint between Move(D,B) and Move(C,D).Start is label of current state during planning. Before the agent can execute the plan, nature intervenes: D is moved onto B

- Start contains now On(D,B)
- Agent perceives: Clear(B) and On(D,G) are no longer true
- Update model of current state (start)
- Causal links from Start to Move(D,B) (Clear(B) and On(D,G)) no longer valid.
- Remove causal relations and two PRECOND of Move(D,B) are open
- Replace action and causal links to Finish by connecting Start to Finish.

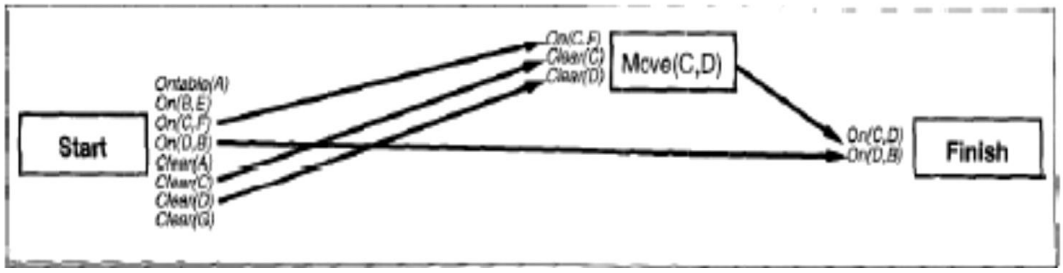
Extending: whenever a causal link can be supplied by a previous step All redundant steps (Move(D,B) and their causal links) are removed from the plan .Execute new plan, perform action Move(C,D) .This removes the step from the plan

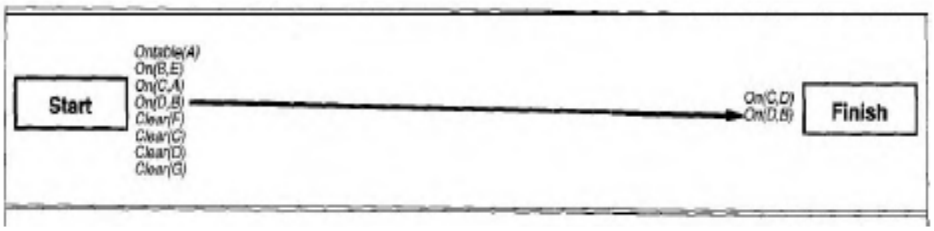
Execute new plan, perform action Move(C,D) .Assume agent is clumsy and drops C on A. No plan but still an open PRECOND. Determine new plan for open condition. Again Move(C,D)

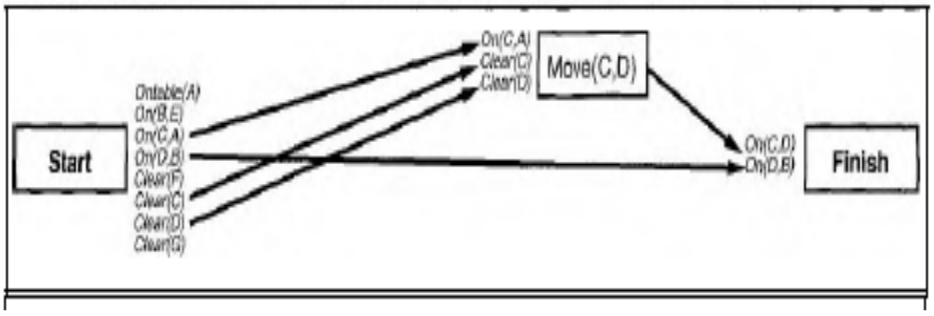
1. 

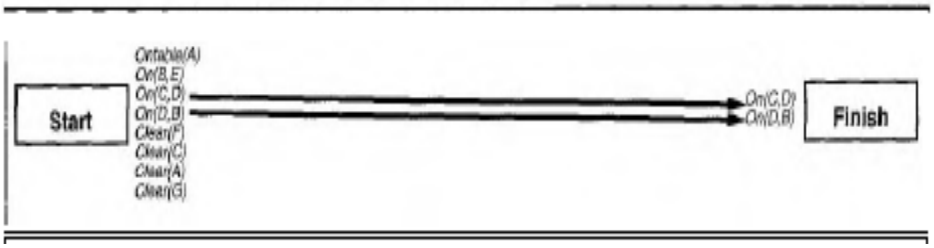
2. 

3.



4. 

5. 

6. 

Similar to POP, On each iteration find plan-flaw and fix it . Possible flaws: Missing goal, Open precondition, Causal conflict, Unsupported link, Redundant action, Unexecuted action, unnecessary historical goal

Poor performance since agents are not indifferent at other agents' intentions. In general two types of multiagent environments:

- Cooperative
- Competitive

Multi-planning problem:

Assume double tennis example where agents want to return ball.

Agents(A,B) Init(At(A,[Left,Baseline])∧ At(B,[Right, Net]) ∧ Approaching(Ball,[Right, Baseline]) ∧ PArtner(A,B) ∧ Partner(B,A)) Goal(Returned(Ball) ∧ At(agent,[x,Net])) Action(Hit(agent, Ball) PRECOND: Approaching(Ball,[x,y]) ∧ At(agent,[x,y]) ∧ Partner(agent, partner) ∧ ¬At(partner,[x,y])

EFFECT: Returned(Ball)) Action(Go(agent,[x,y])

PRECOND: At(agent,[a,b])

EFFECT: At(agent,[x,y]) ∧ ¬ At(agent,[a,b]))

Cooperation:

Joint Goals and Plans (II) ƒ A solution is a joint-plan consisting of actions for both agents. ƒ Example: A: [Go(A,[Right, Baseline]), Hit(A,Ball)] B: [NoOp(B), NoOp(B)] Or A: [Go(A,[Left, net), NoOp(A)] B: [Go(B,[Right, Baseline]), Hit(B, Ball)]

Coordination is required to reach same joint plan

Multi-Body Planning

Planning problem faced by a single centralized agent that can dictate action to each of several physical entities.

Assume for simplicity that every action takes one time step and at each point in the joint plan the actions are performed simultaneously [;]. Planning can be performed using POP applied to the set of all possible joint actions. - Size of this set???

Concurrent action Action(Hit(A, Ball)

CONCURRENT: ¬Hit(B,Ball)

PRECOND: Approaching(Ball,[x,y]) ∧ At(A,[x,y])

EFFECT: Returned(Ball))

Required actions (carrying object by two agents)

Action(Carry(A, cooler, here, there)

CONCURRENT: Carry(B,cooler, here there)

PRECOND: …) ƒ Planner similar to POP with some small changes in possible ordering relations

Coordination Mechanisms

To ensure agreement on joint plan: use convention.

Convention = a constraint on the selection of joint plans (beyond the constraint that the joint plan must work if the agents adopt it).

e.g. stick to your court or one player stays at the net. Conventions which are widely adopted= social laws e.g. language. Can be domain-specific or independent. Could arise through evolutionary process (flocking behavior).

Separation: Steer away from neighbors when you get too close

- Cohesion Steer toward the average position of neighbors
- Alignment Steer toward average orientation (heading) of neighbors
- Flock exhibits emergent behavior of flying as a pseudo-rigid body

Coordination Mechanisms

- In the absence of conventions: Communication e.g. Mine! Or Yours! in tennis example
- The burden of arriving at a succesfull joint plan can be placed on
- Agent designer (agents are reactive, no explicit models of other agents)
- Agent (agents are deliberative, model of other agents required)

Competitive Environments: Agents can have conflicting utilities e.g. zero-sum games like chess .The agent must:

• Recognise that there are other agents

• Compute some of the other agents plans

• Compute how the other agents interact with its own plan

• Decide on the best action in view of these interactions.

Model of other agent is required YET, no commitment to joint action plan.

ACTING UNDER UNCERTAINTY

Uncertainty is a situation which involves imperfect and/or unknown information. Uncertainty arises in partially observable and/or stochasticenvironments. A state of having limited knowledge where it is impossible to exactly describe the existing state, a future outcome, or more than one possible outcome.

Let action A_t = leave for airport t minutes before flight.

Will A_t get me there on time?

Problems:

1. partial observability (road state, other drivers' plans, etc.)
2. noisy sensors (traffic reports)
3. uncertainty in action outcomes (flat tire, etc.)
4. immense complexity of modeling and predicting traffic

Hence a purely logical approach either

1. risks falsehood: “A25 will get me there on time”, or
2. leads to conclusions that are too weak for decision making:

“A25 will get me there on time if there's no accident on the bridge and it doesn't rain and my tires remain intact etc etc.” (A1440 might reasonably be said to get me there on time but I'd have to stay overnight in the airport.

BAYES’S RULES

Flip a coin. What is the chance of it landing heads side up? There are 2 sides, heads and tails. The heads side is one of them, thus

P(Heads) = $\frac{\text{Heads side}}{\text{Heads side + Tails side}}$

P(Heads) = 1/(1+1) = ½ = .50 = 50%

Throw a die. What is the probability of getting a 3.

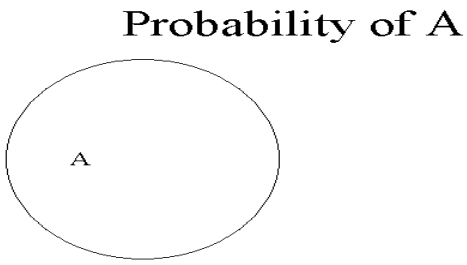
P(die = 3) = $\frac{\text{1 side with a 3}}{\text{6 sides total}}$

P(die = 3) = 1/6 = .167 = 16.7%

Conditional Probability

Events A and B are events that are not mutually exclusive, but occur conditionally on the occurrence of one another.

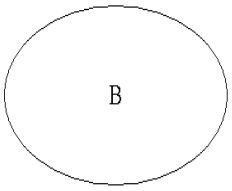
The probability that event A will occur:p(A)



The probability that event B will occur:

p(B)

Probability of B



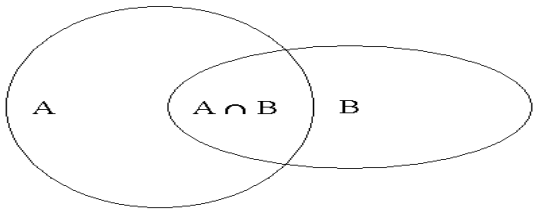
The number of times that both A and B occur or the probability that both events A and B will occur is called the **joint probability.**

Mathematically joint probability is defined as:

$p(A \cap B)$

i.e. the probability that both A and B will occur

Joint probability of A and B



The probability that event A will occur if event B occurs is called **conditional probability.**

$$P(A|B) = \frac{\text{the number of times A and B can occur}}{\text{the number of times B can occur}}$$

or

$$P(A|B) = \frac{P(A \text{ and } B)}{P(B)}$$

or

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Lets take an example.

Suppose we have 2 dice and we want to know what the probability of getting an 8 is. Normally if we roll both at the same time the probability is 5/36.

1,1	1,2	1,3	1,4	1,5	1,6
2,1	2,2	2,3	2,4	2,5	2,6
3,1	3,2	3,3	3,4	3,5	3,6
4,1	4,2	4,3	4,4	4,5	4,6
5,1	5,2	5,3	5,4	5,5	5,6
6,1	6,2	6,3	6,4	6,5	6,6

But what happens if we roll the first die and get a 5, now what is the probability of getting an 8?

There is only one way to get an 8 after a 5 has been rolled. You have to roll a 3.

Looking at the formula:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Lets rephrase it for our problem:

$$P(\text{getting an 8 using 2 die} \mid \text{given that we roll a 5 with first dice}) = \frac{P(\text{rolling 5 and 3})}{P(\text{rolling a 5})}$$

$P(A|B) = (1/36) / (1/6) = (1/36) * (6/1) = 6/36 = 1/6$

So the chances improve slightly $1/6 > 5/36$ by only 1/36.

Why do they improve, but only slightly?

An intuitive explanation is that the only way to not be able to get an 8 using two dice rolled in sequence (one then the other) is if the first die is a 1. The chance of that is only 1/6. So it would not really matter what the first die is, as long as is not a 1, you can still get an 8 with the combination of the two.

Bayes theorem is an algebraic rewriting of conditional probability.

Starting with conditional probability:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(A \cap B) = P(A|B) * P(B)$$

Recognizing that the intersection of A and B is communitve:

$$P(A \cap B) = P(B \cap A)$$

And that the following is true:

$$P(B|A) = \frac{P(B \cap A)}{P(A)}$$

$$P(B \cap A) = P(B|A) * P(A)$$

Remembering that the intersection is communitve we can form the following equation:

$$P(A \cap B) = P(B|A) * P(A)$$

Now this can be substituted into the conditional probability equation:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Yielding Bayes theorem:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

We like to rewrite it using h and D, h being a hypotheses that we are testing and D being data that we want to support our hypotheses.

Bayes theorem:

$$p(h|D) = \frac{p(D|h) * p(h)}{p(D)}$$

P(h|D) - the probability that hypotheses *h* is true given the data *D*.

- Often referred to as the ***posterior probability***. It reflects our confidence that *h* is true after we have seen the data *D*.
- The posterior probability reflects the influence of the data *D* while the prior probability *P(h)* does not, it is independent of *D*.

P(D|h) – The probability that data *D* exists in when hypotheses *h* is true.

- Think of *h* as the answer to what happened in a murder case. One way to think of this is that if h is true, what is the probability that the data (evidence) *D* will exist.
- *D* is the evidence that backs up the case, it is what proves that *h* is true.
- In general *P(x|y)* denotes the probability of x given y. It is the conditional probability as discussed above.

P(h) – initial probability that hypotheses *h* holds, before we have observed the data.

- Often called the ***prior probability*** of *h*. It will reflect any background knowledge that we have about the correctness of hypothesis *h*.
- If we have no initial knowledge about the hypotheses ($h_0 ..... h_n$), we would divide the probability equally among the set of available hypotheses (in which *h* is a member).

P(D) – The prior probability that the data *D* will be observed (this the probability of *D* given no prior knowledge that *h* will hold). Remember that it is completely independent of *h.*

Looking at the equation we can make some observations:

- *P(h|D)*, the probability of h being true given the presence of D increases with commonness of h being true independently.
- *P(h|D)*, the probability of h being true given the presence of *D* increases with the likelihood of data *D* being associated with hypotheses *h*. That is the higher our confidence is in saying that data *D* is present only when *h* is true, the more we can say for our hypothesis depends on *D.*
- When *p(D)* is high that means our evidence is likely to exist independently of *h*, so it weakens the link between *h* and *D*.

Bayes Rule in AI

Brute Force Bayes Algorithm

Here is a basic Bayes rule decision algorithm:

- Initialize H a set of hypothesis such that $h_0 \ldots\ldots h_n \in H$
- For each h_i in H calculate the posterior probability

$$p(h_i|D) = \frac{p(D|h_i) * p(h_i)}{p(D)}$$

- Output MAX($h_0 \ldots\ldots h_n$)

Naïve Bayes Classifier

This classifier applies to tasks in which each example is described by a conjunction of attributes and the target value f(x) can take any value from the set of v.

A set of training examples for f(x) is provided.

In this example we want to use Bayes theorem to find out the likelihood of playing tennis for a given set weather attributes.

$f(x) \in v$ = (yes, no) i.e. v = (yes we will play tennis, no we will not play tennis)

The attribute values are $a_0 \ldots a_3$ = (Outlook, Temperature, Humidity, and Wind).

To determine our answer (if we are going to play tennis given a certain set of conditions) we make an expression that determines the probability based on our training examples from the table.

Day	**Outlook**	**Temperature**	**Humidity**	**Wind**	**Play Tennis**
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes

10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Remembering Bayes Rule:

$$p(h|D) = \frac{p(D|h) * p(h)}{p(D)}$$

We write our f(x) in that form:

$$P(\text{Play Tennis} \mid \text{Attributes}) = \frac{P(\text{Attributes} \mid \text{Play Tennis}) * P(\text{Play Tennis})}{P(\text{Attributes})}$$

Or

$$P(v|a) = \frac{P(a|v) * P(v)}{P(a)}$$

Lets look closely at P(a|v)

$$P(a|v) = P(a_0 \ldots a_3 \mid v_{0,1})$$

Or

P(a|v) = P(Outlook, Temperature, Humidity, Wind | Play tennis, Don't Play tennis)

In order to get a table with reliable measurements every combination of each attribute $a_0 \ldots a_3$ for each hypotheses $v_{0,1}$ our table would have be of size 3*3*2*2*2 = 72 and each combination would have to be observed multiple times to ensure its reliability. Why, because we are assuming an inter-dependence of the attributes (probably a good assumption). ***The Naïve Bayes classifier is based on simplifying this assumption.*** That is to say, cool temperature is completely independent of it being sunny and so on.

So :

$$P(a_0 \ldots a_3 \mid v_{j=0,1}) \approx P(a_0|v_0) * P(a_1|v_0) * P(a_n|v_0)$$
$$\approx P(a_0|v_1) * P(a_1|v_1) * P(a_n|v_1)$$

or

$$P(a_0 \ldots a_n \mid v_j) \approx \prod_i P(a_i \mid v_j)$$

A more concrete example:

P(outlook = sunny, temperature = cool, humidity = normal, wind = strong | Play tennis) ≈

P(outlook = sunny | Play tennis) * P(temperature = cool | Play tennis) *
P(humidity = normal | Play tennis) * P(wind = strong | Play tennis)

The probability of observing $P(a_0 \ldots a_n \mid v_j)$ is equal the product of probabilities of observing the individual attributes. Quite an assumption.

Using the table of 14 examples we can calculate our overall probabilities and conditional probabilities.

First we estimate the probability of playing tennis:

P(Play Tennis = Yes) = 9/14 = .64

P(Play Tennis = No) = 5/14 = .36

Then we estimate the conditional probabilities of the individual attributes. Remember this is the step in which we are assuming that the attributes are independent of each other:

Outlook:

P(Outlook = Sunny | Play Tennis = Yes) = 2/9 = .22

P(Outlook = Sunny | Play Tennis = No) = 3/5 = .6

P(Outlook = Overcast | Play Tennis = Yes) = 4/9 = .44

P(Outlook = Overcast | Play Tennis = No) = 0/5 = 0

P(Outlook = Rain | Play Tennis = Yes) = 3/9 = .33

P(Outlook = Rain | Play Tennis = No) = 2/5 = .4

Temperature

P(Temperature = Hot | Play Tennis = Yes) = 2/9 = .22

P(Temperature = Hot | Play Tennis = No) = 2/5 = .40

P(Temperature = Mild | Play Tennis = Yes) = 4/9 = .44

P(Temperature = Mild | Play Tennis = No) = 2/5 = .40

P(Temperature = Cool | Play Tennis = Yes) = 3/9 = .33

P(Temperature = Cool | Play Tennis = No) = 1/5 = .20

Humidity

P(Humidity = Hi | Play Tennis = Yes) = 3/9 = .33

P(Humidity = Hi | Play Tennis = No) = 4/5 = .80

P(Humidity = Normal | Play Tennis = Yes) = 6/9 = .66

P(Humidity = Normal | Play Tennis = No) = 1/5 = .20

Wind

P(Wind = Weak | Play Tennis = Yes) = 6/9 = .66

P(Wind = Weak | Play Tennis = No) = 2/5 = .40

P(Wind = Strong | Play Tennis = Yes) = 3/9 = .33

P(Wind = Strong | Play Tennis = No) = 3/5 = .60

Suppose the day is described by :

a = (Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong)

What would our Naïve Bayes classifier predict in terms of playing tennis on a day like this?

Which ever equation has the higher probability (greater numerical value)

P(Playtennis = Yes | (Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong))

Or

P(Playtennis = No | (Outlook = sunny, Temperature = cool, Humidity = high, Wind = strong))

Is the prediction of the Naïve Bayes Classifier. *(for brevity we have omitted the attribute names)*

Working through the first equation….

$$P(Yes|(sunny, cool, high, strong)) = \frac{P((sunny, cool, high, strong) | Yes) * P(Yes)}{P(sunny, cool, high, strong)}$$

Now we do the independent substitution for:

P((sunny…..)|Yes)

And noting that the denominator, P(sunny, cool, high, strong), includes both:

P((sunny, cool, high, strong) | Yes) and

P((sunny, cool, high, strong) | No)

our equation expands to:

$$= \frac{P(sunny|Yes)*P(cool|Yes)*P(high|Yes)*P(strong|Yes) * P(yes)}{P((sunny, cool, high, strong) | Yes) + P((sunny, cool, high, strong) | No)}$$

Remember the quantities in the denominator are expanded using the independent assumption in a similar way that the first term in the numerator.

$$= \frac{(.22 * .33 * .33 * .33) * .64}{(.22 * .33 * .33 * .33)*64 + (.6 * .2 * .8 * .6)*.36}$$

$$= \frac{.0051}{.0051 + .0207}$$

$$= .1977$$

Working through the second equation in a similar fashion….

$$P(No|(sunny, cool, high, strong)) = \frac{P((sunny, cool, high, strong) | No) * P(No)}{P(sunny, cool, high, strong)}$$

$$= \frac{.0207}{.0051 + .0207}$$

$$= .8023$$

As we can see, the Bayes Naïve classifier gives a value of just about 20% for playing tennis in the described conditions, and value of 80% for not playing tennis in these conditions, therefore the prediction is that no tennis will be played if the day is like these conditions.

SEMANTICS OF BELIEF NETWORKS

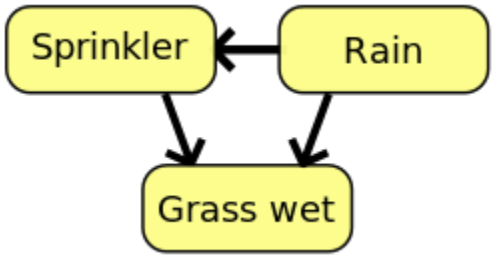
A **Bayesian network**, **Bayes network**, **belief network**, **Bayes(ian) model** or**probabilistic directed acyclic graphical model** is a probabilistic graphical model(a type of statistical model) that represents a set of random variables and theirconditional dependencies via a directed acyclic graph (DAG).

For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Formally, Bayesian networks are DAGs whose nodes represent random variables in the Bayesian sense: they may be observable quantities, latent variables, unknown parameters or hypotheses. Edges represent conditional dependencies; nodes that are not connected (there is no path from one of the variables to the other in the bayesian network) represent variables that are conditionally independent of each other. Each node is associated with a probability function that takes, as input, a particular set of values for the node's parent variables, and gives (as output) the probability (or probability distribution, if applicable) of the variable represented by the node.

For example, if parent nodes represent Boolean variables then the probability function could be represented by a table of entries, one entry for each of the possible combinations of its parents being true or false. Similar ideas may be applied to undirected, and possibly cyclic, graphs; such are called Markov networks.

Suppose that there are two events which could cause grass to be wet: either the sprinkler is on or it's raining. Also, suppose that the rain has a direct effect on the use of the sprinkler (namely that when it rains, the sprinkler is usually not turned on). Then the situation can be modeled with a Bayesian network (shown to the right). All three variables have two possible values, T (for true) and F (for false).



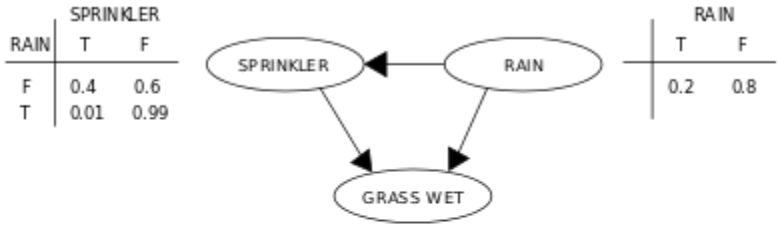
The names of the variables have been abbreviated to G = *Grass wet (yes/no)*, S = *Sprinkler turned on (yes/no)*, and R = *Raining (yes/no)*.

The model can answer questions like "What is the probability that it is raining, given the grass is wet?" by using theconditional probability formula and summing over all nuisance variables:

Using the expansion for the joint probability function and the conditional probabilities from the conditional probability tables (CPTs) stated in the diagram, one can evaluate each term in the sums in the numerator and denominator.

If, on the other hand, we wish to answer an interventional question: "What is the probability that it would rain, given that we wet the grass?" the answer would be governed by the post-intervention joint distribution function obtained by removing the factor from the pre-intervention distribution. As expected, the probability of rain is unaffected by the action: .

These predictions may not be feasible when some of the variables are unobserved, as in most policy evaluation problems. The effect of the action can still be predicted, however, whenever a criterion called "back-door" is satisfied. It states that, if a set Z of nodes can be observed that d-separates[3] (or blocks) all back-door paths from X to Y then . A back-door path is one that ends with an arrow into X. Sets that satisfy the back-door criterion are called "sufficient" or "admissible." For example, the set $Z = R$ is admissible for predicting the effect of $S = T$ on G, because R d-separate the (only) back-door path $S \leftarrow R \rightarrow G$. However, if S is not observed, there is no other set that d-separates this path and the effect of turning the sprinkler on ($S = T$) on the grass (G) cannot be predicted from passive observations. We then say that $P(G \mid \text{do}(S = T))$ is not "identified." This reflects the fact that, lacking interventional data, we cannot determine if the observed dependence between S and G is due to a causal connection or is spurious (apparent dependence arising from a common cause, R).



RAIN	SPRINKLER T	SPRINKLER F
F	0.4	0.6
T	0.01	0.99

RAIN T	RAIN F
0.2	0.8

SPRINKLER	RAIN	GRASS WET T	GRASS WET F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

To determine whether a causal relation is identified from an arbitrary Bayesian network with unobserved variables, one can use the three rules of "*do*-calculus" and test whether all *do* terms can be removed from the expression of that relation, thus confirming that the desired quantity is estimable from frequency data.[5]

Using a Bayesian network can save considerable amounts of memory, if the dependencies in the joint distribution are sparse. For example, a naive way of storing the conditional probabilities of 10 two-valued variables as a table requires storage space for

values. If the local distributions of no variable depends on more than three parent variables, the Bayesian network representation only needs to store at most values. One advantage of Bayesian networks is that it is intuitively easier for a human to understand (a sparse set of) direct dependencies and local distributions than complete joint distributions.

INFERENCE IN BELIEF NETWORKS

Because a Bayesian network is a complete model for the variables and their relationships, it can be used to answer probabilistic queries about them. For example, the network can be used to find out updated knowledge of the state of a subset of variables when other variables (the *evidence* variables) are observed. This process of computing the *posterior* distribution of variables given evidence is called probabilistic inference. The posterior gives a universal sufficient statistic for detection applications, when one wants to choose values for the variable subset which minimize some expected loss function, for instance the probability of decision error. A Bayesian network can thus be considered a mechanism for automatically applying Bayes' theorem to complex problems.

The most common exact inference methods are: variable elimination, which eliminates (by integration or summation) the non-observed non-query variables one by one by distributing the sum over the product; clique tree propagation, which caches the computation so that many variables can be queried at one time and new evidence can be propagated quickly; and recursive conditioning and AND/OR search, which allow for a space-time tradeoff and match the efficiency of variable elimination when enough space is used. All of these methods have complexity that is exponential in the network's treewidth. The most common approximate inference algorithms are importance sampling, stochastic MCMC simulation, mini-bucket elimination, loopy belief propagation, generalized belief propagation, and variational methods.

```
function ENUMERATION-ASK(X, e, bn) returns a distribution over X
  inputs: X, the query variable
          e, observed values for variables E
          bn, a Bayes net with variables {X} ∪ E ∪ Y   /* Y = hidden variables */

  Q(X) ← a distribution over X, initially empty
  for each value x_i of X do
      extend e with value x_i for X
      Q(x_i) ← ENUMERATE-ALL(VARS[bn], e)
  return NORMALIZE(Q(X))
```

```
function ENUMERATE-ALL(vars, e) returns a real number
  if EMPTY?(vars) then return 1.0
  Y ← FIRST(vars)
  if Y has value y in e
      then return P(y | parents(Y)) × ENUMERATE-ALL(REST(vars), e)
      else return Σ_y P(y | parents(Y)) × ENUMERATE-ALL(REST(vars), e_y)
          where e_y is e extended with Y = y
```

$$\mathbf{P}(B|j,m) = \alpha \underbrace{\mathbf{P}(B)}_{B} \sum_{e} \underbrace{P(e)}_{E} \sum_{a} \underbrace{\mathbf{P}(a|B,e)}_{A} \underbrace{P(j|a)}_{J} \underbrace{P(m|a)}_{M} .$$

$$\mathbf{f}_M(A) = \begin{pmatrix} P(m|a) \\ P(m|\neg a) \end{pmatrix}$$

$$\begin{aligned} \mathbf{f}_{\bar{A}JM}(B,E) &= \sum_a \mathbf{f}_A(a,B,E) \times \mathbf{f}_J(a) \times \mathbf{f}_M(a) \\ &= \mathbf{f}_A(a,B,E) \times \mathbf{f}_J(a) \times \mathbf{f}_M(a) \\ &\quad + \mathbf{f}_A(\neg a,B,E) \times \mathbf{f}_J(\neg a) \times \mathbf{f}_M(\neg a) . \end{aligned}$$

MAKING SIMPLE DECISIONS

Combining Beliefs & Desires , Rational Decision

– based on Beliefs & Desires

– where uncertainty & conflicting goals exist.

An agent's preferences are captured by a utility function U which maps a state S to a number U(S) describing the desirability of S.

A nondeterministic action A may have several outcome states Resulti(A) indexed by the different outcomes of A. Prior to executing an action A, the agent assigns a probability P(Resulti(A) | Do(A),E)) to each outcome.

EU(A| E) P(Resulti(A) | E, Do(A)) U(Resulti(A))

Maximum Expected Utility, Expected Utility= EU(A| E) P(Resulti(A) | E, Do(A)) U(Resulti(A))

Maximum Expected Utility (MEU) principle – Choose an action which maximize agent's expected utility.If the agent's utility function U correctly reflects its performance

measure, then it will achieve the highest possible performance averaged over the environments in which it could be placed.

Simple decision – concerned only with single or one-shot decisions

The basis of Utility Theory

As a justification for the MEU principle, some constraints are imposed on the preferences that a rational agent should possess. „ In utility theory, different attainable outcomes (prizes) and the respective probabilities (chances) are formalized as lotteries:

– A lottery L having outcomes A1,..., An with probabilities p1+...+pn =1 is denoted [p1, A1;…; pn, An]. – A lottery [1,A] with a single outcome A is abbreviated as A. □ Preference relations for lotteries (or states) A and B: – A > B ⇔ the agent prefers A to B – A ~ B ⇔ the agent is indifferent between A and B – A ≥ B ⇔ the agent prefers A to B or is indifferent

Axioms of utility theory □ Orderability: (A > B) ∨ (A < B) ∨ (A ~ B) □

Transitivity: (A > B) ∧ (B>C) ⇒ (A>C). □

Continuity: If A>B>C then ∃p[p,A; 1-p,C] ~ B □

Substitutability: – If A ~ B then [p, A; 1-p, C] ~ [p, B; 1-p, C] □

Monotonicity: If A is better than B, then a lottery that differs only in assigning higher prob to A is better: p ≥q ⇔ [p, A; 1-p, B] ≥ [q, A; 1-q, B] □

Decomposability: Compound lotteries can be reduced using the laws of probability (“no fun in gambling” – you don’t care whether you gamble once or twice, as long as the results are the same)

Utility principle : Utility principle, Maximum Expected Utility principle ,

Utility Function – Represents what the agent’s actions are trying to achieve. – Can be constructed by observing agent’s preferences. U A >U B ⇔A> B

U(A) =U(B) ⇔A ~B

Decision Networks: Represents information about the agent’s current state, its possible actions, the state that will result from the agent’s action, and the utility of that state.

Belief network + decision & utility node.

Nodes – Chance node : represent random variables

Decision node : represent points where the decision-maker has a choice of actions

Utility node : represent the agent’s utility function

Evaluating decision networks 1. set the evidence variables for the current state 2. For each possible value of the decision node (a) Set the decision node to that value (b) Calculate the posterior probabilities for the parent nodes of the utility node, using a standard probabilistic inference algorithm. (c) Calculate the resulting utility for the action 3. Return the action with the highest utility

The value of information. Not all available information is provided to the agent before it makes its decision. One of the most important parts of decision making is knowing what questions to ask.

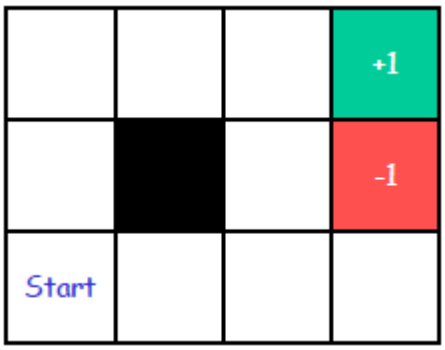
To conduct expensive and critical tests or not depends on two factors:

– Whether the different possible outcomes would make a significant difference to the optimal course of action

– The likelihood of the various outcomes „ Information value theory enables an agent to choose what information to acquire.

MAKING COMPLEX DECISIONS

Suppose that an agent is situated in the 4 x **3** environment shown in below Figure . Beginning in the start state, it must choose an action at each time step. The interaction with the environment terminates when the agent reaches one of the goal states, marked +1 or -1. In each location, the available actions are called Up, *Down, Left,* and *Right.* The environment is **fully observable,** so that the agent always knows where it is.



If the environment were deterministic, a solution would be easy: the agent will always reach +1 with moves [U, U, R, R, R]. Because actions are unreliable, a sequence of moves will

not always leads to the desired outcome. Let each action achieve the intended effect with probability 0.8. But with probability 0.1 the action moves the agent to either of the right angles to the intended direction. If the agent bumps into a wall, it stays in the same square. Now the sequence [U, U, R, R, R] leads to the goal state with probability 0.85 = 0.32768. In addition, the agent has a small chance of reaching the goal by accident going the other way around the obstacle with a probability 0.14 × 0.8, for a grand total of 0.32776.

A *transition model* specifies outcome probabilities for each action in each possible state

- Let P(s' | s, a) denote the probability of reaching state s' if action a is done in state s
- The transitions are *Markovian* in the sense that the probability of reaching s' depends only on s and not the earlier states
- To specify the utility function for the agent
- The decision problem is sequential, so the utility function depends on a sequence of states
- For now, we will simply stipulate that is each state s, the agent receives a *reward* R(s), which may be positive or negative

For our particular example, the reward is -0.04 in all states except in the terminal states

- The utility of an environment history is just (for now) the sum of rewards received
- If the agent reaches the state +1, e.g., after ten steps, its total utility will be 0.6
- The small negative reward gives the agent an incentive to reach [4, 3] quickly
- A sequential decision problem for a fully observable environment
- A Markovian transition model and
- Additive rewards is called a *Markov decision problem* (MDP)
- An MDP is defined by the following four components:
    - Initial state s0,
    - A set Actions(s) of actions in each state,
    - Transition model P(s' | s, a), and
    - Reward function R(s)

• As a solution to an MDP we cannot take a fixed action sequence, because the agent might end up in a state other than the goal

• A solution must be a policy, which specifies what the agent should do for any state that the agent might reach

• The action recommended by policy for state s

• If the agent has a complete policy, then no matter what the outcome of any action, the agent will always know what to do next.Each time a given policy is executed starting from the initial state, the stochastic nature of the environment will lead to a different environment history

• The quality of a policy is therefore measured by the expected utility of the possible environment histories generated by the policy

• An optimal policy yields the highest expected utility

• A policy represents the agent function explicitly and is therefore a description of a simple reflex agent

Utilities over time

• In case of an *infinite horizon* the agent's action time has no upper bound

• With a finite time horizon, the optimal action in a given state could change over time — the optimal policy for a finite horizon is *nonstationary*

• With no fixed time limit, on the other hand, there is no reason to behave differently in the same state at different times, and the optimal policy is stationary

• The *discounted* utility of a state sequence s0, s1, s2, … is R(s0) + R(s1) + R(s2) + …,

• If an infinite horizon environment does not contain a terminal state or if the agent never reaches one, then all environment histories will be infinitely long

• Then, utilities with additive rewards will generally be infinite

Value Iteration

• For calculating an optimal policy we calculate the utility of each state and then use the state utilities to select an optimal action in each state

• The utility of a state is the expected utility of the state sequence

that might follow it Obviously, the state sequences depend on the policy that is executed

• Let st be the state the agent is in after executing

• Note that st is a random variable

• R(s) is the short-term reward for being in s, whereas U(s) is the long-term total reward from s onwards

• In our example grid the utilities are higher for states closer to the +1 exit, because fewer steps are required to reach the exit

The Bellman equations for utilities

• The agent may select actions using the MEU principle

□*(s) = arg maxa □s‘ P(s’ | s, a) U(s') (*)

• The utility of state s is the expected sum of discounted rewards

from this point onwards, hence, we can calculate it:

• Immediate reward in state s, R(s)

+ The expected discounted utility of the next state, assuming

that the agent chooses the optimal action

U(s) = R(s) + maxa s‘ P(s’ | s, a) U(s')

• This is called the Bellman equation

• If there are n possible states, then there are n Bellman equations,

one for each state

U(1,1) = -0.04 + max{ 0.8 U(1,2) + 0.1 U(2,1) + 0.1 U(1,1), (U) 0.9 U(1,1) + 0.1 U(1,2), (L)

0.9 U(1,1) + 0.1 U(2,1), (D)

0.8 U(2,1) + 0.1 U(1,2) + 0.1 U(1,1) } (R)

Using the values from the previous picture, this becomes:

U(1,1) = -0.04 +max{ 0.6096 + 0.0655 + 0.0705 = 0.7456, (U)

0.6345 + 0.0762 = 0.7107, (L)

0.6345 + 0.0655 = 0.7000, (D)

0.5240 + 0.0762 + 0.0705 = 0.6707 } (R)

• Simultaneously solving the Bellman equations using does not work using the efficient techniques for systems of linear equations, because max is a nonlinear operation

• In the iterative approach we start with arbitrary initial values for the utilities, calculate the right-hand side of the equation and plug it into the left-hand side

• If we apply the Bellman update infinitely often, we are guaranteed
to reach an equilibrium, in which case the final utility values must
be solutions to the Bellman equations

• They are also the unique solutions, and the corresponding policy is optimal

Policy Iteration

• Beginning from some initial policy 0 alternate

• Policy evaluation: given a policy i , calculate Ui = Ui , the utility of each state if i were to be executed

• Policy improvement: Calculate the new MEU policy i+1, using one-step look-ahead based on Ui (Equation (*))

• The algorithm terminates when the policy improvement step yields no change in utilities

• At this point, we know that the utility function Ui is a fixed point of the Bellman update and a solution to the Bellman equations, so i must be an optimal policy

• Because there are only finitely many policies for a finite state space, and each iteration can be shown to yield a better policy, policy iteration must terminate

• Because at the ith iteration the policy i specifies the action i (s) in state s, there is no need to maximize over actions in policy iteration

• We have a simplified version of the Bellman equation: Ui (s) = R(s) + s‘ P(s’ | s, i (s)) Ui (s')

• For example: Ui (1,1) = –0.04 + 0.8 Ui (1,2) + 0.1 Ui (1,1) + 0.1 Ui (2,1) Ui (1,2) = –0.04 + 0.8 Ui (1,3) + 0.2 Ui (1,2) etc.

• Now the nonlinear max has been removed, and we have linear equations

• A system of linear equations with n equations with n unknowns can be solved exactly in time O(n3) by standard linear algebra methods

Instead of using a cubic amount of time to reach the exact solution (for large state spaces), we can instead perform some number simplified value iteration steps to give a reasonably good approximation of the utilities Ui+1(s) = R(s) + ∑s‘ P(s’ | s, πi (s)) Ui (s')

• This algorithm is called modified policy iteration

• In asynchronous policy iteration we pick any subset of the states on each iteration and apply either policy improvement or simplified value iteration to that subset

• Given certain conditions on the initial policy and initial utility function, asynchronous policy iteration is guaranteed to converge to an optimal policy

• We can design, e.g., algorithms that concentrate on updating the values of states that are likely to be reached by a good policy

Partially Observable Decision Processes

A **partially observable Markov decision process** (**POMDP**) is a combination of an MDP and a hidden Markov model. Instead of assuming that the state is observable, we assume that there are some partial and/or noisy observations of the state that the agent gets to observe before it has to act.

A POMDP consists of the following:

- *S*, a set of states of the world;
- *A*, a set of actions;
- *O*, a set of possible observations;
- $P(S_0)$, which gives the probability distribution of the starting state;
- *P(S'|S,A)*, which specifies the dynamics - the probability of getting to state *S'* by doing action *A* from state *S*;
- *R(S,A,S')*, which gives the expected reward of starting in state *S*, doing action *A*, and transitioning to state *S'*; and
- *P(O|S)*, which gives the probability of observing *O* given the state is *S*.
    - Make the policy a function of the belief state - a probability distribution over the states. Maintaining the belief state is the problem of filtering. The problem with this approach is that, with *n* states, the set of belief states is an *(n-1)*-dimensional real

space. However, because the value of a sequence of actions only depends on the states, the expected value is a linear function of the values of the states. Because plans can be conditional on observations, and we only consider optimal actions for any belief state, the optimal policy for any finite look-ahead, is piecewise linear and convex.

- Search over the space of controllers for the best controller. Thus, the agent searches over what to remember and what to do based on its belief state and observations. Note that the first two proposals are instances of this approach: the agent remembers all of its history or the agent has a belief state that is a probability distribution over possible states. In general, the agent may want to remember some parts of its history but have probabilities over some other features. Because it is unconstrained over what to remember, the search space is enormous.