1 General Formula

2 Agents

Have:
- Abilities
- Goals/Preferences
- Prior Knowledge
- Stimuli
- Past Experiences
- Actions

Belief State: Internal belief about the world **Knowledge**: Information used to solve tasks **Representation**: Data structure to encode knowledge **Knowledge Base**: Representation of all knowledge possessed **Model**: How KB relates to world

2.1 Dimensions of Complexity

1. Modularity
   - Flat: No modularity in computation
   - Modular: Each component is separateand siloed
   - Hierarchical: Modular components are broken into a hierarchical manner of subproblems
2. Planning Horizon:
   - Non-Planning: World doesn't change as a result (Ex: Protein Folding)
   - Finite: Reason ahead fixed number of steps
   - Indefinite: Reason ahead finite number (but undetermined) of steps
   - Infinite: Reason forever (focus on process)
3. Representation:
   - States: State describes how world exists
   - Features: An attribute of the world
   - Individuals and relations: How features relate to one another (Eg: child.failing() relates to child.grade)
4. Computational limits:
   - Perfect rationality: Agent always picks best action (Eg: Tic-Tac-Toe)
   - Bounded rationality: Agent picks best action given limited computation (Eg: Chess)
5. Learning:
   - Given knowledge (Eg Road laws)
   - Learned knowledge (Eg How car steers in rain)
6. Uncertainty:
   - Fully observable: Agent knows full state of world from observations (Eg: Chess)
   - Partially observable: Many states can lead to same representation (Eg: Battleship)
   - Deterministic: Action has predictable effect
   - Stochastic: Uncertainty exists over effect of action to state
7. Preference:
   - Achievement Goal: Goal to reach (binary)
   - Maintenance Goal: State to maintain
   - Complex Preferences: Complex tradeoffs between criteria and ordinality (can't please everyone)
8. Num Agents:
   - Single agent
   - Adversarial
   - Multiagent
9. Interactivity:
   - Offline: Compute its set of actions before agent has to act, so no computations required
   - Online: Computation is done between observing and acting

3 Search

Search Problem:
- Set of states
- Initial state
- Goal function
- Successor function
- (optional) cost

Frontier: Ends of paths from start node that have been explored

3.1 Graph Search Algorithm

```
frontier is just start node
while frontier isn't empty
    select and remove path $\langle n_0, ..., n_k \rangle$ from frontier
    if goal($n_k$) then
        return $\langle n_0, ..., n_k \rangle$
    for each neighbor $n$ of $n_k$ do
        add $\langle n_0, ..., n_k, n \rangle$
```

4 Constraints

4.1 Constraint Satisfaction Problems

- A set of variables
- Domain for each variable
- Two kinds of problems:
  - Satisfiability problems: Assignment satisfying hard constraints
  - Optimizatoin: Find assignment optimizing evaluation function (soft constraints)
- Solution is assignment to variables satisfying all constraints
- Solution is model of constraints

4.1.1 CSPs as graphs

Search spaces can be very large, path isn't important, only goal, and no set starting nodes make this bad idea **Complete Assignment**: Nodes: Assignment of value to all variables Neighbors: Change one variable value **Partial Assignment**: Nodes: Assignment to first $k - 1$ variables Neighbors: Assignment to k^{th} variable

4.2 Constraints

- Can be **N-ary** (over sets of N variables) (Ex: A + B = C involves is 3-ary for 3 vars)

4.2.1 Generate and Test

Exhaust every possible assignment of vars and test validity

4.2.2 Backtracking

Order all variables and evaluate constraints in order as soon as they are fixed. (Ex: $A = 1 \wedge B = 1$ is inconsistent with $A \neq B$ so go to last assigned variable and change its value)

4.2.3 Consistency

Represent constraints as network to determine how all variables are related. **Domain Constraint**: Unary constraint on values in domain written $\langle X, c(X) \rangle$ (Eg: $B, B \neq 3$) **Domain Consistent**: A node is domain consistent if no domain value violates any domain constraint, and a network is domain consistent if all nodes are domain consistent. **Arc**: Arc $\langle X, c(X, Y) \rangle$ is a constraint on X **Arc Consistent**: Arc $\langle X, c(X, Y) \rangle$ is arc consistent if for every valid x there is a valid y such that constraint is satisfied. **Path Consistent**: A set of variables is path consistent if all arcs and domains are consistent.

4.2.4 AC-3

Make Consistency network arc consistent
- To-Do Arcs Queue contains all inconsistent arcs
- Make all domains domain consistent
- Put all arcs in TDA
- Repeat until TDA is empty:
  - Select and remove an arc from TDA
  - Remove all values of domain of X that don't have value in domain of Y that satisfy constraint
  - If any were removed, add all arcs to TDA

Termination:
- If every domain is empty, no solution
- If every domain has a single value, solution
- If some domain has more than one value, split in two run AC-3 recursively on two halves
- Guaranteed to terminate
- Takes $O(cd^3)$ time, with n variables, c binary constraints, and max domain size is d because each arc $\langle X_k, X_i \rangle$ can be added to queue at most d times because we can delete at most d values from X_i. Checking consistency takes $O(d^2)$ time.

4.2.5 Variable Elimination

- Eliminate variables one-by-one passing constraints to neighbours.
- When single variable remains, if no values exist then network was inconsistent.
- Variables are eliminated according to elimination ordering.

Pseudocode:
- If only one variable, return intersection of unary constraints referencing it
- Select variable X
  - Join constraints affecting X, forming constraint R
  - Project R onto its variables other than X, calling this R2
  - Place new constraint between all variables that were connected to X
  - Remove X
  - Recursively solve simplified problem
  - Return R joined with recursive solution

4.2.6 Local Search

- Maintain assignment of value to each variable
- At each step, select neighbor of current assignment
- Stop when satisfying assignment found or return best assignment found
- Heuristic function to be minimized: Number of conflicts
- Goal is an assignment with zero conflicts

4.2.7 Greedy Descent

Select some variable (through some method) and then select the value that minimizes the number of conflicts. THe problem is that we could be stuck in a local minimum, without reaching the proper global minimum.

4.2.8 Stochastic Local Search

Do Greedy descent, but allow some steps to be random, and the potential to restart randomly, to minimize potential for being stuck in local minimum.

Problem: in high dimensions often consist of long, nearly flat "canyons" so it's hard to optimize using local search.

4.2.9 Simulated Annealing

Pick variable at random, if it improves, adopt it. If it doesn't improve, then accept it with a probability through the temperature parameter, which can get slowly reduced.

4.2.10 Tabu Lists

Variant of Greedy Satisfiability, where to prevent cycling and getting stuck in local optimum, we maintain a "tabu list" of the k last assignments, and don't allow assignment that has already existed.

4.2.11 Parallel Search

- Total assignment is called individual
- Maintain population of k individuals
- At each stage, update each individual in population
- Whenever individual is a solution, it can be reported
- Similar to k restarts, but uses k times minimum number of steps

4.2.12 Beam Search

- Like parallel search, with k individuals, but choose the k best out of all the neighbors. The value of k can limit space and induce parallelism

4.2.13 Stochastic Beam Search

- Like beam search, but probabilistically choose k individualls at next generation. Probability of selecting neighbor is proportional to heuristic: $e^{-\frac{h(n)}{T}}$. This maintains diversity among the individuals, because it's similar to simulated annealing.

4.2.14 Genetic Algorithms

- Like stochastic beam search, but pairs of individuals are combined to create offspring.
- For each generation, randomly choose pairs where fittest individuals are more likely selected
- For each pair, do cross-over (form two offspring as mutants of parents)
- Mutate some values
- Stop when solution is found

4.3 Comparing Algorithms

Since some algorithms are super fast some of the time and super slow other times, and others are mediocre all of the time, how do you compare? You use runtime distribution plots to see the proportion of runs that are solved within a specific runtime.

5 Inference and planning

5.1 Problem Solving

Procedural solving: Devise algorithm, program, execute **Declarative solving**: Identify required knowledge, encode knowledge in representation, use logical consequences to solve.