CSE 3402: Intro to Artificial Intelligence Decision Making Under Uncertainty

- Readings: Russell & Norvig Chapter 16 and Chapter 17 Sec. 1 to 3 (in both 3rd and 2nd editions)

Preferences

- I give robot a planning problem: I want coffee
  - but coffee maker is broken: robot reports "No plan!"

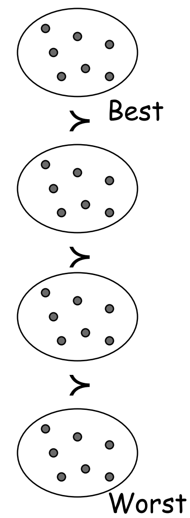
Preferences

- We really want more robust behavior.
  - Robot to know what to do if my primary goal can't be satisfied – I should provide it with some indication of my *preferences over alternatives*
  - e.g., coffee better than tea, tea better than water, water better than nothing, etc.
- But it's more complex:
  - it could wait 45 minutes for coffee maker to be fixed
  - what's better: tea now? coffee in 45 minutes?
  - could express preferences for <beverage,time> pairs

Preference Orderings

- A *preference ordering* $\succcurlyeq$ is a ranking of all possible states of affairs (worlds) S
  - these could be outcomes of actions, truth assts, states in a search problem, etc.
  - s $\succcurlyeq$ t: means that state s is *at least as good as* t
  - s $\succ$ t: means that state s is *strictly preferred to* t
- We insist that $\succcurlyeq$ is
  - reflexive: i.e., s $\succcurlyeq$ s for all states s
  - transitive: i.e., if s $\succcurlyeq$ t and t $\succcurlyeq$ w, then s $\succcurlyeq$ w
  - connected: for all states s,t, either s $\succcurlyeq$ t or t $\succcurlyeq$ s

Why Impose These Conditions?



- Structure of preference ordering imposes certain "rationality requirements" (it is a weak ordering)
- E.g., why transitivity?
  - Suppose you (strictly) prefer coffee to tea, tea to OJ, OJ to coffee
  - If you prefer X to Y, you'll trade me Y plus $1 for X
  - I can construct a "money pump" and extract arbitrary amounts of money from you

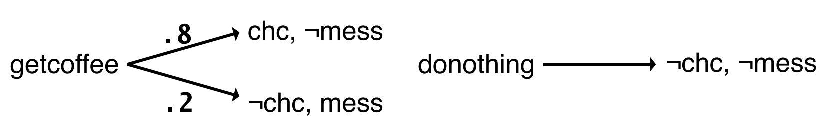
Decision Problems: Certainty

- A *decision problem under certainty* is:
  - a set of *decisions* D
    - e.g., paths in search graph, plans, actions…
  - a set of *outcomes* or states S
    - e.g., states you could reach by executing a plan
  - an *outcome function* f : D →S
    - the outcome of any decision
  - a preference ordering ≽ over S
- A *solution* to a decision problem is any d*∈ D such that f(d*) ≽ f(d) for all d∈D

Decision Problems: Certainty

- A *decision problem under certainty* is:
  - a set of *decisions* D
  - a set of *outcomes* or states S
  - an *outcome function* f : D →S
  - a preference ordering ≽ over S
- A *solution* to a decision problem is any d*∈ D such that f(d*) ≽ f(d) for all d∈D
  - e.g., in classical planning we that any goal state s is preferred/equal to every other state. So d* is a solution iff f(d*) is a solution state. I.e., d* is a solution iff it is a plan that achieves the goal.
  - More generally, in classical planning we might consider different goals with different values, and we want d* to be a plan that optimizes our value.

Decision Making under Uncertainty



- Suppose actions don' t have deterministic outcomes
  - e.g., when robot pours coffee, it spills 20% of time, making a mess
  - preferences: chc, ¬mess ≻ ¬chc,¬mess ≻ ¬chc, mess
- What should robot do?
  - decision *getcoffee* leads to a good outcome and a bad outcome with some probability
  - decision *donothing* leads to a medium outcome for sure
- Should robot be optimistic? pessimistic?
- Really odds of success should influence decision
  - but how?

Utilities

- Rather than just ranking outcomes, we must quantify our degree of preference
  - e.g., how much more important is having coffee than having tea?
- A *utility function* **U**: S →ℝ associates a real-valued *utility* with each outcome (state).
  - U(s) quantifies our degree of preference for s
- Note: U induces a preference ordering $\succcurlyeq_U$ over the states S defined as: s $\succcurlyeq_U$ t iff U(s) ≥ U(t)
  - $\succcurlyeq_U$ is reflexive, transitive, connected

Expected Utility

- With utilities we can compute **expected utilities!**
- In decision making under uncertainty, each decision d induces a distribution Pr_d over possible outcomes
  - $Pr_d(s)$ is probability of outcome s under decision d
- The *expected utility* of decision d is defined

$$EU(d) = \sum_{s \in S} \Pr_d(s)U(s)$$

Expected Utility

- Say U(chc,¬ms) = 10, U(¬chc,¬ms) = 5, U (¬chc,ms) = 0,
- Then
  - EU(getcoffee) = 8
  - EU(donothing) = 5
- If U(chc,¬ms) = 10, U(¬chc,¬ms) = 9, U (¬chc,ms) = 0,
  - EU(getcoffee) = 8
  - EU(donothing) = 9

The MEU Principle

- The *principle of maximum expected utility (MEU)* states that the optimal decision under conditions of uncertainty is the decision that has greatest expected utility.
- In our example
  - if my utility function is the first one, my robot should get coffee
  - if your utility function is the second one, your robot should do nothing

Computational Issues

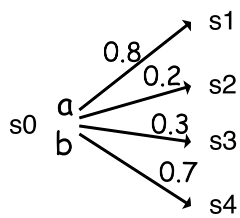
- At some level, solution to a dec. prob. is trivial
  - complexity lies in the fact that the decisions and outcome function are rarely specified explicitly
  - e.g., in planning or search problem, you *construct* the set of decisions by constructing paths or exploring search paths. Then we have to evaluate the expected utility of each. Computationally hard!
  - e.g., we find a plan achieving some expected utility e
    - Can we stop searching?
    - Must convince ourselves no better plan exists
    - Generally requires searching entire plan space, unless we have some clever tricks

Decision Problems: Uncertainty

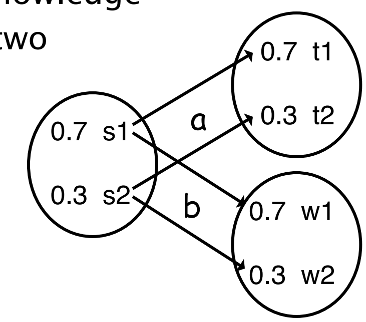
- A *decision problem under uncertainty* is:
  - a set of *decisions* D
  - a set of *outcomes* or states S
  - an *outcome function* Pr : D →Δ(S)
    - Δ(S) is the set of distributions over S (e.g., Pr_d)
  - a utility function U over S
- A *solution* to a decision problem under uncertainty is any d*∈ D such that EU(d*) ≽ EU(d) for all d∈D

Expected Utility: Notes

- Note that this viewpoint accounts for both:
  - uncertainty in action outcomes
  - uncertainty in state of knowledge
  - any combination of the two



Stochastic actions



Uncertain knowledge

Expected Utility: Notes

- Why MEU? Where do utilities come from?
  - underlying foundations of utility theory tightly couple utility with action/choice
  - a utility function can be determined by asking someone about their preferences for actions in specific scenarios (or “lotteries” over outcomes)
- Utility functions needn’t be unique
  - if I multiply U by a positive constant, all decisions have same relative utility
  - if I add a constant to U, same thing
  - *U is unique up to positive affine transformation*

So What are the Complications?

- Outcome space is large
  - like all of our problems, states spaces can be huge
  - don't want to spell out distributions like Pr_d explicitly
  - Soln: Bayes nets (or related: *influence diagrams*)

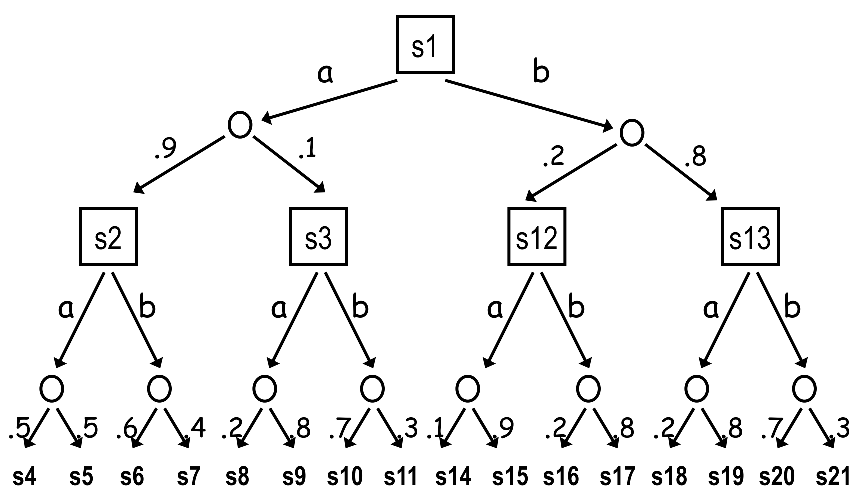
So What are the Complications?

- Decision space is large
  - usually our decisions are not one-shot actions
  - rather they involve sequential choices (like plans)
  - if we treat each plan as a distinct decision, decision space is too large to handle directly
  - Soln: use dynamic programming methods to *construct* optimal plans (actually generalizations of plans, called policies… like in game trees)

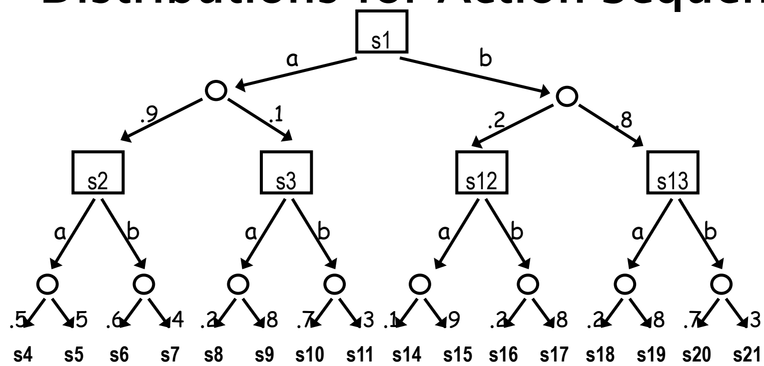
An Simple Example

- Suppose we have two actions: a, b
- We have time to execute *two* actions in sequence
- This means we can do either:
  - [a,a], [a,b], [b,a], [b,b]
- Actions are stochastic: action a induces distribution $Pr_a(s_i \mid s_j)$ over states
  - e.g., $Pr_a(s_2 \mid s_1) = .9$ means prob. of moving to state s_2 when a is performed at s_1 is .9
  - similar distribution for action b
- How good is a particular sequence of actions?

Distributions for Action Sequences



Distributions for Action Sequences

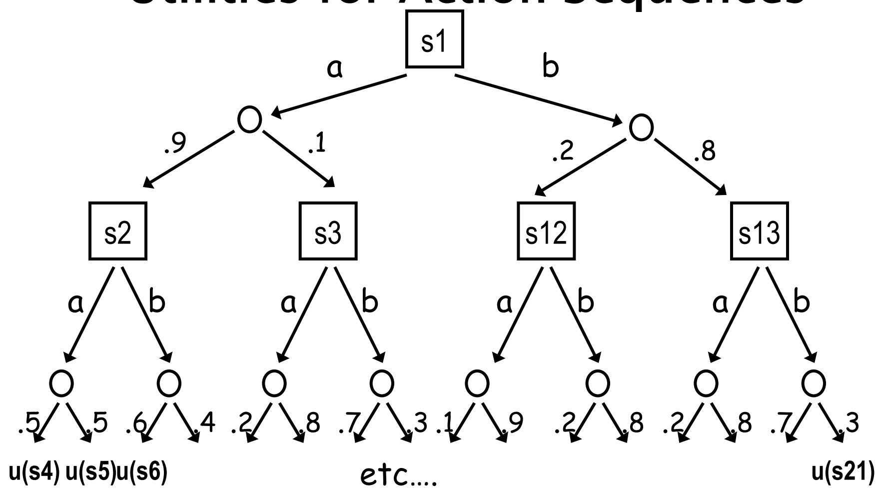


- Sequence [a,a] gives distribution over "final states"
  - Pr(s4) = .45, Pr(s5) = .45, Pr(s8) = .02, Pr(s9) = .08
- Similarly:
  - [a,b]: Pr(s6) = .54, Pr(s7) = .36, Pr(s10) = .07, Pr(s11) = .03
  - and similar distributions for sequences [b,a] and [b,b]

How Good is a Sequence?

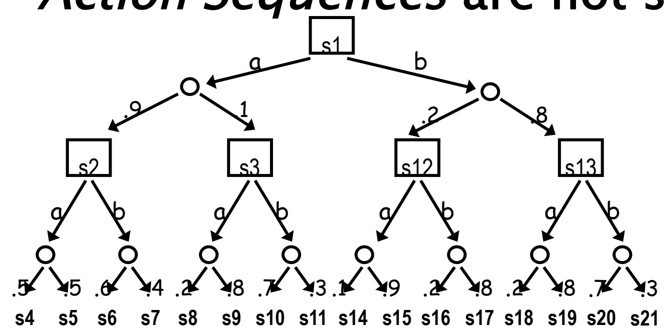
- We associate *utilities with the "final" outcomes*
  - how good is it to end up at s4, s5, s6, …
- Now we have:
  - EU(aa) = .45u(s4) + .45u(s5) + .02u(s8) + .08u(s9)
  - EU(ab) = .54u(s6) + .36u(s7) + .07u(s10) + .03u(s11)
  - etc…

Utilities for Action Sequences



Looks a lot like a game tree, but with chance nodes instead of min nodes. (We average instead of minimizing)

Action Sequences are not sufficient



- Suppose we do *a* first; we could reach s2 or s3:
  - At s2, assume: EU(a) = .5u(s4) + .5u(s5) > EU(b) = .6u(s6) + .4u(s7)
  - At s3: EU(a) = .2u(s8) + .8u(s9) < EU(b) = .7u(s10) + .3u(s11)
- After doing *a* first, we want to do *a* next *if we reach s2*, but we want to do *b* second *if we reach s3*

Policies

- This suggests that when dealing with uncertainty we want to consider *policies*, **not** just sequences of actions (plans)
- We have eight policies for this decision tree:

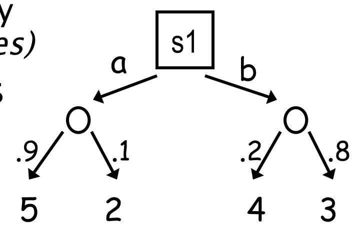
  [a; if s2 a, if s3 a]   [b; if s12 a, if s13 a]
  [a; if s2 a, if s3 b]   [b; if s12 a, if s13 b]
  [a; if s2 b, if s3 a]   [b; if s12 b, if s13 a]
  [a; if s2 b, if s3 b]   [b; if s12 b, if s13 b]
- Contrast this with four "plans"
  - [a; a], [a; b], [b; a], [b; b]
  - note: each plan corresponds to a policy, so we can only *gain* by allowing decision maker to use policies

Evaluating Policies

- Number of plans (sequences) of length *k*
  - exponential in *k*: $|A|^k$ if *A* is our action set
- Number of policies is even larger
  - if we have $n=|A|$ actions and $m=|O|$ outcomes per action, then we have $(nm)^k$ policies
- Fortunately, *dynamic programming* can be used
  - e.g., suppose EU(a) > EU(b) at s2
  - never consider a policy that does anything else at s2
- How to do this?
  - back values up the tree much like minimax search

Decision Trees

- Squares denote *choice* nodes
  - these denote action choices by decision maker *(decision nodes)*
- Circles denote *chance* nodes
  - these denote uncertainty regarding action effects
  - "nature" will choose the child with specified probability
- Terminal nodes labeled with *utilities*
  - denote utility of final state (or it could denote the utility of "trajectory" (branch) to decision maker

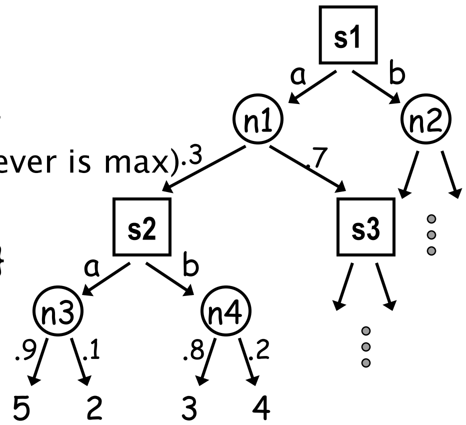


Evaluating Decision Trees

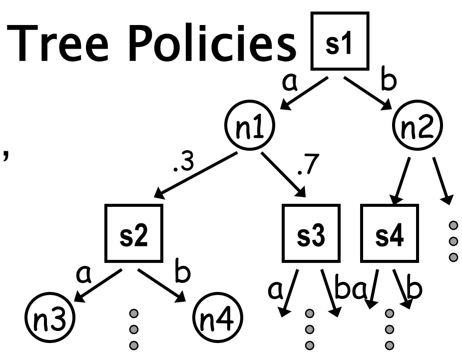
- Procedure is exactly like game trees, except…
  - key difference: the "opponent" is "nature" who simply chooses outcomes at chance nodes with specified probability: so we take expectations instead of minimizing
- Back values *up* the tree
  - *U(t)* is defined for all terminals (part of input)
  - *U(n)* = exp {*U(c) : c* a child of *n*} if *n* is a chance node
  - *U(n)* = max {*U(c) : c* a child of *n*} if *n* is a choice node
- At any choice node (state), the decision maker chooses action that leads to *highest utility child*

Evaluating a Decision Tree

- U(n3) = .9*5 + .1*2
- U(n4) = .8*3 + .2*4
- U(s2) = max{U(n3), U(n4)}
  - decision a or b (whichever is max)
- U(n1) = .3U(s2) + .7U(s3)
- U(s1) = max{U(n1), U(n2)}
  - decision: max of a, b



Decision Tree Policies



- Note that we don't just compute values, but policies for the tree
- A *policy* assigns a decision to each choice node in tree
- Some policies can't be distinguished in terms of their expected values
  - e.g., if policy chooses a at node s1, choice at s4 doesn't matter because it won't be reached
  - Two policies are *implementationally indistinguishable* if they disagree only at unreachable decision nodes
    - reachability is determined by policy themselves

Key Assumption: Observability

- **Full observability:** we must know the initial state and outcome of each action
  - specifically, to implement the policy, *we must be able to resolve the uncertainty of <u>any chance node that is followed by a decision node</u>*
  - e.g., after doing a at s1, we must know which of the outcomes (s2 or s3) was realized so we know what action to do next (note: s2 and s3 may prescribe different ations)

Computational Issues

- Savings compared to explicit policy evaluation is substantial
- Evaluate only $O((nm)^d)$ nodes in tree of depth d
  - total computational cost is thus $O((nm)^d)$
- Note that this is how many ***policies*** there are
  - but evaluating a single policy explicitly requires substantial computation: $O(nm^d)$
  - total computation for explicity evaluating each policy would be $O(n^d m^{2d})$!!!
- Tremendous value to dynamic programming solution

Computational Issues

- **Tree size**: grows exponentially with depth
- Possible solutions:
  - bounded lookahead with heuristics (like game trees)
  - heuristic search procedures (like A*)

Other Issues

- **Specification**: suppose each state is an assignment to variables; then representing action probability distributions is complex (and branching factor could be immense)
- Possible solutions:
  - represent distribution using Bayes nets
  - solve problems using *decision networks* (or influence diagrams)

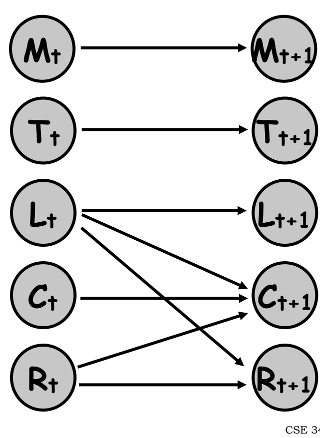
Large State Spaces (Variables)

- To represent outcomes of actions or decisions, we need to specify distributions
  - Pr(s|d) : probability of outcome s given decision d
  - Pr(s|a,s’): prob. of state s given that action a performed in state s’
- But state space exponential in # of variables
  - spelling out distributions explicitly is intractable
- Bayes nets can be used to represent actions
  - this is just a joint distribution over variables, conditioned on action/decision and previous state

Example Action using Dynamic BN

M – mail waiting C – Craig has coffee
T – lab tidy R – robot has coffee
L – robot located in Craig’s office

Deliver Coffee action



T	T(t+1)	$\overline{T(t+1)}$
T	1.0	0.0
F	0.0	1.0

$f_J(T_t, T_{t+1})$

L	R	C	C(t+1)	$\overline{C(t+1)}$
T	T	T	1.0	0.0
F	T	T	1.0	0.0
T	F	T	1.0	0.0
F	F	T	1.0	0.0
T	T	F	0.8	0.2
F	T	F	0.0	1.0
T	F	F	0.0	1.0
F	F	F	0.0	1.0

$f_R(L_t, R_t, C_t, C_{t+1})$

Dynamic BN Action Representation

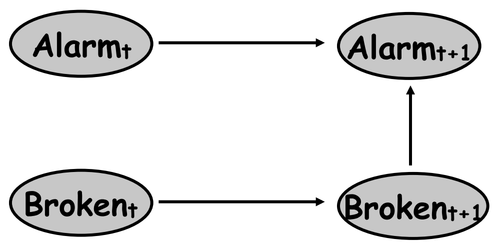
- Dynamic Bayesian networks (DBNs):
  - a way to use BNs to represent *specific* actions
  - list all state variables for time t (pre-action)
  - list all state variables for time t+1 (post-action)
  - indicate parents of all t+1 variables
    - these can include time t and time t+1 variables
    - network must be acyclic
  - specify CPT for each time t+1 variable

Dynamic BN Action Representation

- Note: generally *no prior given* for time t variables
  - we' re (generally) interested in *conditional* distribution over post-action states given pre-action state
  - so time t vars are instantiated as "evidence" when using a DBN (generally)

Example of Dependence within Slice

Throw rock at window action



$P(al_{t+1} \mid al_t, Br_t) = 1$
$P(al_{t+1} \mid \neg al_t, \neg br_{t+1}) = 0$
$P(al_{t+1} \mid \neg al_t, br_{t+1}) = .95$

$P(broken_{t+1} \mid broken_t) = 1$
$P(broken_{t+1} \mid \neg broken_t) = .6$

Throwing rock has certain probability of breaking window and setting off alarm; but whether alarm is triggered depends on whether rock *actually* broke the window.

Use of BN Action Reprsnt'n

- DBNs: actions concisely,naturally specified
  - These look a bit like STRIPS and the situtation calculus, but allow for probabilistic effects

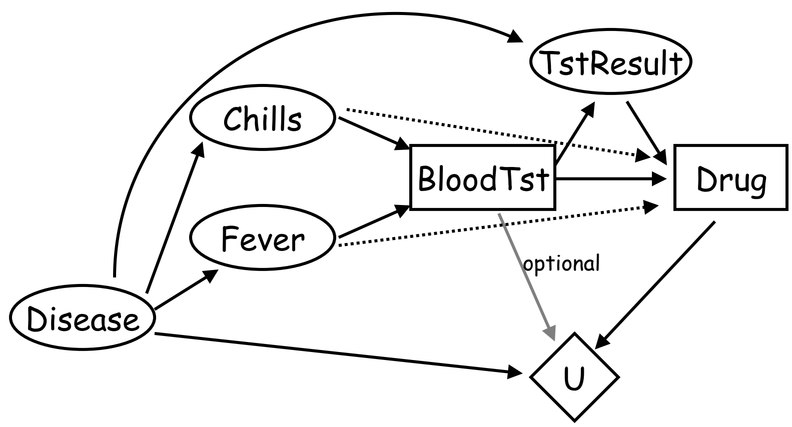
Use of BN Action Reprsnt' n

- How to use:
  - use to generate "expectimax" search tree to solve decision problems
  - use directly in stochastic decision making algorithms
- First use doesn' t buy us much computationally when solving decision problems. But second use allows us to compute expected utilities without enumerating the outcome space (tree)
  - well see something like this with *decision networks*

Decision Networks

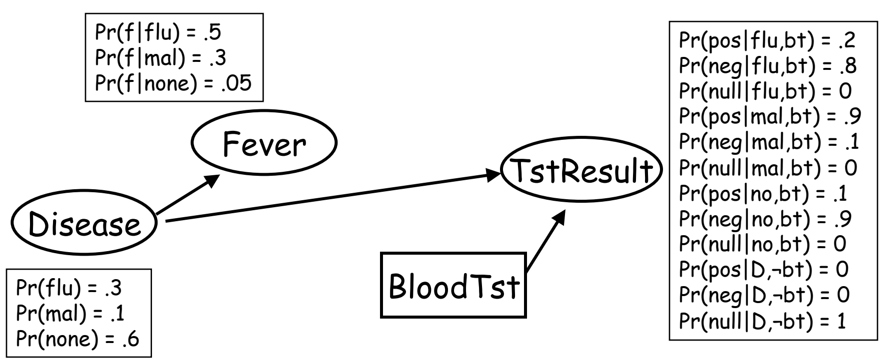
- *Decision networks* (more commonly known as *influence diagrams*) provide a way of representing sequential decision problems
  - basic idea: represent the variables in the problem as you would in a BN
  - add decision variables – variables that you "control"
  - add utility variables – how good different states are

Sample Decision Network



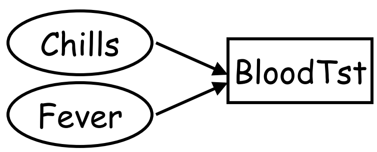
Decision Networks: Chance Nodes

- Chance nodes
  - random variables, denoted by circles
  - as in a BN, probabilistic dependence on parents



Decision Networks: Decision Nodes

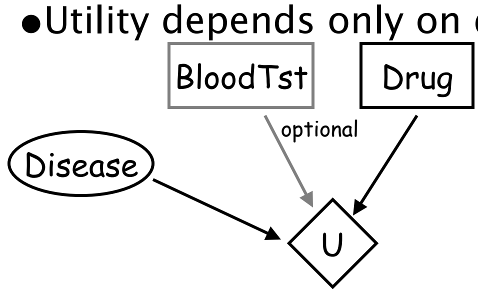
- **Decision nodes**
  - variables decision maker sets, denoted by squares
  - parents reflect *information available* at time decision is to be made
- In example decision node: the actual values of Ch and Fev will be observed before the decision to take test must be made
  - agent can make *different decisions* for each instantiation of parents



BT ∈ {bt, ¬bt}

Decision Networks: Value Node

- **Value node**
  - specifies utility of a state, denoted by a diamond
  - utility depends *only on state of parents* of value node
  - generally: only one value node in a decision network
- Utility depends only on disease and drug



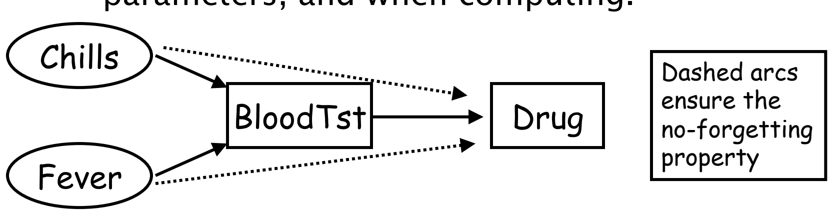
U(fludrug, flu) = 20
U(fludrug, mal) = -300
U(fludrug, none) = -5
U(maldrug, flu) = -30
U(maldrug, mal) = 10
U(maldrug, none) = -20
U(no drug, flu) = -10
U(no drug, mal) = -285
U(no drug, none) = 30

Decision Networks: Assumptions

- Decision nodes are totally ordered
  - decision variables D_1, D_2, …, D_n
  - decisions are made in sequence
  - e.g., BloodTst (yes,no) decided before Drug (fd,md,no)

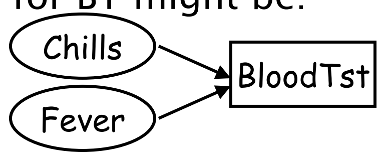
Decision Networks: Assumptions

- *No-forgetting property*
  - any information available when decision D_i is made is available when decision D_j is made (for $i < j$)
  - thus all parents of D_i are parents of D_j
    - Network does not show these "implicit parents", but the links are present, and must be considered when specifying the network parameters, and when computing.



Policies

- Let $Par(D_i)$ be the parents of decision node D_i
  - $Dom(Par(D_i))$ is the set of assignments to parents
- A policy δ is a set of mappings δ_i, one for each decision node D_i
  - $\delta_i : Dom(Par(D_i)) \rightarrow Dom(D_i)$
  - δ_i associates a decision with each parent asst for D_i
- For example, a policy for BT might be:
  - $\delta_{BT}(c,f) = bt$
  - $\delta_{BT}(c,\neg f) = \neg bt$
  - $\delta_{BT}(\neg c,f) = bt$
  - $\delta_{BT}(\neg c,\neg f) = \neg bt$



Value of a Policy

- *Value of a policy δ* is the expected utility given that decision nodes are executed according to δ
- Given asst **x** to the set **X** of all chance variables, let δ(**x**) denote the asst to decision variables dictated by δ
  - e.g., asst to D_1 determined by it's parents' asst in **x**
  - e.g., asst to D_2 determined by it's parents' asst in **x** along with whatever was assigned to D_1
  - etc.
- Value of δ :

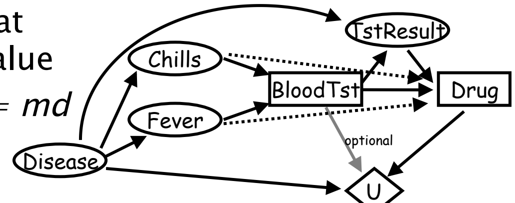
$$EU(\delta) = \sum_{\mathbf{X}} P(\mathbf{X}, \delta(\mathbf{X}))\, U(\mathbf{X}, \delta(\mathbf{X}))$$

Optimal Policies

- An *optimal policy* is a policy δ^* such that
  EU(δ^*) $\geq$ EU(δ) for all policies δ
- We can use the dynamic programming principle to avoid enumerating all policies
- We can also use the structure of the decision network to use variable elimination to aid in the computation

Computing the Best Policy

- We can work backwards as follows
- First compute optimal policy for Drug (last dec'n)
  - for each asst to parents (C,F,BT,TR) and for each decision value (D = md,fd,none), *compute the expected value* of choosing that value of D
  - set policy choice for each value of parents to be the value of D that has max value
  - eg: $\delta_D(c,f,bt,pos) = md$

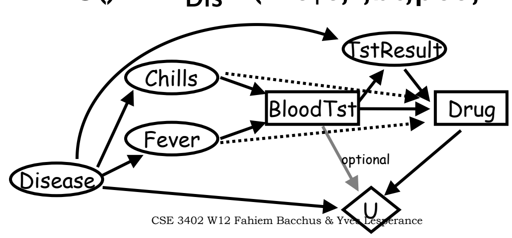


Computing the Best Policy

- Next compute policy for BT given policy δ_D *(C,F,BT,TR)* just determined for Drug
  - since $\delta_D(C,F,BT,TR)$ is fixed, we can treat Drug as a normal random variable with deterministic probabilities
  - i.e., for any instantiation of parents, value of Drug is fixed by policy δ_D
  - this means we can solve for optimal policy for BT just as before
  - only uninstantiated vars are random vars (once we fix *its* parents)

Computing the Best Policy

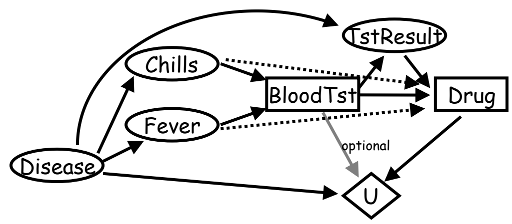
- How do we compute these expected values?
  - suppose we have asst *<c,f,bt,pos>* to parents of *Drug*
  - we want to compute EU of deciding to set *Drug = md*
  - we can run variable elimination!
- Treat *C,F,BT,TR,Dr* as evidence
  - this reduces factors (e.g., *U* restricted to *bt,md*: depends on *Dis*)
  - eliminate remaining variables (e.g., only *Disease* left)
  - left with factor: $U() = \Sigma_{Dis}\ P(Dis|c,f,bt,pos,md)U(Dis)$



Computing the Best Policy

We now know EU of doing *Dr=md* when *c,f,bt,pos* true

- Can do same for *fd,no* to decide which is best

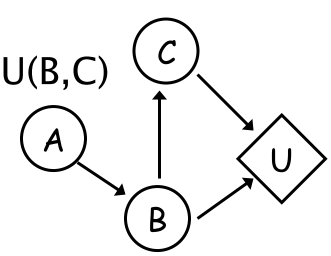


Computing Expected Utilities

- The preceding illustrates a general phenomenon
  - computing expected utilities with BNs is quite easy
  - utility nodes are just factors that can be dealt with using variable elimination

$$EU = \Sigma_{A,B,C}\ P(A,B,C)\ U(B,C)$$
$$= \Sigma_{A,B,C}\ P(C|B)\ P(B|A)\ P(A)\ U(B,C)$$

- Just eliminate variables in the usual way



Optimizing Policies: Key Points

- If a decision node D has no decisions that follow it, we can find its policy by instantiating each of its parents and computing the expected utility of each decision for each parent instantiation

Optimizing Policies: Key Points

- no-forgetting means that all other decisions are instantiated (they must be parents)
- its easy to compute the expected utility using VE
- the number of computations is quite large: we run expected utility calculations (VE) for each parent instantiation together with each possible decision D might allow
- policy: choose max decision for each parent instant’n

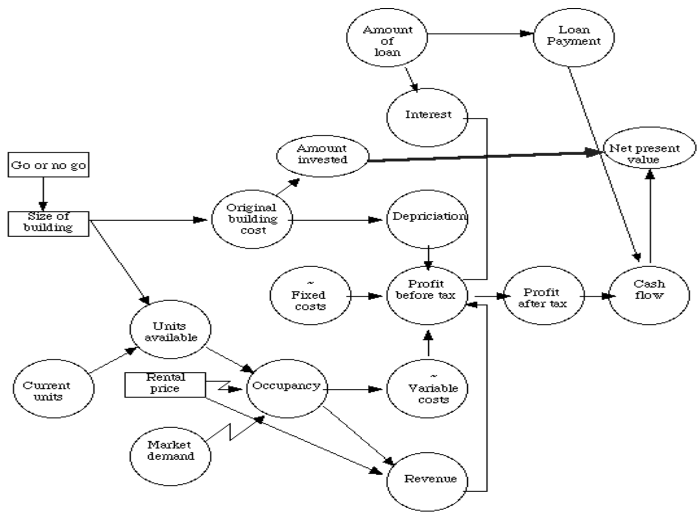
Optimizing Policies: Key Points

- When a decision D node is optimized, it can be treated as a random variable
  - for each instantiation of its parents we now know what value the decision should take
  - just treat policy as a new CPT: for a given parent instantiation **x**, D gets $\delta(\mathbf{x})$ with probability 1(all other decisions get probability zero)
- If we optimize from last decision to first, at each point we can optimize a specific decision by (a bunch of) simple VE calculations
  - it's successor decisions (optimized) are just normal nodes in the BNs (with CPTs)

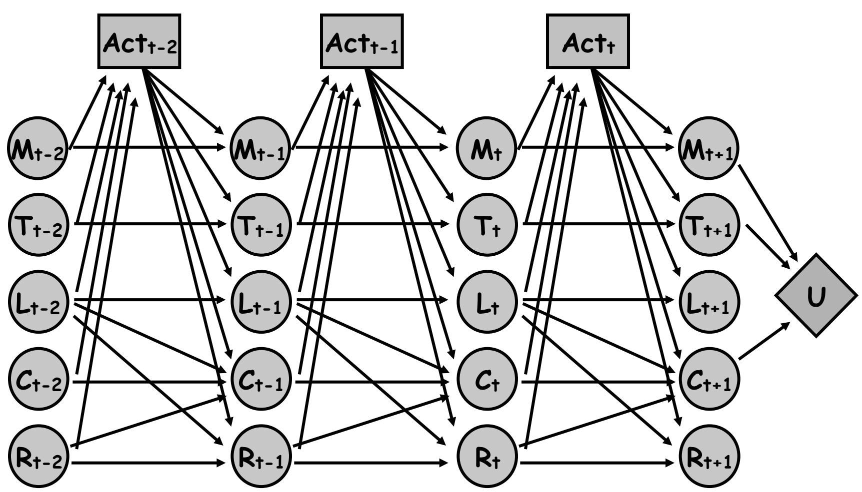
Decision Network Notes

- Decision networks commonly used by decision analysts to help structure decision problems
- Much work put into computationally effective techniques to solve these
- Complexity much greater than BN inference
  - we need to solve a number of BN inference problems
  - one BN problem for each setting of decision node parents and decision node value

Real Estate Investment



DBN-Decision Nets for Planning



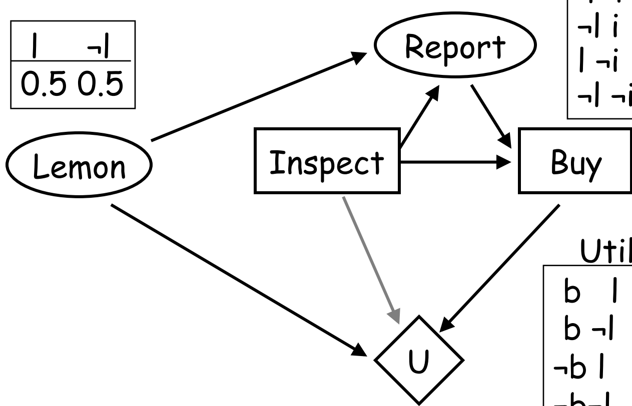
A Detailed Decision Net Example

- Setting: you want to buy a used car, but there's a good chance it is a "lemon" (i.e., prone to breakdown). Before deciding to buy it, you can take it to a mechanic for inspection. They will give you a report on the car, labeling it either "good" or "bad". A good report is positively correlated with the car being sound, while a bad report is positively correlated with the car being a lemon.

A Detailed Decision Net Example

- However the report costs $50. So you could risk it, and buy the car without the report.
- Owning a sound car is better than having no car, which is better than owning a lemon.

Car Buyer's Network



l	¬l
0.5	0.5

Rep: good,bad,none

	g	b	n
l i	0.2	0.8	0
¬l i	0.9	0.1	0
l ¬i	0	0	1
¬l ¬i	0	0	1

Utility

b l	-600
b ¬l	1000
¬b l	-300
¬b¬l	-300

-50 if inspect

Evaluate Last Decision: Buy (1)

- EU(B|I,R) = Σ_L P(L|I,R,B)U(L,B)
  - The probability of the remaining variables in the Utility function, times the utility function. Note P(L|I,R,B) = P(L|I,R), as B is a decision variable that does not influence L.
- I = i, R = g:
  - P(L|I,g): use variable elimination. Query variable L is only remaining variable, so we only need to normalize (no summations).
  - P(L,i,g) = P(L)P(g|L,i)
    HENCE: P(L|i,g) = normalized [P(l)P(g|l,i),P(¬l)P(g|¬l,i)
    = [0.5*.2, 0.5*0.9] = [.18, .82]
  - EU(buy) = P(l|i,g)U(buy,l) + P(¬l)P(¬l|i,g) U(buy,¬l)–50
    = .18*–600 + .82*1000 – 50 = 662
  - EU(¬buy) = P(l|i, g) U(¬buy,l) + P(¬l|i, g) U(¬buy,¬l) – 50
    = .18*–300 + .82*–300 –50 = –350
- So optimal *δ_{Buy} (i,g) = buy*

Evaluate Last Decision: Buy (2)

- I = i, R = b:
  - P(L,i,b) = P(L)P(b|L,i)
    P(L|i,g) = normalized [P(l)P(b|l,i),P(¬l)P(b|¬l,i)
    = [0.5*.8, 0.5*0.1] = [.89, .11]
  - EU(buy) = P(l|i, b) U(l,buy) + P(¬l|i, b) U(¬l,buy) - 50
    = .89*-600 + .11*1000 - 50 = -474
  - EU(¬buy) = P(l|i, b) U(l,¬buy) + P(¬l|i, b) U(¬l,¬buy) - 50
    = .89*-300 + .11*-300 -50 = -350
- So optimal δ_{Buy} *(i,b) = ¬buy*

Evaluate Last Decision: Buy (3)

- I = ¬i, R = n
  - P(L,¬i,n) = P(L)P(n|L,¬i)
    P(L|¬i,n) = normalized [P(l)P(n|l,¬i),P(¬l)P(n|¬l,¬i)
    = [0.5*1, 0.5*1] = [.5,.5]
  - EU(buy) = P(l|¬i,n) U(l,buy) + P(¬l|¬i,n) U(¬l,buy)
    = .5*-600 + .5*1000 = 200 (no inspection cost)
  - EU(¬buy) = P(l|¬i, n) U(l,¬buy) + P(¬l|¬i, n) U(¬l,¬buy)
    = .5*-300 +.5*-300 = -300
- So optimal δ_{Buy} *(¬i,n) = buy*
- Overall optimal policy for Buy is:
  - δ_{Buy} *(i,g) = buy ;* δ_{Buy} *(i,b) = ¬buy ;* δ_{Buy} *(¬i,n) = buy*
- Note: we don't bother computing policy for (i,¬n), (¬i, g), or (¬i, b), since these occur with probability 0

Evaluate First Decision: Inspect

- EU(I) = $\Sigma_{L,R}$ P(L,R|I) U(L, δ_{Buy} *(I,R)*)
  - where P(R,L|I) = P(R|L,I)P(L|I)

	P(R,L |i)	δ_{Buy}	U(L, δ_{Buy})
g,l	0.2*.5 = .1	buy	-600 - 50 = -650
b,l	0.8*.5 = .4	¬buy	-300 - 50 = -350
g,¬l	0.9*.5 = .45	buy	1000 - 50 = 950
b,¬l	0.1*.5 = .05	¬buy	-300 - 50 = -350

  - EU(i) = .1*-600 + .4*-300 + .45*1000 + .05*-300 - 50
    = 237.5 - 50 = 187.5
  - EU(¬i) = P(l|¬i, n) U(l,buy) + P(¬l|¬i, n) U(¬l,buy)
    = .5*-600 + .5*1000 = 200
  - So optimal $\delta_{Inspect}$ *(¬i) = buy*

Value of Information

- So optimal policy is: don't inspect, buy the car
  - EU = 200
  - Notice that the EU of inspecting the car, then buying it iff you get a good report, is 237.5 less the cost of the inspection (50). So inspection not worth the improvement in EU.
  - But suppose inspection cost $25: then it would be worth it (EU = 237.5 - 25 = 212.5 > EU(¬i))
  - The *expected value of information* associated with inspection is 37.5 (it improves expected utility by this amount ignoring cost of inspection). How? Gives opportunity to change decision (¬buy if bad).
  - You should be willing to pay up to $37.5 for the report