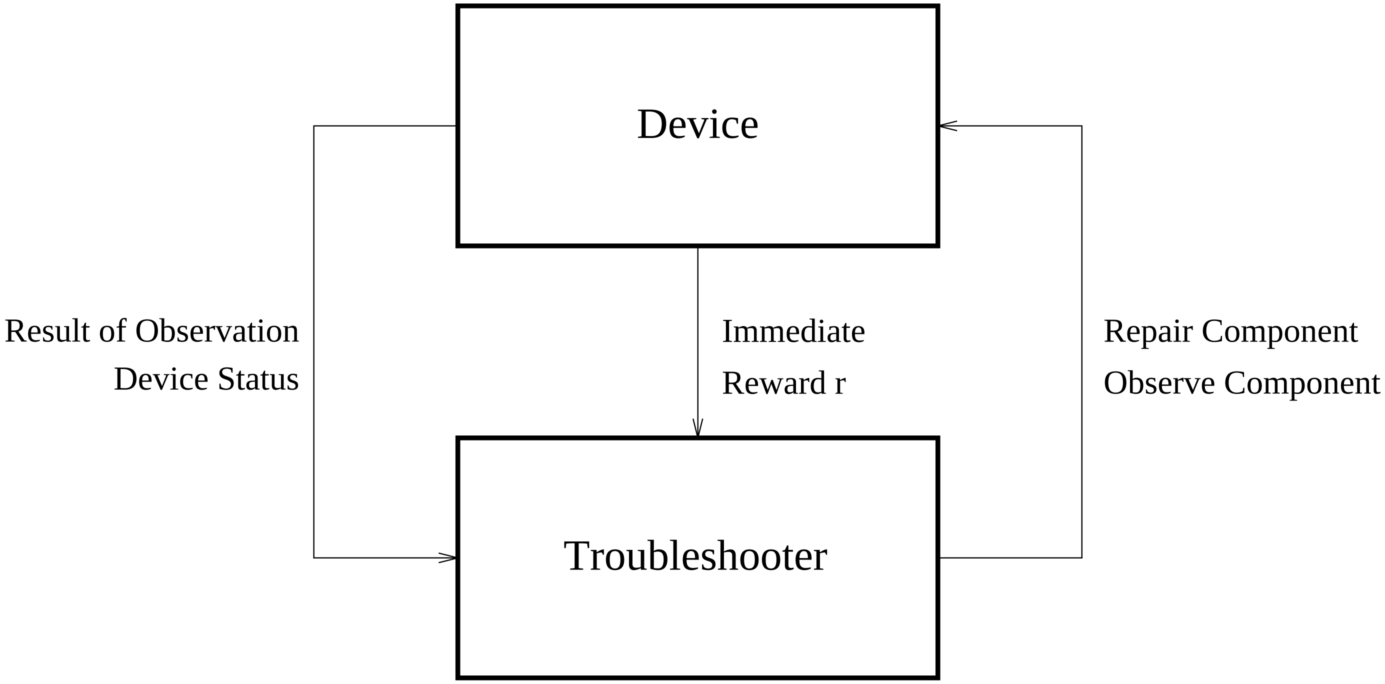
Diagnosis and Bayesian Networks

Diagnosis and repair of faults in designed systems

- **Photocopiers**
- **Automobiles**
- **Jet aircraft**
- **Digital circuits**
- **Spacecraft**

Troubleshooting Involves Choosing a Sequence of Actions



- **Like reinforcement learning**.
  But troubleshooter *cannot observe entire state* at each step.
- **Problem:** How do we choose actions to minimize expected total cost of repair? (i.e., maximize expected total reward?)

The Diagnosis Problem

- **state:** An assignment of values to each of the variables describing the device.

  SparkPlugs = Bad, Distributor = Ok, Starter = Ok, BatteryAge = new, EngineCranks = NoCrank, Starts = no, . . .

- **start state:** An unknown state where Starts = no.

- **actions:**

  Observe component: can be applied to SparkPlugs, Distributor, Starter, BatteryAge, Alternator, FanBelt, EngineCranks, Starts, Radio, GasGauge, Lights

  Repair component: can be applied to SparkPlugs, Distributor, FuelPump, Starter, BatteryAge, Alternator, FanBelt, GasIntank, Leak, Leak2, Radio, Lights

- **reward function:** Negative reward (cost) for each action.

  Positive reward for getting into a state where Starts = yes.

Key point: States are states of the device, not of the troubleshooter!

Observation actions do not change the state of the device (assumption).

Repair actions do change the state of the device.

Probability Review (1): Random Variables and Expected Values

Let X be a random variable with values $x_1, \ldots x_n$:

$$\begin{array}{rccccc} \text{value of } X: & x_1 & x_2 & x_3 & \ldots & x_n \\ P(X): & p_1 & p_2 & p_3 & \ldots & p_n \end{array}$$

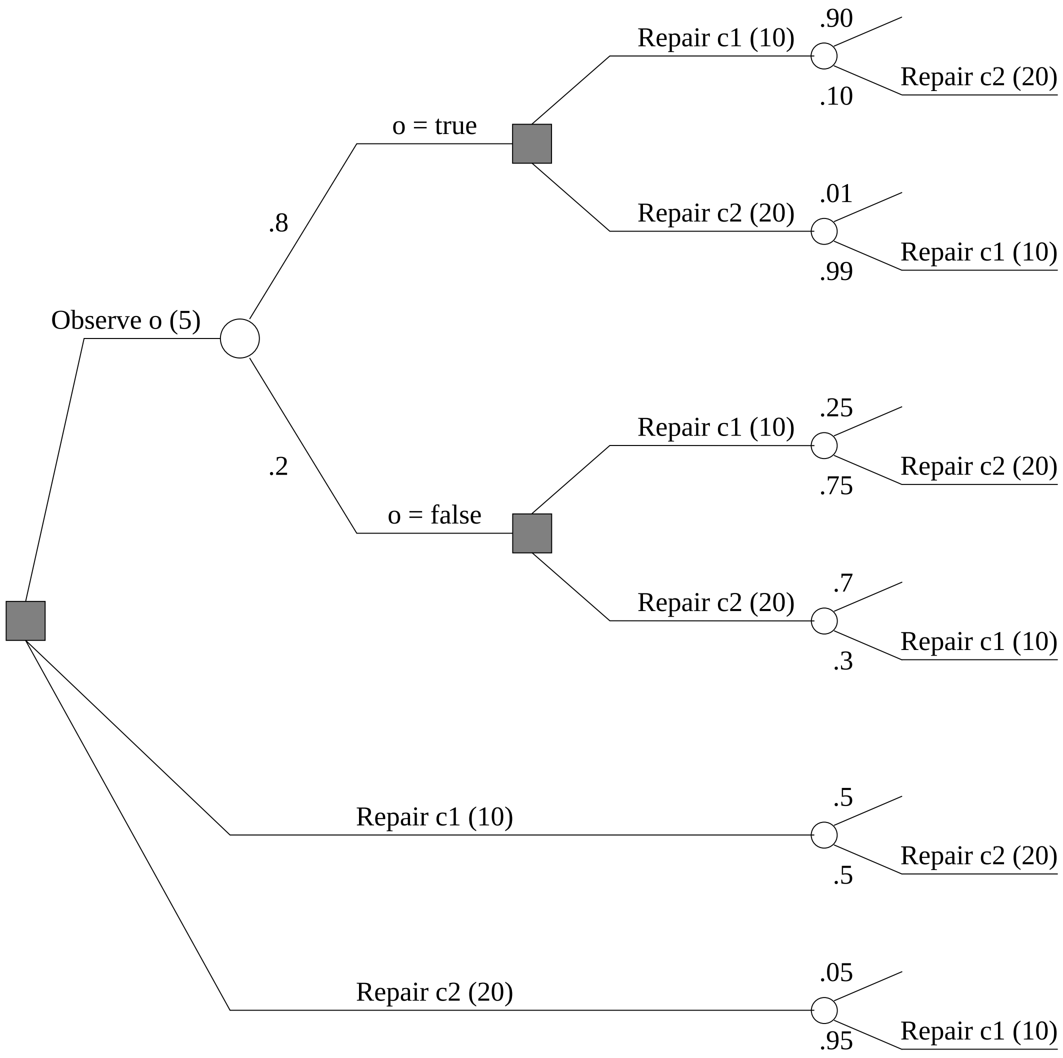
Then the *expected value of* X is the weighted sum:

$$E(X) = \sum_{i=1}^{n} p_i x_i$$

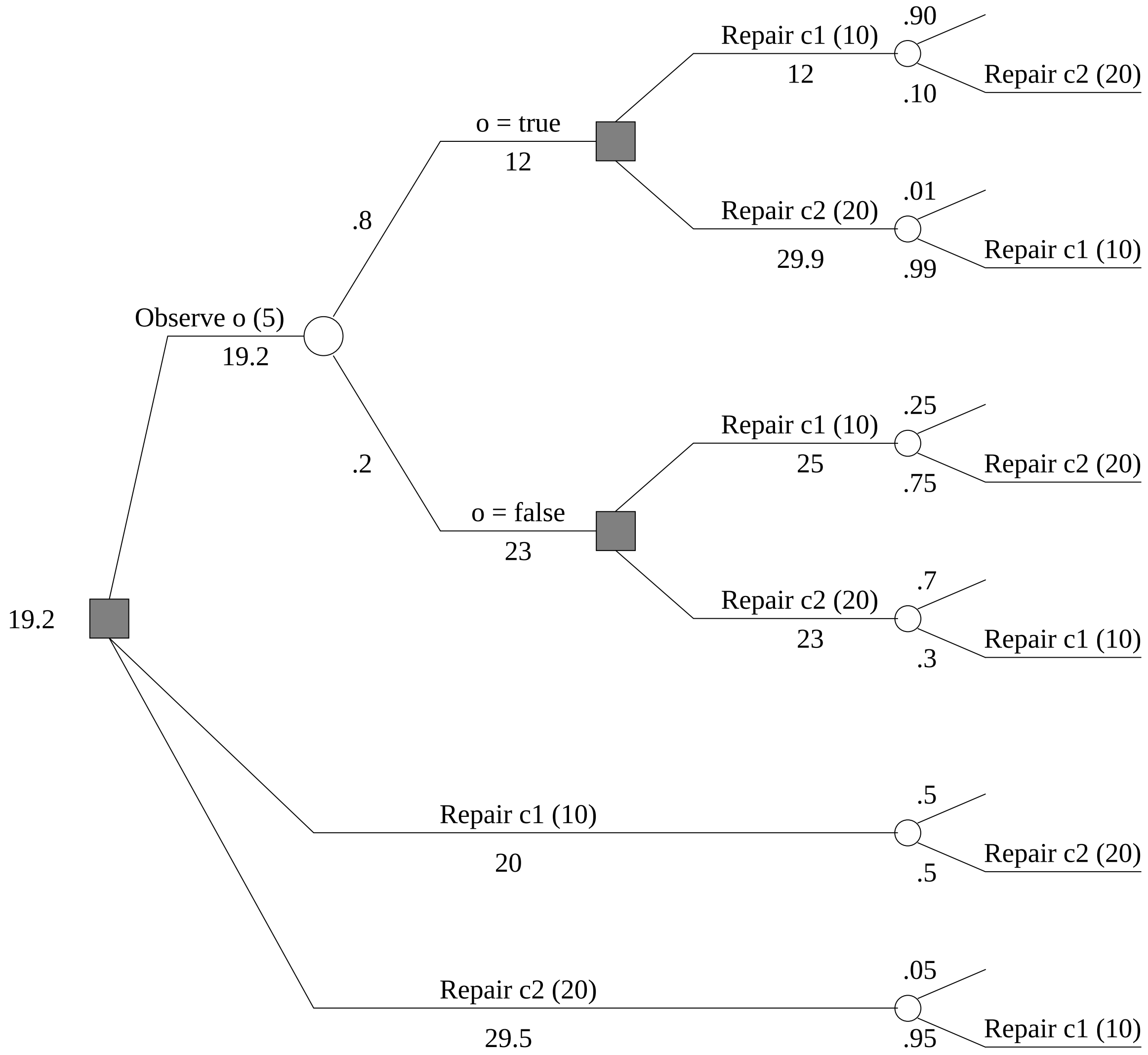
Computing Optimal Policy via Dynamic Programming (RL)

Device: two components: c_1, c_2

Actions: observe o (5), repair c_1 (10), repair c_2 (20)



Value Function and Optimal Policy



Full Dynamic Programming is Infeasible

- **Infeasible for larger problems.** A device with 5 components and 3 observations would have a tree with more than 340,000 leaves.
- **Alternative 1:** Train a reinforcement learning system.
- **Alternative 2:** Directly compute a good (optimal?) policy for special cases.

A Special Case Where Optimal Policy can be Computed

- **There is a single problem-defining node**. E.g., engine-starts.
- **In the start state, device is malfunctioning**
- **Single Fault Assumption:** Exactly one component is broken.
- **After each repair, the problem-defining node is always observed**
- **Only two kinds of components:**

  (a) observable and repairable and (b) unobservable and repairable.
- **Only two kinds of actions:**

  For *observable and repairable components*, observe, if broken, repair.

  For *unobservable and repairable components*, repair.
- **Costs of actions are fixed and independent of order.**
- **No other observations are permitted.**

The Policy Has A Simple Form

A policy in this problem just specifies *the order in which the components will be processed.*

```
troubleShoot(list<component> policy)
{
   listIterator<component> itr(policy);
   do {
     if (itr().isObservable()) {
        itr().Observe();
        if(itr().isBroken()) {
           itr().Repair();
           return;  // we are done, by SFA
        }
     } else {
       itr().Repair();
       if (isFixed()) return;
     }
     ++itr;
   } while !itr;
   // Control reaches here if all observable components were not
   // broken and we have replaced all other components.
}
```

Computing the Value of the Policy

- **Cost of observing component c_i:** $Obs(c_i)$.

  If a component is not observable, this is the cost of repairing it.

- **Cost of repairing component c_i:** $Rep(c_i)$.

  (includes cost of re-observing the problem-defining variable.)

  If a component is not observable, this is zero.

- **Probability that component c_i is broken:** p_{c_i}.

Let $\pi = (c_1, c_2, \ldots, c_n)$ be the policy to be evaluated.

Let $f_\pi(s_0)$ be the value of the start state s_0 under policy π.

$$f_\pi(s_0) = \sum_{i=1}^{n} \left[\left(1 - \sum_{j=1}^{i-1} p_{c_j} \right) Obs(c_i) + p_{c_i} Rep(c_i) \right]$$

The ith term is probability that the first $i-1$ components were ok (times the cost of observing the ith component) plus the probability that the ith component is broken times the cost of repairing it.

Computing the Value of the Policy (2)

Suppose we compare two policies:

$$\pi_1 = (c_1, \dots, c_{k-1}, c_k, c_{k+1}, \dots, c_n)$$

$$\pi_2 = (c_1, \dots, c_{k-1}, c_{k+1}, c_k, \dots, c_n)$$

$$f_{\pi_1}(s_0) - f_{\pi_2}(s_0) =$$

$$\left[\left(1 - \sum_{j=1}^{k-1} p_{c_j}\right) Obs(c_k) + p_{c_k} Rep(c_k)\right] \; + \; \left[\left(1 - \sum_{j=1}^{k-1} p_{c_j} - p_{c_k}\right) Obs(c_{k+1}) + p_{c_{k+1}} Rep(c_{k+1})\right] -$$
$$\left[\left(1 - \sum_{j=1}^{k-1} p_{c_j}\right) Obs(c_{k+1}) + p_{c_{k+1}} Rep(c_{k+1})\right] \; - \; \left[\left(1 - \sum_{j=1}^{k-1} p_{c_j} - p_{c_{k+1}}\right) Obs(c_k) + p_{c_k} Rep(c_k)\right]$$

Computing the Value of a Policy (3)

Let $A = \left(1 - \sum_{j=1}^{k-1} p_{c_j}\right)$

Then this can be rewritten as

$f_{\pi_1}(s_0) - f_{\pi_2}(s_0) =$

$$\left[A \cdot Obs(c_k) + p_{c_k} Rep(c_k)\right] \;+\; \left[(A - p_{c_k}) Obs(c_{k+1}) + p_{c_{k+1}} Rep(c_{k+1})\right] -$$
$$\left[A \cdot Obs(c_{k+1}) + p_{c_{k+1}} Rep(c_{k+1})\right] \;-\; \left[(A - p_{c_{k+1}}) Obs(c_k) + p_{c_k} Rep(c_k)\right]$$

$f_{\pi_1}(s_0) - f_{\pi_2}(s_0) =$

$$A \cdot Obs(c_k) + p_{c_k} Rep(c_k) \;+\; A \cdot Obs(c_{k+1}) - p_{c_k} Obs(c_{k+1}) + p_{c_{k+1}} Rep(c_{k+1})$$
$$-A \cdot Obs(c_{k+1}) - p_{c_{k+1}} Rep(c_{k+1}) \;-\; A \cdot Obs(c_k) + p_{c_{k+1}} Obs(c_k) - p_{c_k} Rep(c_k)$$

$f_{\pi_1}(s_0) - f_{\pi_2}(s_0) = p_{c_{k+1}} Obs(c_k) - p_{c_k} Obs(c_{k+1})$

Computing the Value of a Policy (4)

We will prefer policy π_1 if

$f_{\pi_1}(s_0) - f_{\pi_2}(s_0) < 0$

which will be true if

$p_{c_{k+1}} Obs(c_k) - p_{c_k} Obs(c_{k+1}) < 0$

or

$$\frac{p_{c_k}}{Obs(c_k)} > \frac{p_{c_{k+1}}}{Obs(c_{k+1})}$$

From this, we can conclude that the optimal policy is to process the components in descending order of $p_{c_i}/Obs(c_i)$.

Extending The Analysis

- **Multiple Faults**
- **Auxiliary Observation Actions**

  Observing components that cannot be repaired or that do not lie along a causal pathway to the problem-defining node.

  Examples:

  - EngineCranks is observable by not repairable
  - Lights is observable but does not lie along causal path to Starts.

Solving Multiple Faults Iteratively

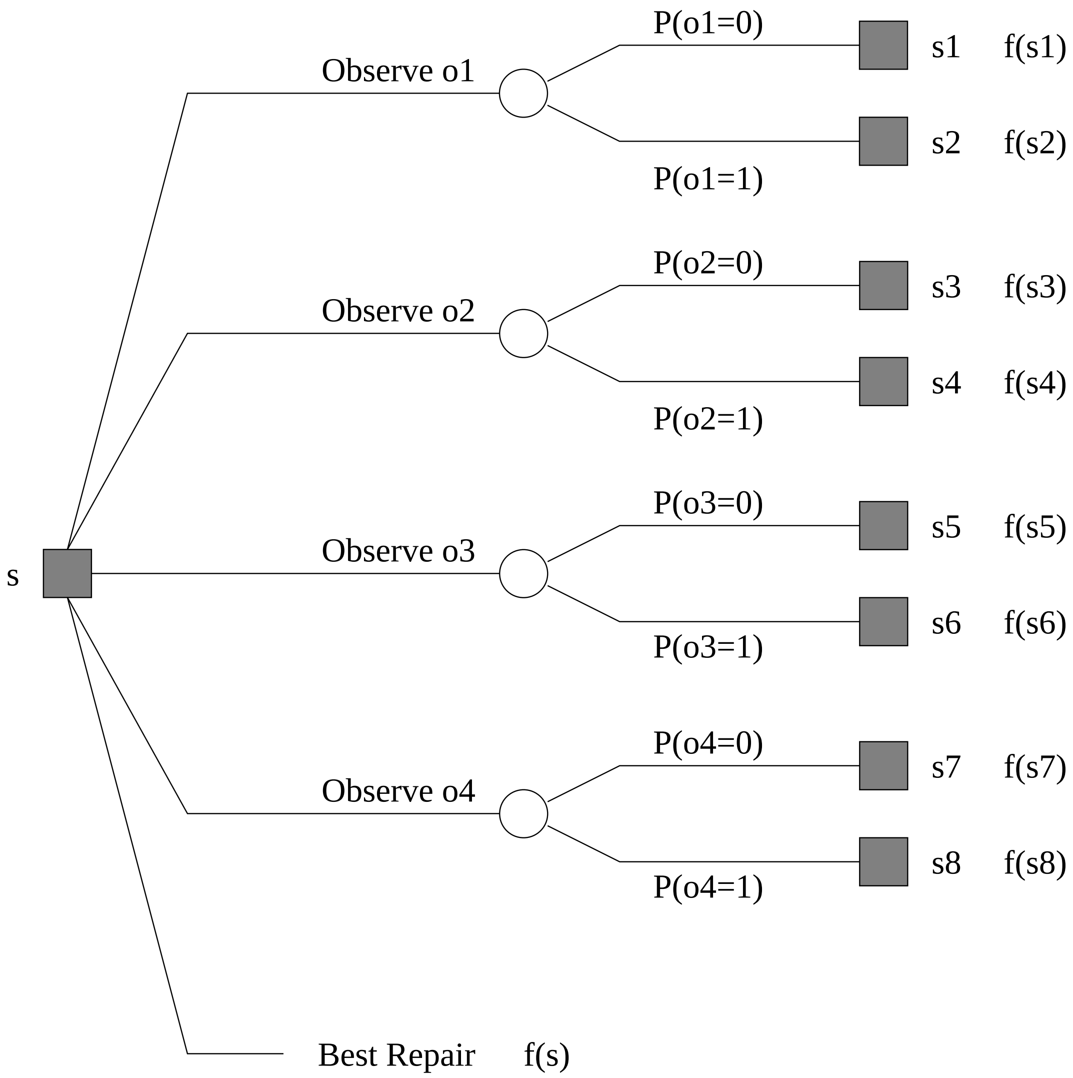
1. Compute the p_{c_i} values given current information.
2. Observe the (as yet unobserved) component with highest ratio $p_{c_i}/Obs(c_i)$. (If not observable, simply repair it.)
3. If the component is faulty, then repair it.
4. If a component was repaired, terminate if the device is working. Otherwise go to step 1.

Two big differences.

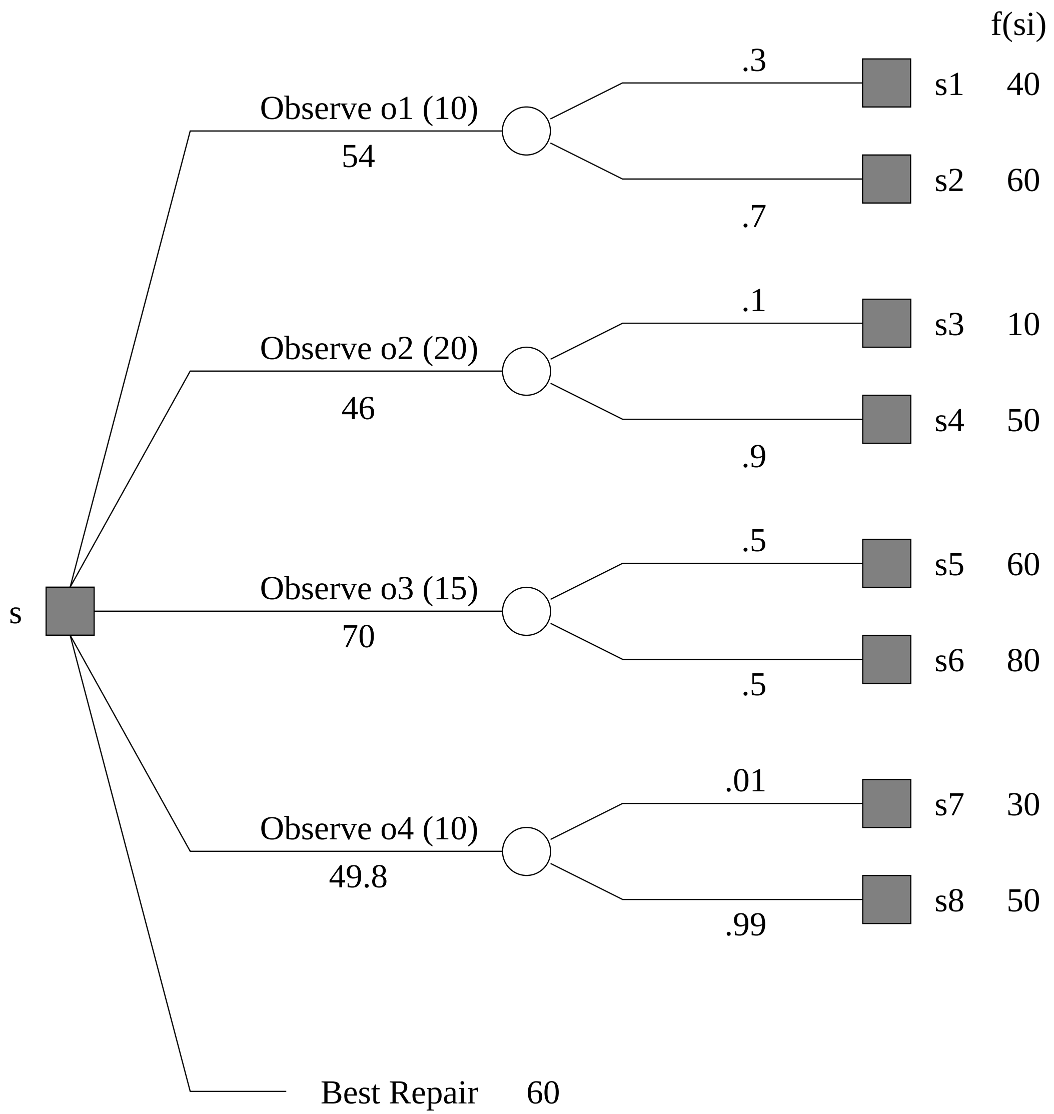
- **We don't assume the first fault repaired fixes the problem.**
- **We recompute the probabilities (and hence the policy) after each repair**. This reflects the fact that $\sum_i p_{c_i} \geq 1$.

Auxiliary Observation Actions

Perform one-step lookahead search and evaluate the resulting states *assuming all subsequent actions are repairs.* (i.e., so that we can compute the values of those states.)



Example



The best greedy action is to Observe o_2. However, the optimal action might be to attempt a repair and then make an observation followed by other repairs.

Probability Computations

To execute this procedure, we need to know many different probabilities:

- **Probability of each possible outcome of an observation**
- **Probability that a component repair will fix the malfunction**

We need to recompute these probabilities after each observation.

Probability Review (2): Joint Distribution

Suppose we have three random variables: Gas, GasGauge, Starts

- Gas: true (there is gas in tank), false (tank is empty)
- GasGauge: true (shows gas in tank), false (shows empty)
- Starts: true (car starts), false (car doesn't start)

The joint probability distribution gives the probability of each of the 8 possible combinations:

	starts		−starts	
	gas	−gas	gas	−gas
GasGauge	.5040	.0006	.2160	.0594
−GasGauge	.0560	.0014	.0240	.1386

These probabilities sum to 1.0.

The Central Problem

- **The joint distribution is sufficient**. We can compute all of the probabilities we need from the joint distribution.
- **The joint distribution is intractable**. The number of cells in the table grows exponentially with the number of random variables.

Bayesian Networks provide a solution to this central problem.

Before we can discuss them, we need to review more things from basic probability.

Probability Review (3): Notation

	Starts		–Starts	
	Gas	–Gas	Gas	–Gas
GasGauge	.5040	.0006	.2160	.0594
–GasGauge	.0560	.0014	.0240	.1386

We will use three notations:

- **Single Cell:** $P(\mathsf{Gas}, -\mathsf{GasGauge}, \mathsf{Starts}) = .0560.$
- **Whole Table:** $\mathbf{P}(\mathsf{Gas}, \mathsf{GasGauge}, \mathsf{Starts})$ stands for the entire joint distribution.
- **Summation Shorthand:** $\sum_{\mathsf{Gas}} P(\mathsf{Gas}, \mathsf{GasGauge}, \mathsf{Starts}) = P(\mathsf{Gas}, \mathsf{GasGauge}, \mathsf{Starts}) + P(-\mathsf{Gas}, \mathsf{GasGauge}, \mathsf{Starts})$ for each possible value of GasGauge and Starts.

  We also write this as $\sum_{\mathsf{Gas}} \mathbf{P}(\mathsf{Gas}, \mathsf{GasGauge}, \mathsf{Starts}) = \mathbf{P}(\mathsf{GasGauge}, \mathsf{Starts})$

Probability Review (4): Marginal Probabilities

Suppose we want to know the probability that $\mathsf{GasGauge}$ is true. We obtain this by summing over the entire top row of the table.

$$P(\mathsf{GasGauge}) = .5040 + .0006 + .2160 + .0594 = 0.78$$

Suppose we want to know the probability that Gas is false:

$$P(-\mathsf{Gas}) = .0006 + .0014 + .0594 + .1386 = 0.2$$

This is called *marginalizing*. We write it as

$$P(-\mathsf{Gas}) = \sum_{GasGauge,Starts} P(-\mathsf{Gas}, \mathsf{GasGauge}, \mathsf{Starts})$$

$\mathbf{P}(\mathsf{Gas})$ is called the *marginal distribution* of Gas.

Probability Review (5): Conditional Probability

Suppose we want to know the probability that the car will start given that we have observed GasGauge and found it to be true.

$$P(\mathsf{Starts}|\mathsf{GasGauge}) = \frac{P(\mathsf{Starts}, \mathsf{GasGauge})}{P(\mathsf{GasGauge})} = \frac{.5040 + .0006}{.5040 + .0006 + .2160 + .0594} = .6469$$

	Starts		–Starts	
	Gas	–Gas	Gas	–Gas
GasGauge	.5040	.0006	.2160	.0594
–GasGauge	.0560	.0014	.0240	.1386

Since GasGauge is true, one of the four situations from the top row of the table must hold. We *renormalize* the probabilities and then marginalize to compute $\mathbf{P}(\mathsf{Starts})$.

Probability Review (6): Conditional Probability Distribution

The conditional probability distribution $\mathbf{P}(\mathsf{GasGauge}|\mathsf{Gas})$:

	GasGauge	¬GasGauge
Gas	.9	.1
¬Gas	.3	.7

Each row must sum to 1.

Probability Review (7): Chain Rule of Probability

$$P(A, B, C) = P(A|B, C) \cdot P(B|C) \cdot P(C)$$

for any number of variables in any order.

Proof:

$$\begin{aligned}
P(A, B, C) &= \frac{P(A, B, C)}{P(B, C)} \cdot P(B, C) \\
&= P(A|B, C) \cdot P(B, C) \\
&= P(A|B, C) \cdot \frac{P(B, C)}{P(C)} \cdot P(C) \\
&= P(A|B, C) \cdot P(B|C) \cdot P(C)
\end{aligned}$$

This holds for distributions as well:

$$\mathbf{P}(A, B, C) = \mathbf{P}(A|B, C) \cdot \mathbf{P}(B|C) \cdot \mathbf{P}(C)$$

Probability Review (8): Independence

Two random variables X and Y are *independent* if $P(X, Y) = P(X) \cdot P(Y)$ for all values of X and Y. This can be written as $\mathbf{P}(X, Y) = \mathbf{P}(X) \cdot \mathbf{P}(Y)$. Intuitively, there is no causal connection betwen X and Y.

Example:

- **X**: coin with probability $X = true$ of .4, and $X = false$ of .6.
- **Y**: coin with probability $Y = true$ of .8, and $Y = false$ of .2.

Joint distribution:

	$X = true$	$X = false$
$Y = true$	.32	.48
$Y = false$	.08	.12

Probability Review (9): Conditional Independence

Two random variables X and Y are *conditionally independent given* Z if $P(X, Y|Z) = P(X|Z) \cdot P(Y|Z)$ for all values of X, Y, and Z.

Example:

$$\mathbf{P}(\mathsf{Starts}, \mathsf{GasGauge}|\mathsf{Gas}) = \mathbf{P}(\mathsf{Starts}|\mathsf{Gas}) \cdot \mathbf{P}(\mathsf{GasGauge}|\mathsf{Gas})$$

	Starts		–Starts	
	GasGauge	–GasGauge	GasGauge	–GasGauge
Gas	.630	.070	.270	.030
–Gas	.003	.007	.297	.693

	GasGauge	–GasGauge
Gas	.9	.1
–Gas	.3	.7

	Starts	–Starts
Gas	.7	.3
–Gas	.01	.99

Simplifying the Joint Distribution by Conditional Independence

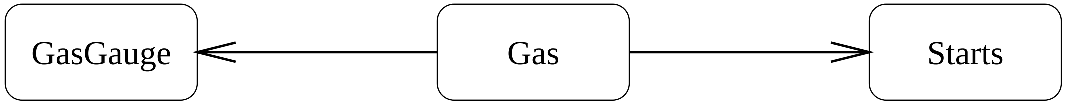
By the definition of conditional probability, we have

$$\mathbf{P}(\mathsf{Gas}, \mathsf{GasGauge}, \mathsf{Starts}) = \mathbf{P}(\mathsf{GasGauge}, \mathsf{Starts} | \mathsf{Gas})\mathbf{P}(\mathsf{Gas})$$

But by conditional independence, the first factor can be rewritten as

$$\mathbf{P}(\mathsf{Gas}, \mathsf{GasGauge}, \mathsf{Starts}) = \mathbf{P}(\mathsf{GasGauge} | \mathsf{Gas})\mathbf{P}(\mathsf{Starts} | \mathsf{Gas})\mathbf{P}(\mathsf{Gas})$$

Bayes Network Diagram

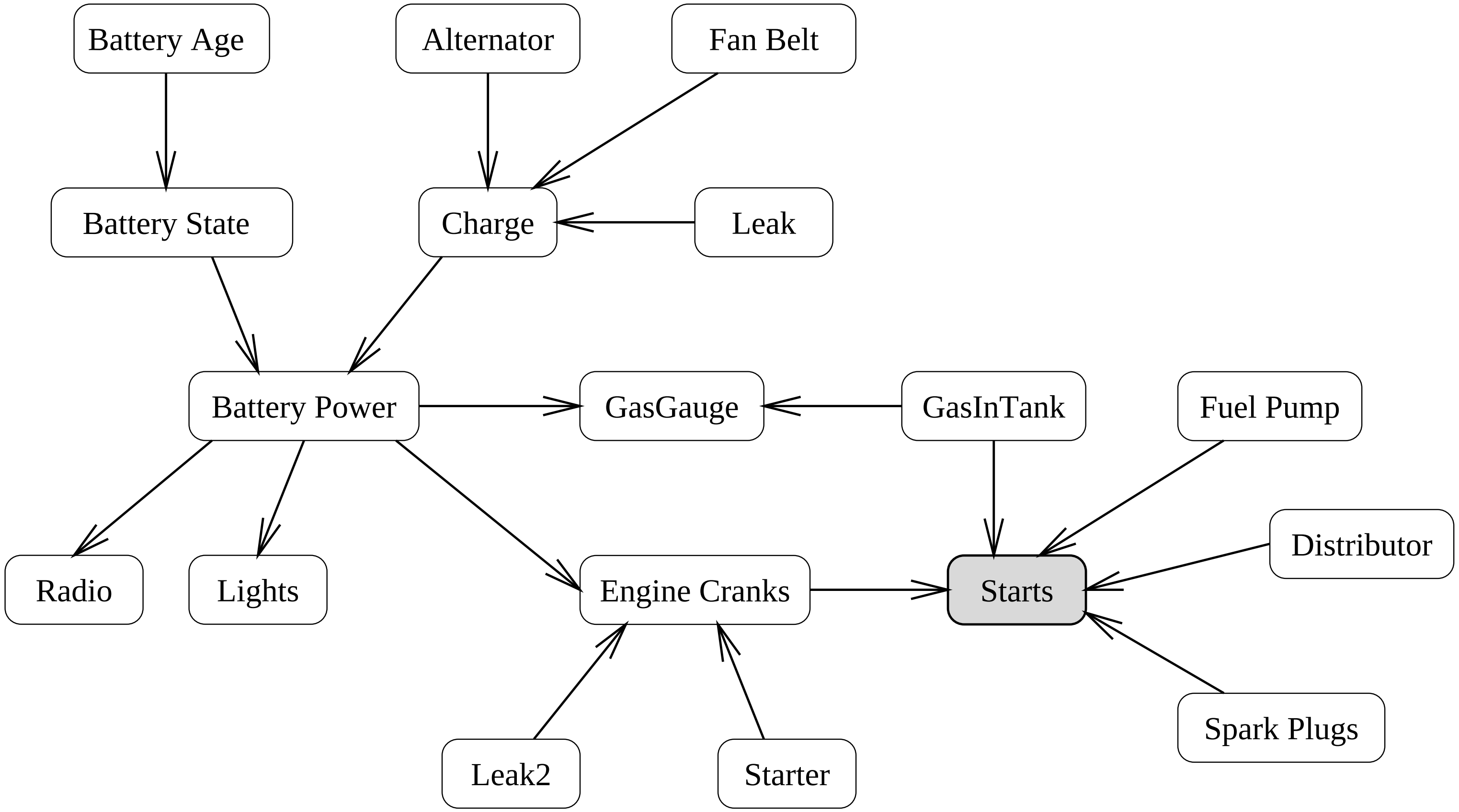


- **One node for each random variable**
- **Nodes with only outgoing arrows store marginal distributions**
  The Gas node stores $\mathbf{P}(\mathsf{Gas})$.
- **Nodes with incoming arrows store conditional probability tables**
  The Starts node stores $\mathbf{P}(\mathsf{Starts}|\mathsf{Gas})$.
- **Only direct dependencies are shown.** If two nodes are not directly connected, then they are conditionally independent on any set of nodes that disconnects them.
  Starts and $\mathsf{GasGauge}$ are conditionally independent on Gas.

Constructing Bayes Nets

The methodology for constructing a Bayesian network is simple:

Write down causal links



The Bayes Network Requires Many Fewer Parameters

Node	Number of Parameters	Node	Number of Parameters
Battery Age	2	Fuel Pump	2
Alternator	2	Radio	4
Fan Belt	2	Lights	4
Battery State	4	Engine Cranks	16
Charge	16	Starts	64
Leak	2	Distributor	2
Battery Power	8	Leak2	2
Gas Gauge	8	Starter	2
GasInTank	2	Spark Plugs	2

This is 144 parameters. By comparison, the full joint distribution has $2^{18} = 262,144$ parameters.

Computing Marginal Probabilities from the Bayes Net

$$\text{GasGauge} \longleftarrow \text{Gas} \longrightarrow \text{Starts}$$

- $\mathbf{P}(\mathsf{Gas})$

  This is stored directly in the gas node.

- $\mathbf{P}(\mathsf{GasGauge})$

$$\begin{aligned}
\mathbf{P}(GasGauge) &= \sum_{Gas,Starts} \mathbf{P}(Gas, GasGauge, Starts) \\
&= \sum_{Gas,Starts} \mathbf{P}(GasGauge|Gas)\mathbf{P}(Starts|Gas)\mathbf{P}(Gas) \\
&= \sum_{Gas} \mathbf{P}(GasGauge|Gas) \left[\sum_{Starts} \mathbf{P}(Starts|Gas)\right] \mathbf{P}(Gas) \\
&= \sum_{Gas} \mathbf{P}(GasGauge|Gas)\mathbf{P}(Gas)
\end{aligned}$$

Computing Probabilities for Diagnosis

- p_{c_i} The probability that component c_i is causing the device malfunction.

  We could compute this as the marginal probability $P(c_i = 1)$. However, this ignores the single-fault assumption. The marginal probabilities can sum > 1, which invalidates the value function calculation.

  I am instead using

  $$p_{c_i} = P(c_i = 1, c_0 = 0, \ldots, c_{i-1} = 0, c_{i+1=0}, \ldots, c_n = 0)$$

  where we include all components not-yet-observed.

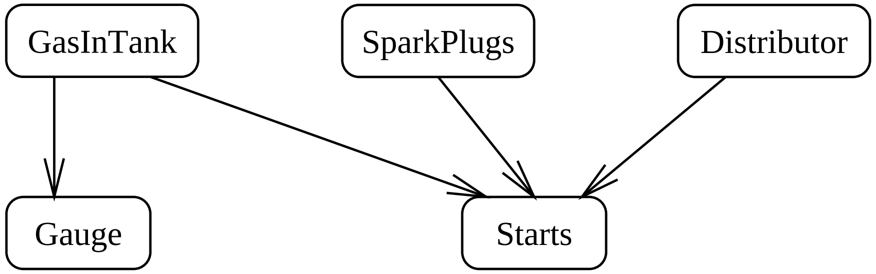
  This explicitly includes the single fault assumption.

Computing Probabilities for Diagnosis (2)

- $P(o_j = 0|E)$. The probability that observation o_j will give a result of 0 given all of the other evidence (observations and repairs) E that we have already obtained.

  We compute this by modifying the Bayes network to incorporate evidence E and then computing the marginal probability $P(o_j)$.

Mini-Car Network



P(GasInTank)

Gas	noGas
0.8	0.2

P(SparkPlugs)

plugsOK	plugsBad
0.6	0.4

P(Distributor)

distOK	distBad
0.7	0.3

P(Gauge|GasInTank)

	nonempty	empty
Gas	0.9	0.1
noGas	0.0	1.0

P(Starts|GasInTank, SparkPlugs, Distributor)

	Starts	WontStart
Gas, plugsOK, distOK	1.0	0.0
Gas, plugsOK, distBad	0.0	1.0
Gas, plugsBad, distOK	0.0	1.0
Gas, plugsBad, distBad	0.0	1.0
noGas, plugsOK, distOK	0.0	1.0
noGas, plugsOK, distBad	0.0	1.0
noGas, plugsBad, distOK	0.0	1.0
noGas, plugsBad, distBad	0.0	1.0

Joint Distribution

Starts	Gauge	GasInTank	SparkPlugs	Distributor	Probability
starts	nonempty	Gas	plugsOK	distOK	0.3024
starts	nonempty	Gas	plugsOK	distBad	0
starts	nonempty	Gas	plugsBad	distOK	0
starts	nonempty	Gas	plugsBad	distBad	0
starts	nonempty	noGas	plugsOK	distOK	0
starts	nonempty	noGas	plugsOK	distBad	0
starts	nonempty	noGas	plugsBad	distOK	0
starts	nonempty	noGas	plugsBad	distBad	0
starts	empty	Gas	plugsOK	distOK	0.0336
starts	empty	Gas	plugsOK	distBad	0
starts	empty	Gas	plugsBad	distOK	0
starts	empty	Gas	plugsBad	distBad	0
starts	empty	noGas	plugsOK	distOK	0
starts	empty	noGas	plugsOK	distBad	0
starts	empty	noGas	plugsBad	distOK	0
starts	empty	noGas	plugsBad	distBad	0

Joint Distribution (part 2)

WontStart	nonempty	Gas	plugsOK	distOK	0
WontStart	nonempty	Gas	plugsOK	distBad	0.1296
WontStart	nonempty	Gas	plugsBad	distOK	0.2016
WontStart	nonempty	Gas	plugsBad	distBad	0.0864
WontStart	nonempty	noGas	plugsOK	distOK	0
WontStart	nonempty	noGas	plugsOK	distBad	0
WontStart	nonempty	noGas	plugsBad	distOK	0
WontStart	nonempty	noGas	plugsBad	distBad	0
WontStart	empty	Gas	plugsOK	distOK	0
WontStart	empty	Gas	plugsOK	distBad	0.0144
WontStart	empty	Gas	plugsBad	distOK	0.0224
WontStart	empty	Gas	plugsBad	distBad	0.0096
WontStart	empty	noGas	plugsOK	distOK	0.084
WontStart	empty	noGas	plugsOK	distBad	0.036
WontStart	empty	noGas	plugsBad	distOK	0.056
WontStart	empty	noGas	plugsBad	distBad	0.024

Example Diagnosis

We begin with the knowledge that the car won't start, so we are in the bottom half of the joint distribution, which we can re-normalize:

Starts	Gauge	GasInTank	SparkPlugs	Distributor	Probability
WontStart	nonempty	Gas	plugsOK	distOK	0
WontStart	nonempty	Gas	plugsOK	distBad	0.1952
WontStart	nonempty	Gas	plugsBad	distOK	0.3036
WontStart	nonempty	Gas	plugsBad	distBad	0.1301
WontStart	nonempty	noGas	plugsOK	distOK	0
WontStart	nonempty	noGas	plugsOK	distBad	0
WontStart	nonempty	noGas	plugsBad	distOK	0
WontStart	nonempty	noGas	plugsBad	distBad	0
WontStart	empty	Gas	plugsOK	distOK	0
WontStart	empty	Gas	plugsOK	distBad	0.0217
WontStart	empty	Gas	plugsBad	distOK	0.0337
WontStart	empty	Gas	plugsBad	distBad	0.0145
WontStart	empty	noGas	plugsOK	distOK	0.1265
WontStart	empty	noGas	plugsOK	distBad	0.0542
WontStart	empty	noGas	plugsBad	distOK	0.0843
WontStart	empty	noGas	plugsBad	distBad	0.0361

Computing Optimal Repair-Only Policy (1)

```
Fault     P(fault & all others ok)
noGas     P(noGas, plugsOK, distOK) = 0.1265 = 0.0000 + 0.1265
plugsBad  P(plugsBad, Gas, distOK)  = 0.3373 = 0.3036 + 0.0337
distBad   P(distBad, Gas, plugsOK)  = 0.2169 = 0.1952 + 0.0217
```

Component	Normalized Prob	Obs Cost	Ratio	Repair Cost
GasInTank	0.1858	NA	0.0186	10
SparkPlugs	0.4956	20	0.0248	10
Distributor	0.3186	15	0.0212	20

Cost of observing problem-defining node = 2.

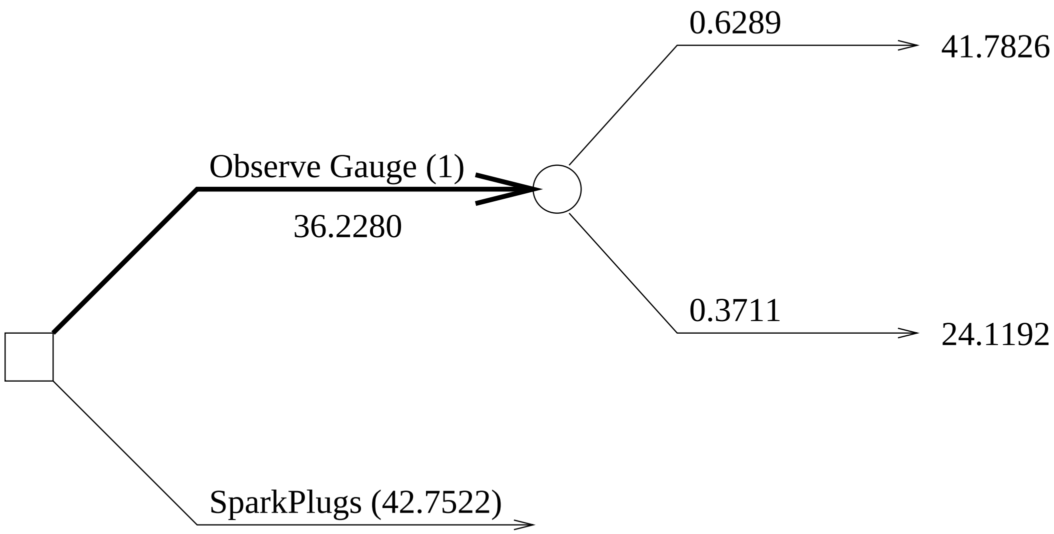
Expected value of this policy:

$$\begin{aligned} 20 + 0.4956(10+2) + \\ 0.5044 * 15 + 0.3186(20+2) + \\ 0.1858(10+2) &= 42.7522 \end{aligned}$$

Computing One-Step Observation Policy

```
Lookahead for observation of variable Gauge
 P(Gauge = 0) = 0.628916
  variable      P(fail)   Ratio
  Distributor   0.391304  0.026087
  SparkPlugs    0.608696  0.0304348
  GasInTank     0  0
 value = 41.7826
 P(Gauge = 1) = 0.371084
  variable      P(fail)   ratio
  Distributor   0.119205  0.00794702
  SparkPlugs    0.18543   0.00927152
  GasInTank     0.695364  0.0695364
 value = 24.1192
Cost of observing Gauge = 1.0
Expected value of observing Gauge = 36.228
```

Selecting the First Action to Perform



So we choose to Observe the Gauge rather than observing and repairing the SparkPlugs.

Narrowing the Joint Distribution

Suppose the true problem is the GasInTank. Then the Gauge will read empty. This narrows down the joint distribution further:

Starts	Gauge	GasInTank	SparkPlugs	Distributor	Probability
WontStart	empty	Gas	plugsOK	distOK	0
WontStart	empty	Gas	plugsOK	distBad	0.0585
WontStart	empty	Gas	plugsBad	distOK	0.0908
WontStart	empty	Gas	plugsBad	distBad	0.0391
WontStart	empty	noGas	plugsOK	distOK	0.3410
WontStart	empty	noGas	plugsOK	distBad	0.1461
WontStart	empty	noGas	plugsBad	distOK	0.2272
WontStart	empty	noGas	plugsBad	distBad	0.0973

Computing the Next Action to Take

```
  variable      P(fail)   ratio
  Distributor   0.119205  0.00794702
  SparkPlugs    0.18543   0.00927152
  GasInTank     0.695364  0.0695364
Expected value of repairing GasInTank = 24.1192
```

There are no other observations to make, so we go ahead and fill the tank. This solves the problem at a total cost of 13.

Programming Assignment

File	Classes
dx.c	dxInfo
	dxState
	dxProblem
bayes.c	node
	factor
	bayesNet
ptable.c	variableSet
	probabilityTable

Key Member Functions: class bayesNet

```
class bayesNet
  void observe(int var, int value);     // update to reflect observed value
  float query(int var, int value);      // compute a marginal probability
  float query(list<int> & queryVars, list<int> & queryValues);
```

Key Member Functions: class dxInfo **and** dxProblem

```
class dxInfo
  string name;
  int isObservable;
  int observationCost;  // incorporates repairCost for repair-only components
  int isRepairable;
  int repairCost;       // zero for repair-only components

class dxProblem
  int problemDefiningVariable;
  vector<dxInfo> variables;
  bayesNet * net;
  int solve();          // perform the diagnosis, returns actual cost
```

Key Member Functions: class dxState

```
class dxState
  int nvars;
  vector<int> state;  // operating status of each device. 0 = ok, 1 =
              // broken, -1 = unknown
  vector<float> probability;  // probability of each unknown
                              // variable being faulted (i.e. = 1)
  int bestAction;     // index of variable affected by best action
  // compute the value of the optimal repair-only policy from this state
  float value(dxProblem & problem, bayesNet & net);
  // compute the probability that var is faulty given that all other
  // not-yet-observed repairable components are not faulty.
  float singleFaultProbability(dxProblem & problem, bayesNet & net,
                               int var);
  // compute the value of a policy consisting of at most one
  // observation followed by a repair-only policy from this state
  float lookaheadValue(dxProblem & problem, int var);
```

dxProblem::solve()

```
int dxProblem::solve()
{
  dxState  currentState(variables.length());
  // assume that the device is currently broken
  currentState.state[problemDefiningVariable] = 1;
  net->observe(problemDefiningVariable, 1);

  int cost = 0;
  // assume single fault
  while (1) {
    float bestCost = currentState.value(*this, *(this->net));  // compute value and best action
    int bestAction = currentState.bestAction;
    if (1) cout << "Expected value of repairing " <<
      variables[bestAction].name << " = " << bestCost << endl;

    // CS533: Your code here

    dxInfo & culprit = variables[bestAction];
```

dxProblem::solve() part 2

```
if(culprit.isObservable) {
  // ask user for result of observation
  cout << "Observe " << culprit.name << ".  What is result (" <<
    net->nodes[bestAction]->valueNames[0] << " = 0, " <<
      net->nodes[bestAction]->valueNames[1] << " = 1)? ";
  int result;
  cin >> result;
  // update the Bayes net with this.  Note in the multiple fault
  // case, we should repair first if we can and then update the network.
  net->observe(bestAction, result);
  currentState.state[bestAction] = result;
  cost += culprit.observationCost;

  // if the component is repairable, repair it
  // did we find the fault?
  if (culprit.isRepairable && result == 1) {
    cout << "Repair " << culprit.name << endl;
    cost += culprit.repairCost;
    cout << "Device should be working now!" << endl;;
    return cost;
  }
}
```

dxProblem::solve() **part 3**

```
    else {
      cout << "Repair " << culprit.name << endl;
      cost += culprit.observationCost;
      // update the state to reflect repair
      currentState.state[bestAction] = 0;
      // update the network to reflect repair
      net->observe(bestAction, 0);
      // now observe problem defining variable
      cout << "Is the device working (ok = 0, broken = 1)? ";
      cost += variables[problemDefiningVariable].observationCost;
      int result;
      cin >> result;
      if(result == 0) return cost;
    }
  }
}
```

dxState::lookaheadValue

```
float dxState::lookaheadValue(dxProblem & problem, int var)
// what is the expected cost of a policy that observes variable var
// and then follows a repair-only policy thereafter?
{
  if (DEBUG) {
    cout << "Lookahead for observation of variable " << problem.variables[var].name <<
endl;
  }

  // CS533 students. Your code goes here.

  float answer = 0.0;
  return answer;
}
```

Modeling Issues

Our current network is designed specifically for diagnosing the problem of "engine doesn't start". It can be used to diagnose some other faults, for example, "engine doesn't crank". We could just change the problem-defining variable.

But it can't be used to diagnose "lights won't turn on" or "radio doesn't work". That is because the variable lights has the following meaning:

$\mathsf{lights} = \mathsf{ok}$	Light shines when switch is turned on.
$\mathsf{lights} = \mathsf{NoLight}$	Light does not shine when switch is turned on.

To diagnose faults, we would need to introduce an additional unobservable node: Bulb:

$\mathsf{Bulb} = \mathsf{ok}$	Bulb will operate correctly if electricity is present
$\mathsf{Bulb} = \mathsf{burnedOut}$	Bulb is burned out.