Review

IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!

... WE'RE NOT INVITING YOU HOME NEXT YEAR.
1. Formally describe problems
2. Search-uninformed
3. Search-informed
4. 2 player games
5. Constraint satisfaction problems
6. Logic-Propositional
7. Logic-First order
8. Planning
9. Ontology (a teeny tiny bit?)
# Rational agent

<table>
<thead>
<tr>
<th>Agent type</th>
<th>Performance</th>
<th>Environment</th>
<th>Actuators</th>
<th>Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vacuum</td>
<td>time to clean</td>
<td>A, B, dirt</td>
<td>suck, move</td>
<td>dust sensor</td>
</tr>
<tr>
<td>Student</td>
<td>GPA, honors</td>
<td>campus, dorm</td>
<td>do HW, take test</td>
<td>eye, ear, hand</td>
</tr>
<tr>
<td>Particles</td>
<td>time alive</td>
<td>boarder, red balls</td>
<td>move mouse</td>
<td>screen-shot</td>
</tr>
</tbody>
</table>
Environment classification

Environments can be further classified on the following characteristics:

1. Fully vs. partially observable
2. Single vs. multi-agent
3. Deterministic vs. stochastic
4. Episodic vs. sequential
5. Static vs. dynamic
6. Discrete vs. continuous
7. Known vs. unknown
States can be generalized into three categories:

1. Atomic, state = identifier only (no relations)
2. Factored, state = fixed number of properties
3. Structured, state = define relations between states in a more open way (FO logic)

Occam's razor = if two results are identical, use the simpler approach
Uninformed search

Uninformed search knows what the goal is but not the layout of the environment

Goal: get out!
Uninformed search

To search, we will build a tree with the root as the initial state

```plaintext
function tree-search(root-node)
    fringe ← successors(root-node)
    while ( notempty(fringe) )
        {node ← remove-first(fringe)
         state ← state(node)
         if goal-test(state) return solution(node)
         fringe ← insert-all(successors(node), fringe) } 
    return failure
end tree-search
```

How to pick from fringe?
Uninformed search

Breadth first search (BFS) tries to explore all directions equally, poking only a bit further than last time...
Uninformed search

Depth first search (DFS) explores until it reaches a dead-end, then turns around and turns off at the first unexplored part.
Uninformed search

Iterative deepening DFS performs a BFS-like search but without the memory limitations (however it repeats some nodes, but not too many)
### Summary of algorithms

**Fig. 3.21, p. 91**

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening DLS</th>
<th>Bidirectional (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete?</td>
<td>Yes[a]</td>
<td>Yes[a,b]</td>
<td>No</td>
<td>No</td>
<td>Yes[a]</td>
<td>Yes[a,d]</td>
</tr>
<tr>
<td>Time</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+C*/\varepsilon})$</td>
<td>$O(b^m)$</td>
<td>$O(b^l)$</td>
<td>$O(b^d)$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Space</td>
<td>$O(b^d)$</td>
<td>$O(b^{1+C*/\varepsilon})$</td>
<td>$O(bm)$</td>
<td>$O(bl)$</td>
<td>$O(bd)$</td>
<td>$O(b^{d/2})$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes[c]</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes[c]</td>
<td>Yes[c,d]</td>
</tr>
</tbody>
</table>

There are a number of footnotes, caveats, and assumptions. See Fig. 3.21, p. 91.

- [a] complete if $b$ is finite
- [b] complete if step costs $\geq \varepsilon > 0$
- [c] optimal if step costs are all identical
  (also if path cost non-decreasing function of depth only)
- [d] if both directions use breadth-first search
  (also if both directions use uniform-cost search with step costs $\geq \varepsilon > 0$)

Generally the preferred uninformed search strategy
Informed search

Informed search knows a basic layout of the environment: it some some *estimate* of the to-goal distance, we call this the heuristic

The heuristic is a value, not a policy!

**A* search is:**

\[ f(\text{node}) = g(\text{node}) + h(\text{node}) \]

- *distance gone (traveled) so far*
- *total cost estimate*
- *heuristic (estimate to-goal distance)*
A*

\[ h(\text{node}) = 0 \]
(bad heuristic, no goal guidance)

\[ h(\text{node}) = \text{straight line distance} \]
(good heuristic)
Heuristics

However, for A* to be optimal the heuristic $h$(node) needs to be...

For trees: **admissible** which means:

\[ h(\text{node}) \leq \text{optimal path from } h \text{ to goal} \]

(i.e. $h$(node) is an underestimate of cost)

For graphs: **consistent** which means:

\[ h(\text{node}) \leq \text{cost}(\text{node to child}) + h(\text{child}) \]

(i.e. triangle inequality holds true)

(i.e. along any path, $f$-cost increases)
Hill climbing

Can get stuck in:
- Local maximum
- Plateau/shoulder

Local maximum will have a range of attraction around it

Can get an infinite loop in a plateau if not careful (step count)
Hill climbing

Modifications to overcome local max/min:
1. Repeated restart
2. Beam search (have multiple people climb, invest more resources in the hopeful ones)
3. Simulated annealing (go downhill sometimes)

Typically hill climbing focus more on the heuristic than A*, which balances both g and h
Genetic algorithms

Selection/survival:
Typically children have a probabilistic survival rate (randomness ensures genetic diversity)

Crossover:
Split the parent's information into two parts, then take part 1 from parent A and 2 from B

Mutation:
Change a random part to a random value
We can use policy iteration to search when there is uncertainty.

Initialize the values

Then we find best action for each square, we use this equation:

$$\arg\max_{a \in \text{actions}} \sum_{s' \text{ from } s} P(s, a, s') \cdot (R_a(s, s') + \gamma V(s'))$$

(called Bellman equation)
Minimax

\[ \text{max}( \text{min}(1,3), 2, \text{min}(0, 4) ) = 2, \text{ should pick action F} \]

Order:
1\textsuperscript{st}. R (can swap B B and R)
2\textsuperscript{nd}. B
3\textsuperscript{rd}. P
max( min(1,3), 2, min(0, ??) ) = 2, should pick action F

Order:
1st. R (can swap B and R)
2nd. B
3rd. P

Do not consider Alpha-beta pruning
Mid-state evaluation

By using **mid-state evaluations** (not terminal) the “best” action can be found quickly

These mid-state evaluations need to be:

1. Based on current state only
2. Fast (and not just a recursive search)
3. Accurate (represents correct win/loss rate)

The quality of your final solution is highly correlated to the quality of your evaluation
Game theory

Nash = point where no single person wants to change

Pareto = A point which has no other point better for all players (i.e. worse for at least one player)
Constraint satisfaction problems (CPS) need to find a non-conflicting value in the domain of each variable.
CSP backtracking

However, this is still hope for searching (called backtracking search (it backups up at conflict))

We will improve it by...
1. The order we pick variables
2. The order we pick values for variables
3. Mix search with inference
4. Smarter backtracking
Logic: entailment

We say $\alpha$ entails $\beta$ ($\alpha \models \beta$) if and only if every model with $\alpha$ true, $\beta$ is also true.

Another definition (mathy): $\alpha \models \beta$ if and only if $M(\alpha)$ subset $M(\beta)$

This means there are fewer models true with proposition $\alpha$ than $\beta$. 
Logic: entailment

However, if we let $\beta = \text{mine at (2,3)}$, we get:

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
\end{array}
\] \quad \begin{array}{cccc}
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
\end{array}
\]

$M(\text{knowledge base (KB)})$ is (again):

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
\end{array}
\] \quad \begin{array}{cccc}
1 & 1 & 1 & 1 \\
2 & 2 & 2 & 2 \\
\end{array}
\]

This is not entailment, as this is not in $M(\beta)$, thus $\text{KB} \not\models \beta$ (in other words “from the KB, you cannot conclude (2,3) is a mine”)}
We have these rules for inference:

1. Any logically equivalent statements (e.g. \( \alpha \leftrightarrow \beta \) )

2. Modus ponens: \( \alpha \Rightarrow \beta, \alpha \rightarrow \beta \Rightarrow \alpha \) (top is two sentences)

3. And-elimination: \( \alpha \land \beta \rightarrow \alpha \)

We repeatedly apply these rules until we reach the statement we desire.
First sentence is the only one with variables, there are 9 options (only 6 if $x \neq y$)

One unification is $\{x/\text{Sue}, y/\text{Devin}\}$

We cannot say $\{x/\text{Devin}, y/\text{Alex}\}$, as this is creates a contradiction
Forward chaining

Sandwich(Bread)

Grilled(M1)

Grilled(Bread)

Sandwich(Bread)
Backward chaining

Grilled(Bread)

2.

Sandwich(Bread)  OnGrill(x,Bread)

1.

Meat(M1)  Make(Bread,M1,Bread)

4.

{z/M1}

5.

{x/any x}

6.

{}

Begin DFS (left branch first)
First-order logic resolution

To do first-order logic resolution we again need to get all the sentences to CNF.

This requires a few more steps for FOL:
1. Use logical equivalence to remove implies
2. Move logical negation next to relations
3. Standardize variables
4. Generalize existential quantifiers
5. Drop universal quantifiers
6. Distribute ORs over ANDs
Forward search

GoTo(Door) can ignore

AddMilk()

At(Door)  ...  At(Aisle1)  Cart(Milk)
At(Aisle1) ...  At(Door)
At(Aisle2) ...  At(Aisle1)
At(Aisle3) ...  ...

GoTo(Checkout)
At(Checkout) ...
Heuristics for planning

In “search” we had no generalize-able heuristics as each problem could be different.

Heuristics in planning are also the same, we want an admissible one found from relaxing the problem and solving that optimally.

There are two ways to always do this:
1. Add more actions
2. Reduce number of states
Graph Plan

Ask:

\[ D^\wedge S^\wedge M \]

Find first no mutex...
Partition

\[ \text{Partition}(x, y) \iff \text{Disjoint}(x) \land \text{Exhaustive Decomposition}(x, y) \]

Every point in \( S \) is either in \( \{A_1, A_2, A_3, A_4\} \) but never in more than one