Define: $g^*(n)$ as minimum cost from root to n $\forall n \in$ Nodes

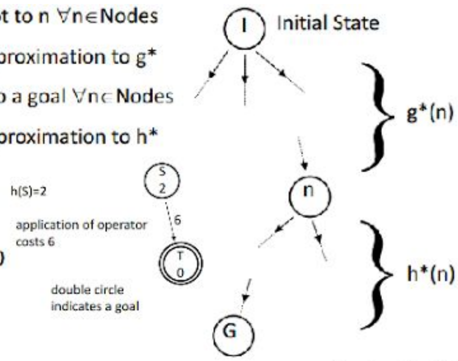
Define: $g(n)$ as an easily computable approximation to g^*

Define: $h^*(n)$ as minimum cost from n to a goal $\forall n \in$ Nodes

Define: $h(n)$ as an easily computable approximation to h^*

h is called a "heuristic function"

Define: $f(n)$ as $g(n) + h(n)$



`QF(a,b): Sort(Append(a,b), [ ])`

[ ] = h → greedy (aka best first)
g → uniform cost
f → A / A*

Suppose `QF(old, new): Append(new, old);` Depth First
Suppose `QF(old, new): Append(old, new);` Breadth First

Generic Search Function

```
SEARCH(Problem P, Queuing Function QF):
  local:    n     /* current node */
            q     /* nodes to explore */
  q ← singleton of Initial_State(P);
  Loop:
     if q = () return failure;
     n ← Pop(q);
     if n Solves P return n;
     q ← QF(q, Expand(n));
  end
```

Admissiblity

A search algorithm (together with its heuristic function if needed) is *admissible* iff for all search problems:

1. If there exists a goal, the search will not fail.
2. If there are multiple goals the search will find an optimal one (best, least expensive to execute).

Admissibility of A*

Some authors use "A" if not met

1) $\forall n\ \forall n' \in$ nodes, $\forall a \in$ actions with $a(n) \rightarrow n'$

$$h(n) \le cost(n,a,n') + h(n')$$

Triangle inequality, Montonicity, or Consistency
(for trees…)

Admissibility of A* (cont)

2) $\forall n \in$ nodes

$h(n) \le h^*(n)$

Informally: be optimistic
(or don't be pessimistic)
Why? Could you prove it?

Important General Principle:
Optimism Under Uncertainty

Admissibility does not depend on problem or tree!

Is "Uniform Cost" admissible?

We have:

A_1^* with some heuristic fcn h_1

A_2^* with another heuristic fcn h_2

A_1^* and A_2^* are admissible

Then we say

A_1^* is *more informed* than A_2^*

iff for all non-goal nodes n

$h_1(n) > h_2(n)$

"More Informed" implies "guaranteed not to search more"

Heuristic fcn h can be used to estimate utility
– Higher utility (smaller h) is better
– SORT still orders nodes best to worst

SEARCH	SPACE	TIME
Depth-First	Linear	Infinite
Breadth-First	Exponential	Exponential
Greedy / Best-First	Exponential	Infinite
Uniform Cost	Exponential	Exponential
A*	Exponential	Exponential

Beam Search w/ beam width k

```
SEARCH(Problem P, Queuing Function QF):
  local:    n     /* current node */
            q     /* nodes to explore */
  q ← singleton of Initial_State(P);
  Loop:
     if q = () return failure;
     n ← Pop(q);
     if n Solves P return n;
     q ← QF(q, Expand(n));
  end
QF(a,b): Sort(Append(a,b), h);
         Delete all but the best k;
```

Locally Optimizing Search: Hill Climbing

```
SEARCH(Problem P, Queuing Function QF):
  local:    n     /* current node */
            q     /* nodes to explore */
  q ← singleton of Initial_State(P);
  Loop:
     if q = () return n;
     if First(q) worse_than n return n;
     n ← Pop(q);
     q ← QF(q, Expand(n));
  end
QF(a,b): Sort(Append(a,b), U);
         Delete all but best;
```

A set S is convex iff

$\forall x,y \in S, \forall t \in [0,1] \Rightarrow (1-t)x + ty \in S$

$$H_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} \end{bmatrix} \text{ and } v^T H_f v > 0 \ \ \forall v$$

Search Properties

- Tentative (don't throw away information)
  Depth First, Breadth First, Uniform Cost, Best First/Greedy, A*
- Irrevocable (throw away information)
  Hill Climbing, Beam, Optimizing Beam
- Exhaustive (will visit all nodes or find goal)
  Breadth First, Uniform Cost, A*
- Admissible
  Uniform Cost, A* (if consistency & optimism)

Object constant → Individuals
Variable
Function expression → Properties
Predicate symbol → Relations

$\neg$	negation	"not"
$\wedge$	conjunction	"and"
$\vee$	disjunction	"or"
$\Rightarrow$	implication	"implies"
$\Leftrightarrow$	equivalence	"if and only if"

$A \Rightarrow B$ means precisely $\neg A \vee B$

$\Leftrightarrow$ is just $\Rightarrow$ both directions

{x = y}	YES
{x = y, z = F(y)}	YES
{x = y, z = F(y), x = A}	NO
{x = y, z = F(y), y = A}	YES
{x = y, y = F(z), z = G(x)}	NO

A *term* denotes an individual in the universe of discourse
variable
object constant
function expression

A *function expression* is an n-ary function symbol with n terms as arguments

An *atom* (also atomic sentence, atomic Well-Formed Formula) is an n-ary predicate symbol with n terms as arguments

A *literal* is an atom or a negated atom

Note quantifiers interact with negations!
$\neg \exists w\ P(w)$ is the same as $\forall x \neg P(w)$

Think DeMorgan…
$\neg (A \wedge B)$ is the same as $\neg A \vee \neg B$

Making explicit what is already true
Employing rules of inference

There is some amount that I like CS440 and I like all other classes less

$\exists z\ \exists w$ [Class(w) $\wedge$ Name(w,"CS440") $\wedge$ Likes(Me,w,z) $\wedge$
$\forall x\ \forall y$ {[Class(x) $\wedge$ Likes(Me,x,y) $\wedge$ Different(x,w)]
$\Rightarrow$ Greater(z,y)}]

Likes(a,b,c) means "a likes b by amount c"
Greater(a,b) means "a is larger than b"

Certain(Death) $\wedge$ Certain(Taxes) $\wedge$
$\forall x$ ([Different(x, Death) $\wedge$ Different(x, Taxes)] $\Rightarrow$
$\neg$Certain(x))

Nothing is certain except death and taxes

p, q, r are WFFs

$p \wedge q \equiv q \wedge p$
$p \vee q \equiv q \vee p$
$p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r$
$p \vee (q \vee r) \equiv (p \vee q) \vee r$
$p \Rightarrow q \equiv \neg q \Rightarrow \neg p$
$\neg(\neg p) \equiv p$
$p \Rightarrow q \equiv \neg p \vee q$
$p \Leftrightarrow q \equiv (p \Rightarrow q) \wedge (q \Rightarrow p)$
$p \vee p \equiv p$
$p \wedge p \equiv p$
$\neg(p \vee q) \equiv \neg p \wedge \neg q$
$\neg(p \wedge q) \equiv \neg p \vee \neg q$
$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

Modus Ponens

A well-known rule of inference
There are many others

$$\frac{\Theta \Rightarrow \Psi \qquad \Theta}{\Psi}$$

Are there entailed WFFs that Modus Ponens cannot derive?

No, it is incomplete

One inference rule, *resolution*, is almost complete on its own

A *unifier* (also *substitution*, *binding list**) is a set of pairings of variables with terms:

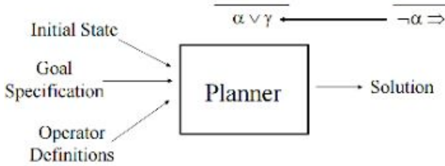
$\{v_1 = e_1, v_2 = e_2, v_3 = e_3, \ldots v_n = e_n\}$

such that

- each variable is paired at most once
- a variable's pairing term may not contain the variable directly or indirectly

{x = Socrates}

$\alpha \vee \beta \longrightarrow \neg\alpha \Rightarrow \beta$
$\neg\beta \vee \gamma \longrightarrow \beta \Rightarrow \gamma$
$\alpha \vee \gamma \longleftarrow \neg\alpha \Rightarrow \gamma$



MoveToBlock (x, z):

PC: Clr (x), Clr (z), On (x, y), Blk (x), Blk (z), Diff (x, z), Diff (y, z)

Effects: $\neg$On (x, y), $\neg$Clr (z), On (x, z), Clr (y)

MoveToTable (x, z):

PC: Clr (x), On (x, y), Blk (x), Tbl (z), Diff (y, z)

Effects: $\neg$On (x, y), On (x, z), Clr (y)

The MGU imposes the fewest constraints, specifying the *weakest* conditions for matching

MGU is unique assuming

order is not important

variable names are not important (alphabetic variants)

Applying the MGU to an expression yields a *most general unification instance*.

Variable substitutions are always interpreted with the unifier applied

FOPC – symbolic representations
- Quantification
- Inference
  - formal: possible worlds
  - algorithmic: inference rules like M.P.
- Unification

STRIPS operators
- Essentially FOPC but slightly reduced expressiveness
- Sometimes requires extra operators

Planning with STRIPS operators
- World states
- Abstract domain-independent procedures

Planning
- First-order representations
- Build in world dynamics
- Plan = sequence of actions
- Predict precise intermediate world states
- Sensing (and execution) play a small part
- Strong single agent assumption

Reinforcement learning
- Propositional (zeroth-order) representations
- Learn world dynamics
- Plan = action policy
- Tolerate arbitrarily bad behavior in the world
- Sensing during execution is crucial
- Weaker single agent assumption

$\pi(s_{14}) = a_3$

$T(s_{14}, a_3) = s_{81}$

We cannot anticipate the world's true transition function

Thus, our model of it must in part be learned

We model uncertainty as a distribution over next states

Our Transition Function
- $T: S \times A \times S \to [0,1]$
- where each $T(S,A,\cdot)$ is a distribution

This approximates (in some sense [we hope]) the world's own transition function

Markov Decision Process (MDP)

- A finite set of states S = {s}
- A finite set of actions A = {a}
- Initial distribution over S (more general than R&N)
- Probabilistic transition model
- Probabilistic rewards (more general than R&N)

The world *may* be deterministic but we *choose* to model it as stochastic

Transition function
- $T: S \times A \times S \to [0,1]$
- Is T a probability distribution?
- $T(s,a,\cdot)$ denotes a probability distribution over next states
- $P(\cdot \mid s, a)$ as conditional probability (in R&N)

Reward function
- $Rw: S \times \Re \to [0,1]$
- Each $Rw(s,\cdot)$ denotes a probability distribution over rewards

What do we care about with rewards?

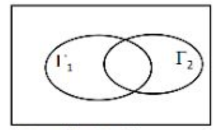
$R: S \to \Re$

R maps states to expected rewards

$$E[X] = \sum (x_i * \Pr(X=x_i))$$

Γ_1 $\exists x\, Bird(x)$

Γ_2 $\exists x\, Flies(x)$



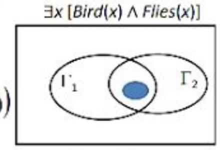
Possible Worlds

$$\pi^*(s) = \arg\max_a \sum_{s'} T(s,a,s') \cdot U^{\pi^*}(s')$$

$$U_{new}(s) \leftarrow U_{old}(s) + \alpha \cdot error$$

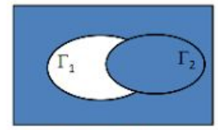
$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha\left(R(s) + \gamma \cdot U^{\pi}(s') - U^{\pi}(s)\right)$$

$\exists x\, [Bird(x) \wedge Flies(x)]$



Possible Worlds

$\exists x\, [Bird(x) \Rightarrow Flies(x)]$



Possible Worlds

P is a unary predicate; F is a unary function; x is a variable; Sam is a constant

P(x)	Atom, Literal, WFF
P(Sam)	Atom, Literal, WFF
P(F(Sam))	Atom, Literal, WFF
P(x) ∧ P(Sam)	WFF, conjunction of two atoms, conjunction of two literals
¬ P(x) ∧ P(x)	WFF, conjunction of a literal and an atom, conjunction of two literals
¬ P(x)	Literal, WFF, the negation of an atom
¬ F(Sam)	ill-formed, not a WFF at all
P(¬ Sam)	ill-formed, not a WFF at all
P(x) ∧ F(Sam)	ill-formed, not a WFF at all

Utility of a state s given a policy π with discount γ

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]$$

U^{π} has a simple form: $U^{\pi}: S \to \Re$ (state utility is policy dependent)

for any $\varepsilon > 0$

$$\lim_{n\to\infty} Pr(|\overline{c_n} - E[c]| > \varepsilon) = 0$$

$$\pi^*(s) = \arg\max_a Q^*(a,s)$$

Let π^* be the optimal policy

Given U^{π^*} and T we can write/implement the optimal policy concisely & efficiently

T(i,j,k) can be estimated as the ratio:

\# times action j takes us from state i to state k divided by # times action i is tried in state i

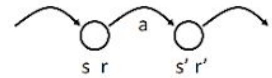
$$U^{\pi}(s) \leftarrow (1-\alpha)U^{\pi}(s) + \alpha\left(R(s) + \gamma \cdot U^{\pi}(s')\right)$$

False - holds in no possible worlds (i.e., self contradictory)

True - holds in all possible worlds (i.e., tautology)

Satisfiable: - holds in some possible worlds but not others

Entailed - holds in all remaining possible worlds after intersecting the axioms' possible worlds



We can learn the policy *directly*

Instead of T and U, learn Q

Q is the utility of performing an action in a state

Q-learning is *model free* Reinforcement Learning

Q-learning is *off policy*
- optimal greedy policy can be learned
- even if it is not followed during learning

General distinctions in learning
- Full or joint model vs. Conditional model
- Generative model vs. Discriminative model

Q function: $Q: A \times S \to \Re$

Q(a,s) - the expected utility of performing action a in state s

Q represents local and non-local information (much as U does)

But T (our model of the world's transition function) is not needed

The (converged) Q-table is independent of ε

Sometimes we may prefer a state-action choice that includes ε in some way

In ε–greedy, ε is reflected within SARSA's Q values

SARSA learns the best policy given our ε–greedy systematic departures from optimal

Convergence of both value iteration and policy iteration is based on *convexity* of the utility space

We will see gradient descent and stochastic gradient descent again with perceptrons & neural networks

But here in RL, both are neglecting information…

```
public void updatePolicy(double reward, int action, int oldState, int newState):
        qValue[oldState][action] = (1-learningRate)*qValue[oldState][action]+
        learningRate*(reward+discountFactor*qValue[newState][policy[newState]]);
        double max = 0,current = 0;
        for(int i=0; i<numOfActions; i++):
                current = qValue[oldState][i];
                if(current >= max):
                        max = current;
                        policy[oldState] = i;
```

ε-Greedy Exploration

$$Q(a,s) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(a',s')$$

- Instead of greedy: $\arg\max_a \sum_{s'} T(s,a,s') \cdot U^{\pi}(s')$
- Be greedy with probability (1-ε)
- But with probability ε choose a random action

$$Q(a,s) \leftarrow (1-\alpha)\cdot Q(a,s) + \alpha \cdot \left(R(s) + \gamma \max_{a'} Q(a',s')\right)$$

$$Q(a,s) \leftarrow Q(a,s) + \alpha \cdot \left(r_s + \gamma \max_{a'} Q(a',s') - Q(a,s)\right)$$

So:
- Choose an ε 0 < ε << 1
- At every decision get a random number x: [0,1]
- If x < ε, choose randomly else be greedy with current T and U

Minimal regret grows as Sqrt (N)
- In fact

  0.264*Sqrt(N) < Regret < 0.376*Sqrt(N)

Assume policy π chooses action a in s

Relate $U^{\pi}(s)$ and $U^{\pi}(s')$

$U^{\pi}(s) = \gamma U^{\pi}(s') + R(s)$

$U^{\pi}(s) - \gamma U^{\pi}(s') = R(s)$

If not equal then there is an error

So *error* = $R(s) + \gamma U^{\pi}(s') - U^{\pi}(s)$

RL, including Q-learning, employ a number of user-specified parameters

As in much of machine learning, understanding their influence and knowing how much to experiment is a key to success

γ - the discount rate
- domain parameter reflecting the relative importance of nearer rewards over more distant ones for the user
- it should not be used to influence the Q-learner

α - the learning rate
- reflects the relative confidence in the old and new information
- typically does not preclude convergence but can have a significant effect on the speed of convergence

ε - exploration probability
- employed to insure an acceptable amount of random exploration
- too low: slow convergence into a small good region around π^*
- too high: quickly converges into a large poor region around π^*

Policy for TD Value Iteration?
- T, U
- T: $\Re^{|S|\cdot|A|\cdot|S|}$
- U: $\Re^{|S|}$
- $|S|^2\cdot|A| + |S|$ real numbers: $O(|S|^2\cdot|A|)$

Policy for Q?
- Q
- Q: $\Re^{|S|\cdot|A|}$
- $|S|\cdot|A|$ real numbers : $O(|S|\cdot|A|)$

Q, an off-policy learner:

$$Q(a,s) \leftarrow Q(a,s) + \alpha \cdot \left(r_s + \gamma \max_{a'} Q(a',s') - Q(a,s)\right)$$

SARSA, an on-policy learner:

$$Q(a,s) \leftarrow Q(a,s) + \alpha \cdot \left(r_s + \gamma \cdot Q(a',s') - Q(a,s)\right)$$

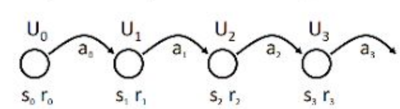
Reinforcement Learning vs. Classical Planning

More robust (this is a big one…)
- Fewer & weaker *a priori* assumptions (esp. actions)
- Empirical model
- Fit (via parameter adjustment) to the *observed* world

A *LOT* of training is required
- Must see many and varied state transitions
- Compared to no training for classical planning

Scaling difficulties
- Propositional expressiveness (vs. first-order for classical planning
- Choosing / defining state distinctions can be challenging
- Space & Time complexity is polynomial (but in what? The generally unknowable ε-return mixing time & others…)
- Generalizing to other similar problems can be difficult
- Some domains require the very same problem solved repeatedly; others do not
- Recognizing convergence?? sufficiency?

Adaptive Dynamic Programming



Imagine successive value iteration…on s_0 … on s_1 … on s_2:

Perform a_2, update U_2 with r_2 and U_3

U_2 is now updated to a better value

We used the old U_2 to update U_1, shouldn't it be changed as well?

What about U_0?

Fully appreciate each r