Making Decisions Under Uncertainty

What an agent should do depends on:

- The agent's ability — what options are available to it.
- The agent's beliefs — the ways the world could be, given the agent's knowledge.
  Sensing updates the agent's beliefs.
- The agent's preferences — what the agent wants and tradeoffs when there are risks.

Decision theory specifies how to trade off the desirability and probabilities of the possible outcomes for competing actions.

Decision Variables

- Decision variables are like random variables that an agent gets to choose a value for.

Decision Variables

- Decision variables are like random variables that an agent gets to choose a value for.
- A possible world specifies a value for each decision variable and each random variable.

Decision Variables

- Decision variables are like random variables that an agent gets to choose a value for.
- A possible world specifies a value for each decision variable and each random variable.
- For each assignment of values to *all* decision variables, there is a probability distribution over random variables.
- The probability of a proposition is undefined unless the agent conditions on the values of all decision variables.

Decision Tree for Delivery Robot

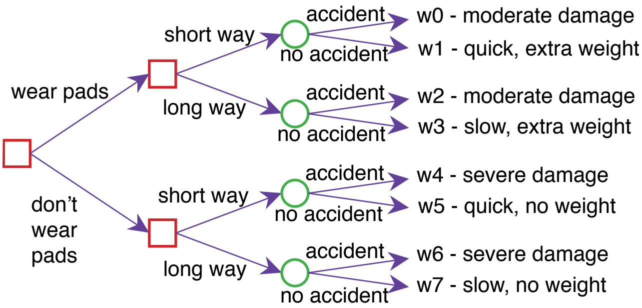
The robot can choose to wear pads to protect itself or not.

The robot can choose to go the short way past the stairs or a long way that reduces the chance of an accident.

There is one random variable of whether there is an accident.

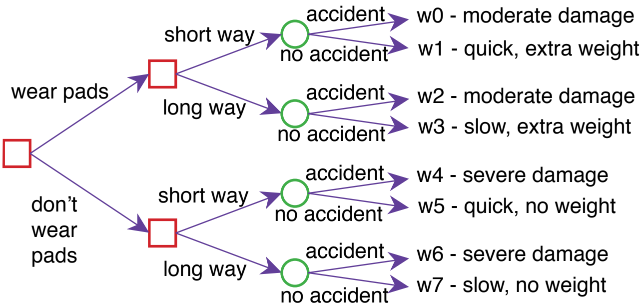
Decision Tree for Delivery Robot

The robot can choose to wear pads to protect itself or not.
The robot can choose to go the short way past the stairs or a long way that reduces the chance of an accident.
There is one random variable of whether there is an accident.



Decision Tree for Delivery Robot

The robot can choose to wear pads to protect itself or not.
The robot can choose to go the short way past the stairs or a long way that reduces the chance of an accident.
There is one random variable of whether there is an accident.



Square boxes represent decisions that the robot can make. Circles represent random variables that the robot can't observe before making its decision.

Single-stage decision networks

Extend belief networks with:

- Decision nodes that the agent chooses the value for.
  Domain is the set of possible actions. Drawn as rectangle.

Single-stage decision networks

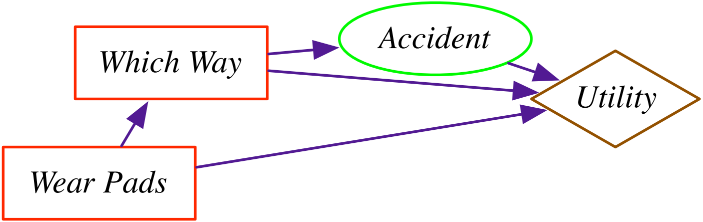
Extend belief networks with:

- Decision nodes that the agent chooses the value for. Domain is the set of possible actions. Drawn as rectangle.
- Utility node, whose parents are the variables on which the utility depends. Drawn as a diamond.

Single-stage decision networks

Extend belief networks with:

- Decision nodes that the agent chooses the value for. Domain is the set of possible actions. Drawn as rectangle.
- Utility node, whose parents are the variables on which the utility depends. Drawn as a diamond.



This shows explicitly which nodes affect whether there is an accident.

Single-stage decision networks

A single-stage decision network:

- DAG with three sorts of nodes: decision, random, utility.

Single-stage decision networks

A single-stage decision network:

- DAG with three sorts of nodes: decision, random, utility.
  Random nodes are the same as the nodes in a belief network.

Single-stage decision networks

A single-stage decision network:

- DAG with three sorts of nodes: decision, random, utility.
  Random nodes are the same as the nodes in a belief network.
- A domain for each decision variable and each random variable.

Single-stage decision networks

A single-stage decision network:

- DAG with three sorts of nodes: decision, random, utility.
  Random nodes are the same as the nodes in a belief network.
- A domain for each decision variable and each random variable.
- A unique utility node.
  The utility node has no children and no domain.

Single-stage decision networks

A single-stage decision network:

- DAG with three sorts of nodes: decision, random, utility.
  Random nodes are the same as the nodes in a belief network.
- A domain for each decision variable and each random variable.
- A unique utility node.
  The utility node has no children and no domain.

A single-stage decision network has the factors:

- A utility function is a factor on the parents of the utility node

Single-stage decision networks

A single-stage decision network:

- DAG with three sorts of nodes: decision, random, utility.
  Random nodes are the same as the nodes in a belief network.
- A domain for each decision variable and each random variable.
- A unique utility node.
  The utility node has no children and no domain.

A single-stage decision network has the factors:

- A utility function is a factor on the parents of the utility node
- A conditional probability for each random variable given its parents

Single-stage decision networks

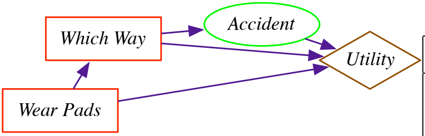
A single-stage decision network:

- DAG with three sorts of nodes: decision, random, utility.
  Random nodes are the same as the nodes in a belief network.
- A domain for each decision variable and each random variable.
- A unique utility node.
  The utility node has no children and no domain.

A single-stage decision network has the factors:

- A utility function is a factor on the parents of the utility node
- A conditional probability for each random variable given its parents
- (No tables associated with the decision nodes.)

Example Initial Factors



Which Way	Accident	Value
long	true	0.01
long	false	0.99
short	true	0.2
short	false	0.8

Which Way	Accident	Wear Pads	Value
long	true	true	30
long	true	false	0
long	false	true	75
long	false	false	80
short	true	true	35
short	true	false	3
short	false	true	95
short	false	false	100

Finding an optimal decision

- Suppose the random variables are $X_1, \ldots, X_n$, and utility depends on $V_{i_1}, \ldots, V_{i_k}$ (random and/or decision variables)

$$\mathbb{E}(u \mid D) \quad =$$

Finding an optimal decision

- Suppose the random variables are $X_1, \ldots, X_n$, and utility depends on $V_{i_1}, \ldots, V_{i_k}$ (random and/or decision variables)

$$\begin{aligned}
\mathbb{E}(u \mid D) &= \sum_{X_1,\ldots,X_n} P(X_1, \ldots, X_n \mid D) \times u(V_{i_1}, \ldots, V_{i_k}) \\
&= \sum_{X_1,\ldots,X_n}
\end{aligned}$$

Finding an optimal decision

- Suppose the random variables are $X_1, \ldots, X_n$, and utility depends on $V_{i_1}, \ldots, V_{i_k}$ (random and/or decision variables)

$$\begin{aligned}
\mathbb{E}(u \mid D) &= \sum_{X_1,\ldots,X_n} P(X_1, \ldots, X_n \mid D) \times u(V_{i_1}, \ldots, V_{i_k}) \\
&= \sum_{X_1,\ldots,X_n} \prod_{i=1}^{n} P(X_i \mid parents(X_i)_D) \times u(V_{i_1}, \ldots, V_{i_k})
\end{aligned}$$

Finding an optimal decision

- Suppose the random variables are $X_1, \ldots, X_n$, and utility depends on $V_{i_1}, \ldots, V_{i_k}$ (random and/or decision variables)

$$\begin{aligned}
\mathbb{E}(u \mid D) &= \sum_{X_1,\ldots,X_n} P(X_1,\ldots,X_n \mid D) \times u(V_{i_1},\ldots,V_{i_k}) \\
&= \sum_{X_1,\ldots,X_n} \prod_{i=1}^{n} P(X_i \mid parents(X_i)_D) \times u(V_{i_1},\ldots,V_{i_k})
\end{aligned}$$

To find an optimal decision:
  - Create a factor for each conditional probability and for the utility

Finding an optimal decision

- Suppose the random variables are $X_1, \ldots, X_n$, and utility depends on $V_{i_1}, \ldots, V_{i_k}$ (random and/or decision variables)

$$\begin{aligned}
\mathbb{E}(u \mid D) &= \sum_{X_1,\ldots,X_n} P(X_1, \ldots, X_n \mid D) \times u(V_{i_1}, \ldots, V_{i_k}) \\
&= \sum_{X_1,\ldots,X_n} \prod_{i=1}^{n} P(X_i \mid parents(X_i)_D) \times u(V_{i_1}, \ldots, V_{i_k})
\end{aligned}$$

  To find an optimal decision:
  - Create a factor for each conditional probability and for the utility
  - Sum out all of the random variables

Finding an optimal decision

- Suppose the random variables are $X_1, \ldots, X_n$, and utility depends on $V_{i_1}, \ldots, V_{i_k}$ (random and/or decision variables)

$$\begin{aligned}\mathbb{E}(u \mid D) &= \sum_{X_1,\ldots,X_n} P(X_1,\ldots,X_n \mid D) \times u(V_{i_1},\ldots,V_{i_k}) \\ &= \sum_{X_1,\ldots,X_n} \prod_{i=1}^{n} P(X_i \mid parents(X_i)_D) \times u(V_{i_1},\ldots,V_{i_k})\end{aligned}$$

  To find an optimal decision:
  - Create a factor for each conditional probability and for the utility
  - Sum out all of the random variables
  - This creates a factor on

Finding an optimal decision

- Suppose the random variables are $X_1, \ldots, X_n$, and utility depends on $V_{i_1}, \ldots, V_{i_k}$ (random and/or decision variables)

$$\begin{aligned}\mathbb{E}(u \mid D) &= \sum_{X_1,\ldots,X_n} P(X_1, \ldots, X_n \mid D) \times u(V_{i_1}, \ldots, V_{i_k}) \\ &= \sum_{X_1,\ldots,X_n} \prod_{i=1}^{n} P(X_i \mid parents(X_i)_D) \times u(V_{i_1}, \ldots, V_{i_k})\end{aligned}$$

  To find an optimal decision:
  - Create a factor for each conditional probability and for the utility
  - Sum out all of the random variables
  - This creates a factor on D that gives the expected utility for each value in the domain of D

Finding an optimal decision

- Suppose the random variables are $X_1, \ldots, X_n$, and utility depends on $V_{i_1}, \ldots, V_{i_k}$ (random and/or decision variables)

$$\begin{aligned}
\mathbb{E}(u \mid D) &= \sum_{X_1,\ldots,X_n} P(X_1,\ldots,X_n \mid D) \times u(V_{i_1},\ldots,V_{i_k}) \\
&= \sum_{X_1,\ldots,X_n} \prod_{i=1}^{n} P(X_i \mid parents(X_i)_D) \times u(V_{i_1},\ldots,V_{i_k})
\end{aligned}$$

  To find an optimal decision:
  - Create a factor for each conditional probability and for the utility
  - Sum out all of the random variables
  - This creates a factor on D that gives the expected utility for each value in the domain of D
  - Choose the D with the maximum value in the factor.

After summing out Accident

Which Way	Wear Pads	Value
long	true	74.55
long	false	79.2
short	true	83.0
short	false	80.6

(Sequential) Decision Networks

- flat or modular or hierarchical
- explicit states or features or individuals and relations
- static or finite stage or indefinite stage or infinite stage
- fully observable or partially observable
- deterministic or stochastic dynamics
- goals or complex preferences
- single agent or multiple agents
- knowledge is given or knowledge is learned
- perfect rationality or bounded rationality

Sequential Decisions

- An intelligent agent doesn't carry out a multi-step plan ignoring information it receives between actions.

Sequential Decisions

- An intelligent agent doesn't carry out a multi-step plan ignoring information it receives between actions.
- A more typical scenario is where the agent:
  observes, acts, observes, acts, ...

Sequential Decisions

- An intelligent agent doesn't carry out a multi-step plan ignoring information it receives between actions.
- A more typical scenario is where the agent:
  observes, acts, observes, acts, ...
- Subsequent actions can depend on what is observed.
  What is observed depends on previous actions.

Sequential Decisions

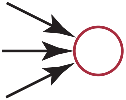
- An intelligent agent doesn't carry out a multi-step plan ignoring information it receives between actions.
- A more typical scenario is where the agent:
  observes, acts, observes, acts, ...
- Subsequent actions can depend on what is observed.
  What is observed depends on previous actions.
- Often the sole reason for carrying out an action is to provide information for future actions.
  For example: diagnostic tests, spying.

Sequential decision problems

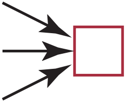
- A sequential decision problem consists of a sequence of decision variables $D_1, \ldots, D_n$.
- Each D_i has an information set of variables $parents(D_i)$, whose value will be known at the time decision D_i is made.

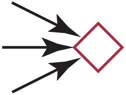
Decisions Networks

A decision network is a graphical representation of a finite sequential decision problem, with 3 types of nodes:



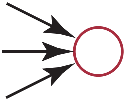
- A random variable is drawn as an ellipse. Arcs into the node represent probabilistic dependence. Each random variable has a domain and an associated factor.



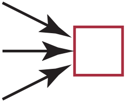


Decisions Networks

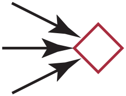
A decision network is a graphical representation of a finite sequential decision problem, with 3 types of nodes:



- A random variable is drawn as an ellipse. Arcs into the node represent probabilistic dependence. Each random variable has a domain and an associated factor.

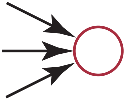


- A decision variable is drawn as an rectangle. Arcs into the node represent information available when the decision is make. Each decision variable has a domain, but no associated factor.

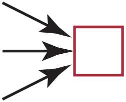


Decisions Networks

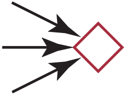
A decision network is a graphical representation of a finite sequential decision problem, with 3 types of nodes:



- A random variable is drawn as an ellipse. Arcs into the node represent probabilistic dependence. Each random variable has a domain and an associated factor.

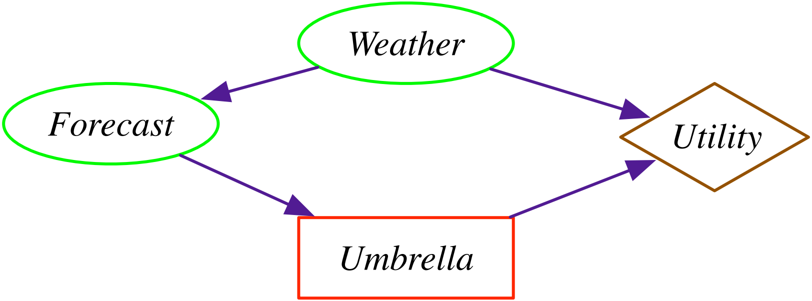


- A decision variable is drawn as an rectangle. Arcs into the node represent information available when the decision is make. Each decision variable has a domain, but no associated factor.



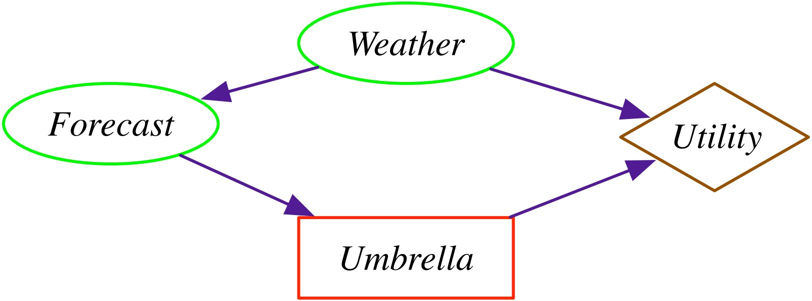
- A utility node is drawn as a diamond. Arcs into the node represent variables that the utility depends on. The utility node has no domain, and a factor on the parents of the node.

Umbrella Decision Network



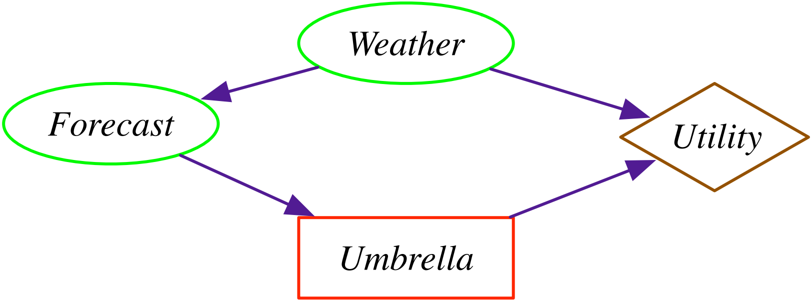
- The agent has to decide whether to take its umbrella.
- It observes

Umbrella Decision Network



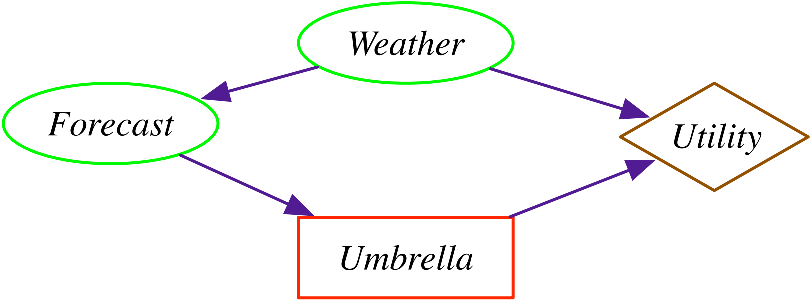
- The agent has to decide whether to take its umbrella.
- It observes the forecast.
- It doesn't observe

Umbrella Decision Network



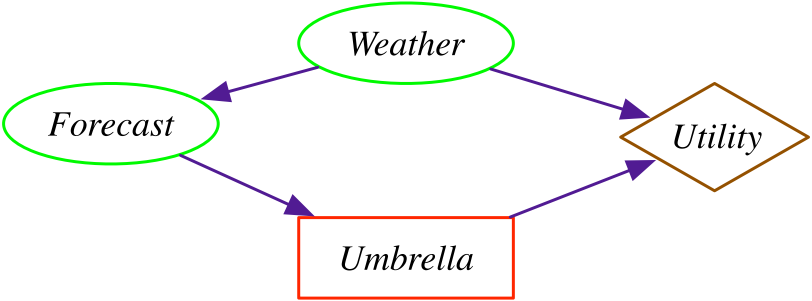
- The agent has to decide whether to take its umbrella.
- It observes the forecast.
- It doesn't observe the weather directly.

Umbrella Decision Network



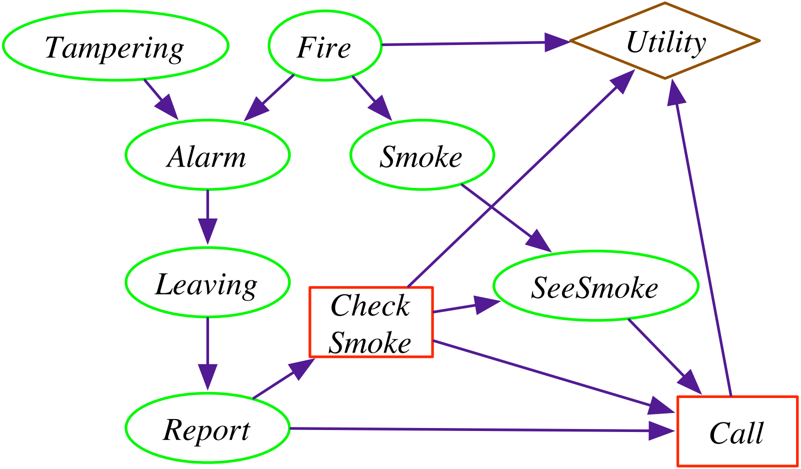
- The agent has to decide whether to take its umbrella.
- It observes the forecast.
- It doesn't observe the weather directly.
- The forecast is a noisy sensor of the weather.

Umbrella Decision Network



- The agent has to decide whether to take its umbrella.
- It observes the forecast.
- It doesn't observe the weather directly.
- The forecast is a noisy sensor of the weather.
- The utility depends on the weather and whether the agent takes the umbrella.

Decision Network for the Alarm Problem



No-forgetting

A No-forgetting decision network is a decision network where:

- The decision nodes are totally ordered. This is the order the actions will be taken.

No-forgetting

A No-forgetting decision network is a decision network where:

- The decision nodes are totally ordered. This is the order the actions will be taken.
- All decision nodes that come before D_i are parents of decision node D_i. Thus the agent remembers its previous actions.

No-forgetting

A No-forgetting decision network is a decision network where:

- The decision nodes are totally ordered. This is the order the actions will be taken.
- All decision nodes that come before D_i are parents of decision node D_i. Thus the agent remembers its previous actions.
- Any parent of a decision node is a parent of subsequent decision nodes. Thus the agent remembers its previous observations.

What should an agent do?

- What an agent should do at any time depends on what it will do in the future.
- What an agent does in the future depends on what it did before.

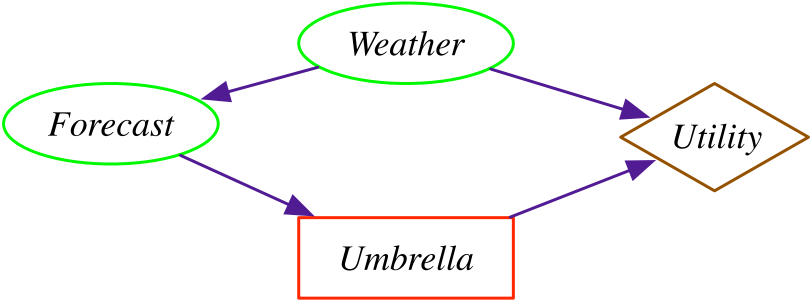
Policies

- A decision function for decision node D_i is a function π_i that specifies what the agent does for each assignment of values to the parents of D_i.
  When it observes O, it does $\pi_i(O)$.

Policies

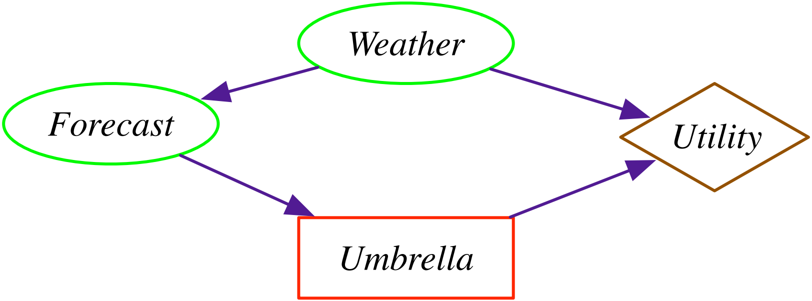
- A decision function for decision node D_i is a function π_i that specifies what the agent does for each assignment of values to the parents of D_i.
  When it observes O, it does $\pi_i(O)$.
- A policy is a sequence of decision functions; one for each decision node.

Umbrella Decision Network



$domain(Forecast) = \{sunny, cloudy, rainy\}$
$domain(Umbbrella) = \{take, leave\}$
Some policies:

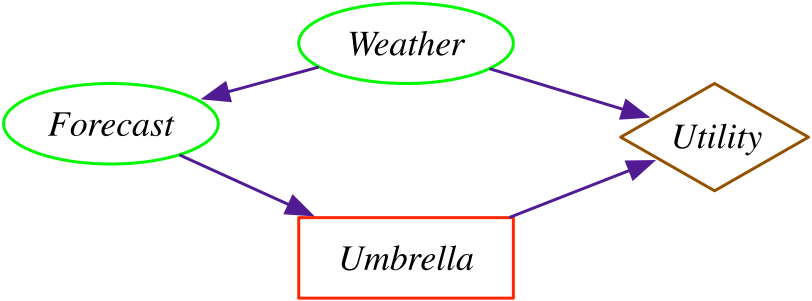
Umbrella Decision Network



$domain(Forecast) = \{sunny, cloudy, rainy\}$
$domain(Umbbrella) = \{take, leave\}$
Some policies:

- take if cloudy else leave
- always take
- always leave

Umbrella Decision Network

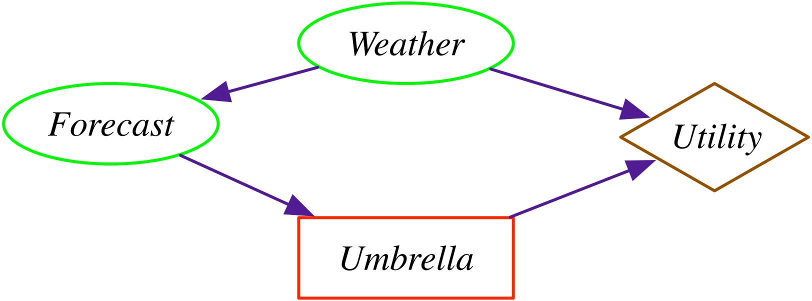


$domain(Forecast) = \{sunny, cloudy, rainy\}$
$domain(Umbbrella) = \{take, leave\}$
Some policies:

- take if cloudy else leave
- always take
- always leave

There are policies

Umbrella Decision Network

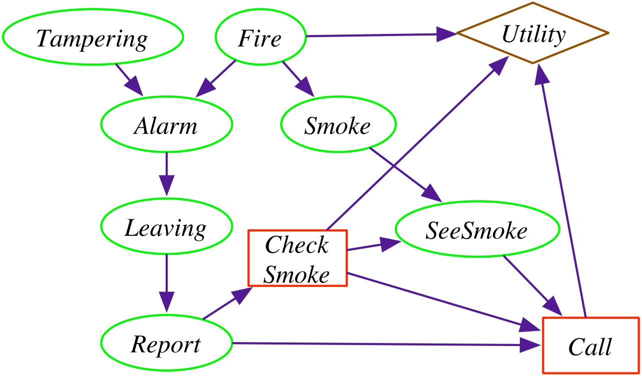


$domain(Forecast) = \{sunny, cloudy, rainy\}$
$domain(Umbbrella) = \{take, leave\}$
Some policies:

- take if cloudy else leave
- always take
- always leave

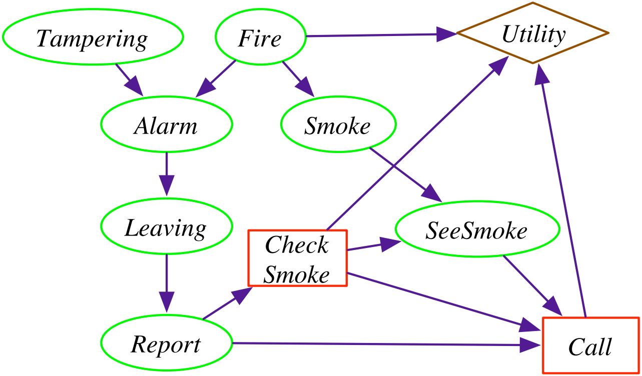
There are $2^3 = 8$ policies

Decision Network for the Alarm Problem



All variables are Boolean. Some policies:

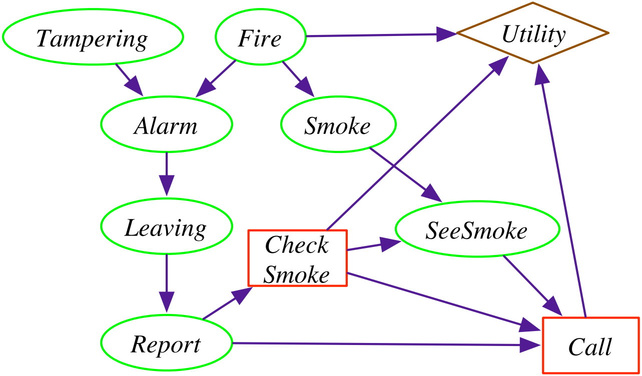
Decision Network for the Alarm Problem



All variables are Boolean. Some policies:

- Never check. Call iff report.
- Check iff report. Call iff report and see smoke.
- Always check. Always call.

Decision Network for the Alarm Problem

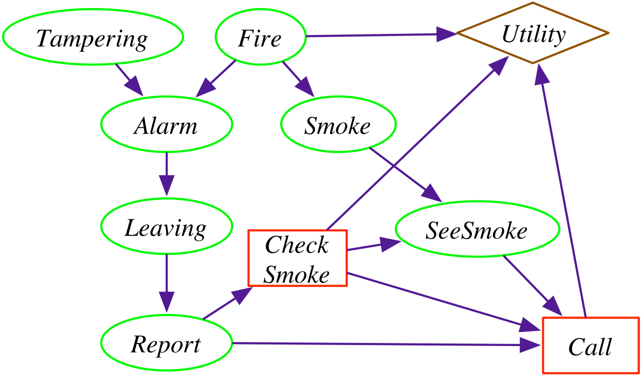


All variables are Boolean. Some policies:

- Never check. Call iff report.
- Check iff report. Call iff report and see smoke.
- Always check. Always call.

There are policies.

Decision Network for the Alarm Problem



All variables are Boolean. Some policies:

- Never check. Call iff report.
- Check iff report. Call iff report and see smoke.
- Always check. Always call.

There are $2^2 * 2^8 = 1024$ policies.

Finding an optimal policy

- Suppose the random variables are $X_1, \ldots, X_n$, and utility depends on $V_{i_1}, \ldots, V_{i_k}$ (random and/or decision variables)

$$\mathbb{E}(u \mid \pi) \quad =$$

Finding an optimal policy

- Suppose the random variables are $X_1, \ldots, X_n$, and utility depends on $V_{i_1}, \ldots, V_{i_k}$ (random and/or decision variables)

$$\begin{aligned}\mathbb{E}(u \mid \pi) &= \sum_{X_1,\ldots,X_n} P(X_1, \ldots, X_n \mid \pi) \times u(V_{i_1}, \ldots, V_{i_k}) \\ &= \sum_{X_1,\ldots,X_n}\end{aligned}$$

Finding an optimal policy

- Suppose the random variables are $X_1, \dots, X_n$, and utility depends on $V_{i_1}, \dots, V_{i_k}$ (random and/or decision variables)

$$\begin{aligned}
\mathbb{E}(u \mid \pi) &= \sum_{X_1,\dots,X_n} P(X_1,\dots,X_n \mid \pi) \times u(V_{i_1},\dots,V_{i_k}) \\
&= \sum_{X_1,\dots,X_n} \prod_{i=1}^{n} P(X_i \mid parents(X_i)_\pi) \times u(V_{i_1},\dots,V_{i_k})
\end{aligned}$$

  where $parents(X_i)_\pi$ include the values chosen by policy π.

Finding an optimal policy

- Suppose the random variables are $X_1, \ldots, X_n$, and utility depends on $V_{i_1}, \ldots, V_{i_k}$ (random and/or decision variables)

$$
\begin{aligned}
\mathbb{E}(u \mid \pi) &= \sum_{X_1,\ldots,X_n} P(X_1, \ldots, X_n \mid \pi) \times u(V_{i_1}, \ldots, V_{i_k}) \\
&= \sum_{X_1,\ldots,X_n} \prod_{i=1}^{n} P(X_i \mid parents(X_i)_\pi) \times u(V_{i_1}, \ldots, V_{i_k})
\end{aligned}
$$

where $parents(X_i)_\pi$ include the values chosen by policy π.

Idea:

  - Sum out all of the random variables to compute expected utility.

Finding an optimal policy

- Suppose the random variables are $X_1, \ldots, X_n$, and utility depends on $V_{i_1}, \ldots, V_{i_k}$ (random and/or decision variables)

$$
\begin{aligned}
\mathbb{E}(u \mid \pi) &= \sum_{X_1,\ldots,X_n} P(X_1,\ldots,X_n \mid \pi) \times u(V_{i_1},\ldots,V_{i_k}) \\
&= \sum_{X_1,\ldots,X_n} \prod_{i=1}^{n} P(X_i \mid parents(X_i)_\pi) \times u(V_{i_1},\ldots,V_{i_k})
\end{aligned}
$$

  where $parents(X_i)_\pi$ include the values chosen by policy π.
  Idea:
  - Sum out all of the random variables to compute expected utility.
  - Choose the policy to maximize the sum: when a decision variable is in a factor with only its parents, select maximum value.

Finding an optimal policy: Search

- Recursive conditioning (chapter 9) can be extended to decision networks.

Finding an optimal policy: Search

- Recursive conditioning (chapter 9) can be extended to decision networks.
- Input: context, set of factors, decision nodes
- Output: optimal policy (set of context-action pairs) for this context and its value

Finding an optimal policy: Search

- Recursive conditioning (chapter 9) can be extended to decision networks.
- Input: context, set of factors, decision nodes
- Output: optimal policy (set of context-action pairs) for this context and its value
- Split on the parents of a decision node before splitting on the decision node (can do if no-forgetting).

Finding an optimal policy: Search

- Recursive conditioning (chapter 9) can be extended to decision networks.
- Input: context, set of factors, decision nodes
- Output: optimal policy (set of context-action pairs) for this context and its value
- Split on the parents of a decision node before splitting on the decision node (can do if no-forgetting).
- To split on a decision node, return the maximum value of its children and the appropriate $(context, action)$

Depth-first search for optimizing decision network

procedure *DN_dfs*(*Con*, *Fs*, *Ds*) returns (*value*, *policy*)
    if $Fs = \{\}$ **then return** $(1, \{\})$

Depth-first search for optimizing decision network

procedure $DN\_dfs(Con, Fs, Ds)$ returns $(value, policy)$
  if $Fs = \{\}$ **then return** $(1, \{\})$
  else if $f \in Fs$ can be evaluated in Con **then**

Depth-first search for optimizing decision network

procedure $DN\_dfs(Con,\ Fs,\ Ds)$ returns $(value, policy)$
    if $Fs = \{\}$ **then return** $(1, \{\})$
    else if $f \in Fs$ can be evaluated in Con **then**
        $(v, p) := prob\_dfs(Con, Fs \setminus \{f\}, Ds)$
        return $(eval(f, Con) * v, p)$

Depth-first search for optimizing decision network

procedure $DN\_dfs(Con, Fs, Ds)$ returns $(value, policy)$
  if $Fs = \{\}$ **then return** $(1, \{\})$
  else if $f \in Fs$ can be evaluated in Con **then**
    $(v, p) := prob\_dfs(Con, Fs \setminus \{f\}, Ds)$
    return $(eval(f, Con) * v, p)$
  else if Con assigns all parents in $D \in Ds$ **then**

Depth-first search for optimizing decision network

```
procedure DN_dfs(Con, Fs, Ds) returns (value, policy)
    if Fs = {} then return (1, {})
    else if f ∈ Fs can be evaluated in Con then
        (v, p) := prob_dfs(Con, Fs \ {f}, Ds)
        return (eval(f, Con) ∗ v, p)
    else if Con assigns all parents in D ∈ Ds then
        max := −∞;   best := ⊥
        for val in domain(D) do
            (v, p) := DN_dfs(Con ∪ {D=val}, Fs, Ds \ {D})
```

Depth-first search for optimizing decision network

```
procedure DN_dfs(Con, Fs, Ds) returns (value, policy)
    if Fs = {} then return (1, {})
    else if f ∈ Fs can be evaluated in Con then
        (v, p) := prob_dfs(Con, Fs \ {f}, Ds)
        return (eval(f, Con) ∗ v, p)
    else if Con assigns all parents in D ∈ Ds then
        max := −∞;   best := ⊥
        for val in domain(D) do
            (v, p) := DN_dfs(Con ∪ {D=val}, Fs, Ds \ {D})
            if v > max then
```

Depth-first search for optimizing decision network

```
procedure DN_dfs(Con, Fs, Ds) returns (value, policy)
    if Fs = {} then return (1, {})
    else if f ∈ Fs can be evaluated in Con then
        (v, p) := prob_dfs(Con, Fs \ {f}, Ds)
        return (eval(f, Con) * v, p)
    else if Con assigns all parents in D ∈ Ds then
        max := −∞;   best := ⊥
        for val in domain(D) do
            (v, p) := DN_dfs(Con ∪ {D=val}, Fs, Ds \ {D})
            if v > max then
                max := v;   best := {⟨Con, D=val⟩} ∪ p
        return (max, best)
```

Depth-first search for optimizing decision network

procedure $DN\_dfs(Con,\ Fs,\ Ds)$ returns $(value, policy)$
  if $Fs = \{\}$ **then return** $(1, \{\})$
  else if $f \in Fs$ can be evaluated in Con **then**
    $(v, p) := prob\_dfs(Con, Fs \setminus \{f\}, Ds)$
    return $(eval(f, Con) * v, p)$
  else if Con assigns all parents in $D \in Ds$ **then**
    $max := -\infty; \quad best := \perp$
    for val in $domain(D)$ **do**
      $(v, p) := DN\_dfs(Con \cup \{D{=}val\}, Fs, Ds \setminus \{D\})$
      if $v > max$ **then**
        $max := v; \quad best := \{\langle Con, D{=}val \rangle\} \cup p$
    return $(max, best)$
  else
    select variable X that is parent of all $D \in Ds$

Depth-first search for optimizing decision network

```
procedure DN_dfs(Con, Fs, Ds) returns (value, policy)
    if Fs = {} then return (1, {})
    else if f ∈ Fs can be evaluated in Con then
        (v, p) := prob_dfs(Con, Fs \ {f}, Ds)
        return (eval(f, Con) ∗ v, p)
    else if Con assigns all parents in D ∈ Ds then
        max := −∞;   best := ⊥
        for val in domain(D) do
            (v, p) := DN_dfs(Con ∪ {D=val}, Fs, Ds \ {D})
            if v > max then
                max := v;   best := {⟨Con, D=val⟩} ∪ p
        return (max, best)
    else
        select variable X that is parent of all D ∈ Ds
        for each value of X, recursivley call DN_dfs, and return
sum of values and union of policies
```

Recursive Conditioning for optimizing decision network

Recursive condition adds to the search algorithm:

- Caching previously computed results

Recursive Conditioning for optimizing decision network

Recursive condition adds to the search algorithm:

- Caching previously computed results
- Forgetting: if a variable isn't in any factors, remove it from the context

Recursive Conditioning for optimizing decision network

Recursive condition adds to the search algorithm:

- Caching previously computed results
- Forgetting: if a variable isn't in any factors, remove it from the context
- Recognizing disconnected components, and solving then independently.

Recursive Conditioning for optimizing decision network

Recursive condition adds to the search algorithm:

- Caching previously computed results
- Forgetting: if a variable isn't in any factors, remove it from the context
- Recognizing disconnected components, and solving then independently.

`AIPython.org` decnNetworks.py

Finding an optimal policy: Variable Elimination

- Create a factor for each conditional probability table and a factor for the utility.

Finding an optimal policy: Variable Elimination

- Create a factor for each conditional probability table and a factor for the utility.
- Repeat:
  - Sum out random variables that are not parents of a decision node.

Finding an optimal policy: Variable Elimination

- Create a factor for each conditional probability table and a factor for the utility.
- Repeat:
  - Sum out random variables that are not parents of a decision node.
  - Let D be last decision variable
    — D is only in a factor f with (some of) its parents.

Finding an optimal policy: Variable Elimination

- Create a factor for each conditional probability table and a factor for the utility.
- Repeat:
  - Sum out random variables that are not parents of a decision node.
  - Let D be last decision variable
    — D is only in a factor f with (some of) its parents.
  - Eliminate D by maximizing.

Finding an optimal policy: Variable Elimination

- Create a factor for each conditional probability table and a factor for the utility.
- Repeat:
  - Sum out random variables that are not parents of a decision node.
  - Let D be last decision variable
    — D is only in a factor f with (some of) its parents.
  - Eliminate D by maximizing. This returns:
    - an optimal decision function for D: $\arg\max_D f$
    - a new factor: $\max_D f$

Finding an optimal policy: Variable Elimination

- Create a factor for each conditional probability table and a factor for the utility.
- Repeat:
  - Sum out random variables that are not parents of a decision node.
  - Let D be last decision variable
    — D is only in a factor f with (some of) its parents.
  - Eliminate D by maximizing. This returns:
    - an optimal decision function for D: $\arg\max_D f$
    - a new factor: $\max_D f$
- until there are no more decision nodes.

Finding an optimal policy: Variable Elimination

- Create a factor for each conditional probability table and a factor for the utility.
- Repeat:
  - Sum out random variables that are not parents of a decision node.
  - Let D be last decision variable
    — D is only in a factor f with (some of) its parents.
  - Eliminate D by maximizing. This returns:
    - an optimal decision function for D: $\arg\max_D f$
    - a new factor: $\max_D f$
- until there are no more decision nodes.
- Sum out the remaining random variables.

Finding an optimal policy: Variable Elimination

- Create a factor for each conditional probability table and a factor for the utility.
- Repeat:
  - Sum out random variables that are not parents of a decision node.
  - Let D be last decision variable
    — D is only in a factor f with (some of) its parents.
  - Eliminate D by maximizing. This returns:
    - an optimal decision function for D: $\arg\max_D f$
    - a new factor: $\max_D f$
- until there are no more decision nodes.
- Sum out the remaining random variables.
- Multiply the factors: this is the expected utility of an optimal policy.

Initial factors for the Umbrella Decision

Weather	Value
norain	0.7
rain	0.3

Weather	Fcast	Value
norain	sunny	0.7
norain	cloudy	0.2
norain	rainy	0.1
rain	sunny	0.15
rain	cloudy	0.25
rain	rainy	0.6

Weather	Umb	Value
norain	take	20
norain	leave	100
rain	take	70
rain	leave	0

. . . first sum out *Weather*.

Eliminating By Maximizing

f:

Fcast	Umb	Val
sunny	take	12.95
sunny	leave	49.0
cloudy	take	8.05
cloudy	leave	14.0
rainy	take	14.0
rainy	leave	7.0

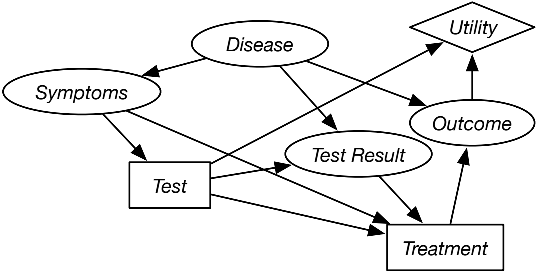
$\max_{Umb} f$:

Fcast	Val
sunny	49.0
cloudy	14.0
rainy	14.0

$\arg\max_{Umb} f$:

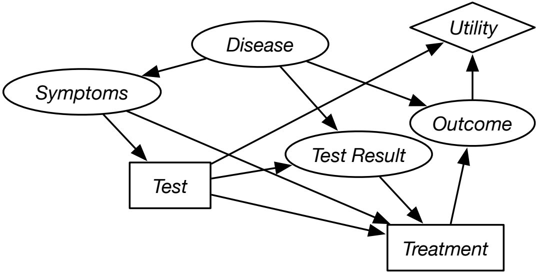
Fcast	Umb
sunny	leave
cloudy	leave
rainy	take

Exercise



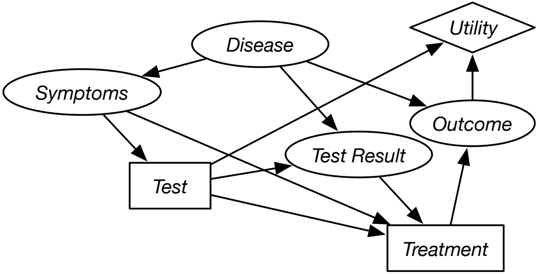
- What are the factors?

Exercise



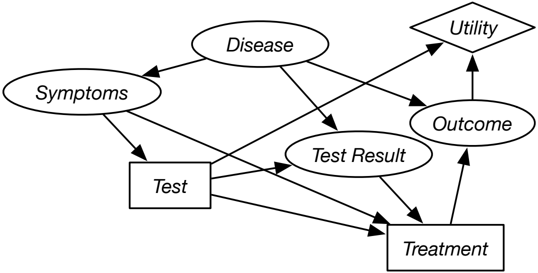
- What are the factors?
- In what order does search split the variables?

Exercise



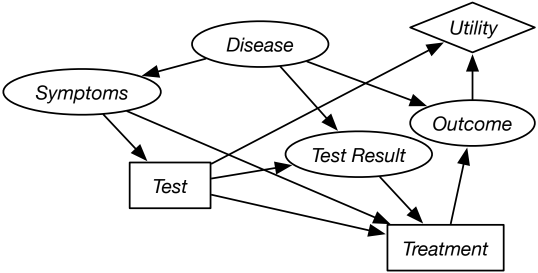
- What are the factors?
- In what order does search split the variables?
- In variable elimination, which random variables get summed out first?

Exercise



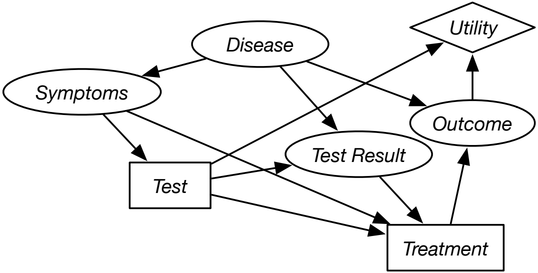
- What are the factors?
- In what order does search split the variables?
- In variable elimination, which random variables get summed out first?
- Which decision variable is eliminated? What factor is created?

Exercise



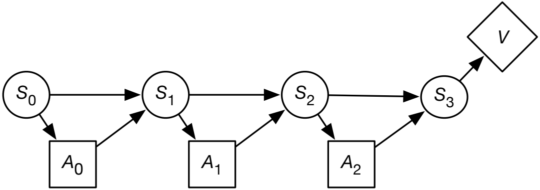
- What are the factors?
- In what order does search split the variables?
- In variable elimination, which random variables get summed out first?
- Which decision variable is eliminated? What factor is created?
- Then what is eliminated (and how)?

Exercise



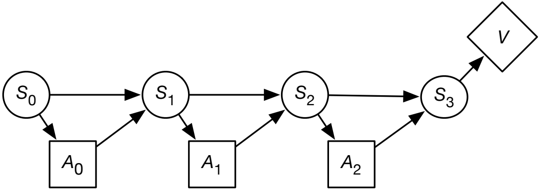
- What are the factors?
- In what order does search split the variables?
- In variable elimination, which random variables get summed out first?
- Which decision variable is eliminated? What factor is created?
- Then what is eliminated (and how)?
- What factors are created after maximization?

Chain



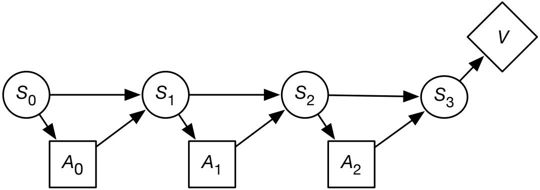
- Is this decision network no-forgetting?

Chain



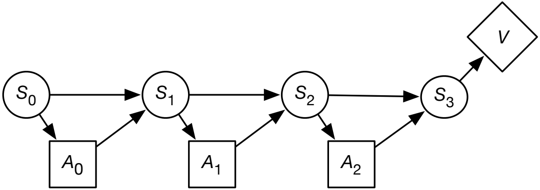
- Is this decision network no-forgetting?
- What happens with the naive search with variable splitting ordering from left to right?

Chain



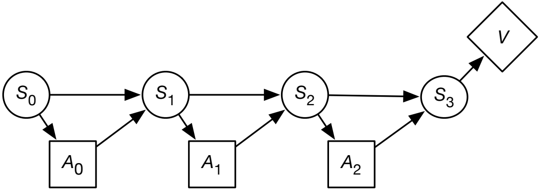
- Is this decision network no-forgetting?
- What happens with the naive search with variable splitting ordering from left to right?
- What happens with recursive conditioning with variable splitting ordering from left to right?

Chain



- Is this decision network no-forgetting?
- What happens with the naive search with variable splitting ordering from left to right?
- What happens with recursive conditioning with variable splitting ordering from left to right?
- What happens with variable elimination with variable splitting ordering from right to left?

Chain



- Is this decision network no-forgetting?
- What happens with the naive search with variable splitting ordering from left to right?
- What happens with recursive conditioning with variable splitting ordering from left to right?
- What happens with variable elimination with variable splitting ordering from right to left?

`AIPython` decnNetworks.py ch3

Complexity of finding an optimal policy

Decision D has k binary parents, and has b possible actions:

- there are assignments of values to the parents.

Complexity of finding an optimal policy

Decision D has k binary parents, and has b possible actions:

- there are 2^k assignments of values to the parents.
- there are   different decision functions.

Complexity of finding an optimal policy

Decision D has k binary parents, and has b possible actions:

- there are 2^k assignments of values to the parents.
- there are b^{2^k} different decision functions.
- To optimize D, the algorithm does optimizations.

Complexity of finding an optimal policy

Decision D has k binary parents, and has b possible actions:

- there are 2^k assignments of values to the parents.
- there are b^{2^k} different decision functions.
- To optimize D, the algorithm does 2^k optimizations.
  The time complexity to optimize D is O( ).

Complexity of finding an optimal policy

Decision D has k binary parents, and has b possible actions:

- there are 2^k assignments of values to the parents.
- there are b^{2^k} different decision functions.
- To optimize D, the algorithm does 2^k optimizations.
  The time complexity to optimize D is $O(b * 2^k)$.

Complexity of finding an optimal policy

Decision D has k binary parents, and has b possible actions:

- there are 2^k assignments of values to the parents.
- there are b^{2^k} different decision functions.
- To optimize D, the algorithm does 2^k optimizations.
  The time complexity to optimize D is $O(b * 2^k)$.

If the decision variables are $D_i, \ldots, D_n$ and decision D_i has k_i binary parents and b_i possible actions:

- there are policies.

Complexity of finding an optimal policy

Decision D has k binary parents, and has b possible actions:

- there are 2^k assignments of values to the parents.
- there are b^{2^k} different decision functions.
- To optimize D, the algorithm does 2^k optimizations. The time complexity to optimize D is $O(b * 2^k)$.

If the decision variables are $D_i, \ldots, D_n$ and decision D_i has k_i binary parents and b_i possible actions:

- there are $\prod_{i=1}^{n} b_i^{2^{k_i}}$ policies.
- optimizing in the variable elimination algorithm takes

$$O\left(\qquad\qquad \right) \text{ time}$$

Complexity of finding an optimal policy

Decision D has k binary parents, and has b possible actions:

- there are 2^k assignments of values to the parents.
- there are b^{2^k} different decision functions.
- To optimize D, the algorithm does 2^k optimizations. The time complexity to optimize D is $O(b * 2^k)$.

If the decision variables are $D_i, \ldots, D_n$ and decision D_i has k_i binary parents and b_i possible actions:

- there are $\prod_{i=1}^{n} b_i^{2^{k_i}}$ policies.
- optimizing in the variable elimination algorithm takes $$O\left(\sum_{i=1}^{n} b_i * 2^{k_i}\right)$$ time
- The dynamic programming algorithm is much more efficient than searching through policy space.

Value of Information

- The value of information X for decision D

Value of Information

- The value of information X for decision D is the utility of the network with an arc from X to D ($+$ no-forgetting arcs) minus the utility of the network without the arc.

Value of Information

- The value of information X for decision D is the utility of the network with an arc from X to D ($+$ no-forgetting arcs) minus the utility of the network without the arc.
  AIPython.org decnNetworks.py: `umbrella_dn` vs `umbrella_dn_2p`

Value of Information

- The value of information X for decision D is the utility of the network with an arc from X to D ($+$ no-forgetting arcs) minus the utility of the network without the arc.
  AIPython.org decnNetworks.py: `umbrella_dn` vs `umbrella_dn_2p`
- The value of information is always

Value of Information

- The value of information X for decision D is the utility of the network with an arc from X to D (+ no-forgetting arcs) minus the utility of the network without the arc. AIPython.org decnNetworks.py: `umbrella_dn` vs `umbrella_dn_2p`
- The value of information is always non-negative.
- It is positive only if

Value of Information

- The value of information X for decision D is the utility of the network with an arc from X to D ($+$ no-forgetting arcs) minus the utility of the network without the arc.
  AIPython.org decnNetworks.py: `umbrella_dn` vs `umbrella_dn_2p`
- The value of information is always non-negative.
- It is positive only if the agent changes its action depending on X.

Value of Information

- The value of information X for decision D is the utility of the network with an arc from X to D ($+$ no-forgetting arcs) minus the utility of the network without the arc.
  AIPython.org decnNetworks.py: `umbrella_dn` vs `umbrella_dn_2p`
- The value of information is always non-negative.
- It is positive only if the agent changes its action depending on X.
- The value of information provides a bound on how much an agent should be prepared to pay for a sensor. How much is a better weather forecast worth?

Value of Information

- The value of information X for decision D is the utility of the network with an arc from X to D (+ no-forgetting arcs) minus the utility of the network without the arc.
  AIPython.org decnNetworks.py: `umbrella_dn` vs `umbrella_dn_2p`
- The value of information is always non-negative.
- It is positive only if the agent changes its action depending on X.
- The value of information provides a bound on how much an agent should be prepared to pay for a sensor. How much is a better weather forecast worth?
- We need to be careful when adding an arc would create a cycle. E.g., how much would it be worth knowing whether the fire truck will arrive quickly when deciding whether to call them?

Value of Control

- The value of control of a variable X is the value of the network when X you make a decision variable (and add no-forgetting arcs) minus the value of the network when X is a random variable.

Value of Control

- The value of control of a variable X is the value of the network when X you make a decision variable (and add no-forgetting arcs) minus the value of the network when X is a random variable.
- You need to be explicit about what information is available when you control X.

Value of Control

- The value of control of a variable X is the value of the network when X you make a decision variable (and add no-forgetting arcs) minus the value of the network when X is a random variable.
- You need to be explicit about what information is available when you control X.
- If you control X without observing, controlling X can be worse than observing X. E.g., controlling a thermometer.

Value of Control

- The value of control of a variable X is the value of the network when X you make a decision variable (and add no-forgetting arcs) minus the value of the network when X is a random variable.
- You need to be explicit about what information is available when you control X.
- If you control X without observing, controlling X can be worse than observing X. E.g., controlling a thermometer.
- If you keep the parents the same, the value of control is always non-negative.