Simulated annealing

- Resembles the annealing process used to cool metals slowly to reach an ordered (low-energy) state

- Basic idea:
  - Allow “bad” moves occasionally, depending on “temperature”
  - High temperature => more bad moves allowed, shake the system out of its local minimum
  - Gradually reduce temperature according to some schedule
  - Sounds pretty flaky, doesn’t it?
function SIMULATED-ANNEALING(problem, schedule) returns a state

current ← problem.initial-state

for t = 1 to ∞ do
    T ← schedule(t)
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← next.value − current.value
    if ΔE > 0 then current ← next
    else current ← next only with probability e^{ΔE/T}
Simulated Annealing

- Theoretical guarantee:
  - Stationary distribution (Boltzmann): \( P(x) \propto e^{E(x)/T} \)
  - If \( T \) decreased slowly enough, will converge to optimal state!

- Proof sketch
  - Consider two adjacent states \( x, y \) with \( E(y) > E(x) \) [high is good]
  - Assume \( x \rightarrow y \) and \( y \rightarrow x \) and outdegrees \( D(x) = D(y) = D \)
  - Let \( P(x), P(y) \) be the equilibrium occupancy probabilities at \( T \)
  - Let \( P(x \rightarrow y) \) be the probability that state \( x \) transitions to state \( y \)
Occupation probability as a function of $T$
Simulated Annealing

- Is this convergence an interesting guarantee?

- Sounds like magic, but reality is reality:
  - The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row
  - “Slowly enough” may mean exponentially slowly
  - Random restart hillclimbing also converges to optimal state...

- Simulated annealing and its relatives are a key workhorse in VLSI layout and other optimal configuration problems
Local beam search

- **Basic idea:**
  - \( K \) copies of a local search algorithm, initialized randomly
  - For each iteration
    - Generate ALL successors from \( K \) current states
    - Choose best \( K \) of these to be the new current states
  - Or, \( K \) chosen randomly with a bias towards good ones
Beam search example \((K=4)\)
Local beam search

- Why is this different from $K$ local searches in parallel?
  - The searches *communicate*! “Come over here, the grass is greener!”
- What other well-known algorithm does this remind you of?
  - Evolution!
Genetic algorithms use a natural selection metaphor

- Resample $K$ individuals at each step (selection) weighted by fitness function
- Combine by pairwise crossover operators, plus mutation to give variety
Example: N-Queens

- Does crossover make sense here?
- What would mutation be?
- What would a good fitness function be?
Local search in continuous spaces
Example: Siting airports in Romania

Place 3 airports to minimize the sum of squared distances from each city to its nearest airport

Airport locations
\[ x = (x_1, y_1), (x_2, y_2), (x_3, y_3) \]

City locations \((x_c, y_c)\)

\(C_a = \text{cities closest to airport } a\)

Objective: minimize
\[ f(x) = \sum_a \sum_{c \in C_a} (x_a - x_c)^2 + (y_a - y_c)^2 \]
Handling a continuous state/action space

1. Discretize it!
   - Define a grid with increment $\delta$, use any of the discrete algorithms

2. Choose random perturbations to the state
   a. First-choice hill-climbing: keep trying until something improves the state
   b. Simulated annealing

3. Compute gradient of $f(x)$ analytically
Finding extrema in continuous space

- Gradient vector $\nabla f(x) = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \ldots)^T$
- For the airports, $f(x) = \sum_a \sum_{c \in C_a} (x_a - x_c)^2 + (y_a - y_c)^2$
- $\frac{\partial f}{\partial x_1} = \sum_{c \in C_1} 2(x_1 - x_c)$
- At an extremum, $\nabla f(x) = 0$
- Can sometimes solve in closed form: $x_1 = (\sum_{c \in C_1} x_c)/|C_1|$
- Is this a local or global minimum of $f$?
- Gradient descent: $x \leftarrow x - \alpha \nabla f(x)$
  - Huge range of algorithms for finding extrema using gradients
Many configuration and optimization problems can be formulated as local search

General families of algorithms:
- Hill-climbing, continuous optimization
- Simulated annealing (and other stochastic methods)
- Local beam search: multiple interaction searches
- Genetic algorithms: break and recombine states

Many machine learning algorithms are local searches
CS 188: Artificial Intelligence

Propositional Logic I

Instructors: Stuart Russell and Peyrin Kao

University of California, Berkeley
Outline

1. Propositional Logic I
   - Basic concepts of knowledge, logic, reasoning
   - Propositional logic: syntax and semantics, Pacworld example

2. Propositional logic II
   - Inference by theorem proving (briefly) and model checking
   - A Pac agent using propositional logic
Agents that know things

- Agents acquire knowledge through perception, learning, language
  - Knowledge of the effects of actions ("transition model")
  - Knowledge of how the world affects sensors ("sensor model")
  - Knowledge of the current state of the world
- Can keep track of a partially observable world
- Can formulate plans to achieve goals
- Can design and build gravitational wave detectors.....
LIGO
Knowledge, contd.

- Knowledge base = set of sentences in a formal language
- Declarative approach to building an agent (or other system):
  - **Tell** it what it needs to know (or have it **Learn** the knowledge)
  - Then it can **Ask** itself what to do—answers should follow from the KB
- Agents can be viewed at the **knowledge level**
  i.e., what they **know**, regardless of how implemented
- A single inference algorithm can answer any answerable question

<table>
<thead>
<tr>
<th>Knowledge base</th>
<th>Domain-specific facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inference engine</td>
<td>Generic code</td>
</tr>
</tbody>
</table>
Logic

- **Syntax**: What sentences are allowed?
- **Semantics**: What are the possible worlds? Which sentences are true in which worlds? (i.e., definition of truth)
Different kinds of logic

- **Propositional logic**
  - Syntax: \( P \lor (\neg Q \land R); \quad X_1 \iff (\text{Raining} \Rightarrow \neg \text{Sunny}) \)
  - Possible world: \{\( P = \text{true}, Q = \text{true}, R = \text{false}, S = \text{true} \)\} or 1101
  - Semantics: \( \alpha \land \beta \) is true in a world iff is \( \alpha \) true and \( \beta \) is true (etc.)

- **First-order logic**
  - Syntax: \( \forall x \exists y \, P(x,y) \land \neg Q(\text{Joe},f(x)) \Rightarrow f(x)=f(y) \)
  - Possible world: Objects \( o_1, o_2, o_3 \); \( P \) holds for \( <o_1,o_2> \); \( Q \) holds for \( <o_3> \);
    \( f(o_1)=o_1; \text{Joe}=o_3; \) etc.
  - Semantics: \( \phi(\sigma) \) is true in a world if \( \sigma = o_j \) and \( \phi \) holds for \( o_j \); etc.
Different kinds of logic, contd.

- **Relational databases:**
  - Syntax: ground relational sentences, e.g., $\text{Sibling}(\text{Ali}, \text{Bo})$
  - Possible worlds: (typed) objects and (typed) relations
  - Semantics: sentences in the DB are true, everything else is false
    - Cannot express disjunction, implication, universals, etc.
    - Query language (SQL etc.) typically some variant of first-order logic
    - Often augmented by first-order rule languages, e.g., Datalog
  - Knowledge graphs (roughly: relational DB + ontology of types and relations)
    - Google Knowledge Graph: 5 billion entities, 500 billion facts, >30% of queries
    - Facebook network: 2.93 billion people, trillions of posts, maybe quadrillions of facts
Inference: entailment

**Entailment:** \( \alpha \models \beta \) ("\( \alpha \) entails \( \beta \)" or "\( \beta \) follows from \( \alpha \)"") iff in every world where \( \alpha \) is true, \( \beta \) is also true

- I.e., the \( \alpha \)-worlds are a **subset** of the \( \beta \)-worlds \([\textit{models}(\alpha) \subseteq \textit{models}(\beta)]\)

- In the example, \( \alpha_2 \models \alpha_1 \)

- (Say \( \alpha_2 \) is \( \neg Q \land R \land S \land W \)
  \( \alpha_1 \) is \( \neg Q \))
Inference: proofs

- A proof is a *demonstration* of entailment between $\alpha$ and $\beta$
- **Sound** algorithm: everything it claims to prove is in fact entailed
- **Complete** algorithm: every that is entailed can be proved
Inference: proofs

- **Method 1: model-checking**
  - For every possible world, if $\alpha$ is true make sure that is $\beta$ true too
  - OK for propositional logic (finitely many worlds); not easy for first-order logic

- **Method 2: theorem-proving**
  - Search for a sequence of proof steps (applications of *inference rules*) leading from $\alpha$ to $\beta$
  - E.g., from $P$ and $(P \Rightarrow Q)$, infer $Q$ by *Modus Ponens*
Propositional logic syntax

- Given: a set of proposition symbols \( \{X_1, X_2, \ldots, X_n\} \)
  - (we often add True and False for convenience)
- \( X_i \) is a sentence
- If \( \alpha \) is a sentence then \( \neg \alpha \) is a sentence
- If \( \alpha \) and \( \beta \) are sentences then \( \alpha \land \beta \) is a sentence
- If \( \alpha \) and \( \beta \) are sentences then \( \alpha \lor \beta \) is a sentence
- If \( \alpha \) and \( \beta \) are sentences then \( \alpha \Rightarrow \beta \) is a sentence
- If \( \alpha \) and \( \beta \) are sentences then \( \alpha \Leftrightarrow \beta \) is a sentence
- And p.s. there are no other sentences!
Propositional logic semantics

- Let $m$ be a model assigning true or false to $\{X_1, X_2, ..., X_n\}$
- If $\alpha$ is a symbol then its truth value is given in $m$
- $\neg\alpha$ is true in $m$ iff $\alpha$ is false in $m$
- $\alpha \land \beta$ is true in $m$ iff $\alpha$ is true in $m$ and $\beta$ is true in $m$
- $\alpha \lor \beta$ is true in $m$ iff $\alpha$ is true in $m$ or $\beta$ is true in $m$
- $\alpha \Rightarrow \beta$ is true in $m$ iff $\alpha$ is false in $m$ or $\beta$ is true in $m$
- $\alpha \Leftrightarrow \beta$ is true in $m$ iff $\alpha \Rightarrow \beta$ is true in $m$ and $\beta \Rightarrow \alpha$ is true in $m$
Example: Partially observable Pacman

- Pacman knows the map but perceives just wall/gap to NSEW

**Formulation:** *what variables do we need?*

- Wall locations
  - Wall_0,0 there is a wall at [0,0]
  - Wall_0,1 there is a wall at [0,1], etc. (*N* symbols for *N* locations)

- Percepts
  - Blocked_W (blocked by wall to my West) etc.
  - Blocked_W_0 (blocked by wall to my West *at time 0*) etc. (*4T* symbols for *T* time steps)

- Actions
  - W_0 (Pacman moves West at time 0), E_0 etc. (*4T* symbols)

- Pacman’s location
  - At_0,0_0 (Pacman is at [0,0] at time 0), At_0,1_0 etc. (*NT* symbols)
How many possible worlds?

- $N$ locations, $T$ time steps $\Rightarrow N + 4T + 4T + NT = O(NT)$ variables
- $O(2^{NT})$ possible worlds!
- $N=200$, $T=400$ $\Rightarrow \sim 10^{24000}$ worlds
- Each world is a complete “history”
  - But most of them are pretty weird!
Pacman’s knowledge base: Map

- Pacman knows where the walls are:
  \[ \text{Wall}_{0,0} \land \text{Wall}_{0,1} \land \text{Wall}_{0,2} \land \text{Wall}_{0,3} \land \text{Wall}_{0,4} \land \text{Wall}_{1,4} \land \ldots \]
- Pacman knows where the walls aren’t!
  \[ \neg \text{Wall}_{1,1} \land \neg \text{Wall}_{1,2} \land \neg \text{Wall}_{1,3} \land \neg \text{Wall}_{2,1} \land \neg \text{Wall}_{2,2} \land \ldots \]
Pacman’s knowledge base: Initial state

- Pacman doesn’t know where he is
- But he knows he’s somewhere!
  - \( \text{At}_1,1_0 \lor \text{At}_1,2_0 \lor \text{At}_1,3_0 \lor \text{At}_2,1_0 \lor \ldots \)
Pacman’s knowledge base: Sensor model

- State facts about how Pacman’s percepts arise...
  - \(<\text{Percept variable at } t> \iff <\text{some condition on world at } t>\)
  - Pacman perceives a wall to the West at time \(t\) if and only if he is in \(x,y\) and there is a wall at \(x-1,y\)
    - \(\text{Blocked}_W_0 \iff ((\text{At}_1,1)_0 \land \text{Wall}_0,1) \lor (\text{At}_1,2)_0 \land \text{Wall}_0,2) \lor (\text{At}_1,3)_0 \land \text{Wall}_0,3) \lor \ldots \)
  - 4\(T\) sentences, each of size \(O(N)\)
  - Note: these are valid for any map
Pacman’s knowledge base: Transition model

- How does each state variable at each time get its value?
  - Here we care about location variables, e.g., At_3,3_17

- A state variable $X$ gets its value according to a successor-state axiom
  - $X_t \iff [X_{t-1} \land \neg (\text{some action}_{t-1} \text{ made it false})] \lor \neg X_{t-1} \land (\text{some action}_{t-1} \text{ made it true})$

- For Pacman location:
  - $At_{3,3}_17 \iff [At_{3,3}_16 \land \neg ((\neg Wall_{3,4} \land N_{16}) \lor (\neg Wall_{4,3} \land E_{16}) \lor ...)] \lor \neg At_{3,3}_16 \land ((At_{3,2}_16 \land \neg Wall_{3,3} \land N_{16}) \lor (At_{2,3}_16 \land \neg Wall_{3,3} \land N_{16}) \lor ...)]$
How many sentences?

- Vast majority of KB occupied by $O(NT)$ transition model sentences
  - Each about 10 lines of text
  - $N=200, T=400 \Rightarrow \sim 800,000$ lines of text, or 20,000 pages
- This is because propositional logic has limited expressive power
- Are we really going to write 20,000 pages of logic sentences???
- No, but your code will generate all those sentences!
- In first-order logic, we need $O(1)$ transition model sentences
- (State-space search uses atomic states: how do we keep the transition model representation small???)
Some reasoning tasks

- **Localization** with a map and local sensing:
  - Given an initial KB, plus a sequence of percepts and actions, where am I?

- **Mapping** with a location sensor:
  - Given an initial KB, plus a sequence of percepts and actions, what is the map?

- **Simultaneous localization and mapping**:
  - Given ..., where am I and what is the map?

- **Planning**:
  - Given ..., what action sequence is guaranteed to reach the goal?

- **ALL OF THESE USE THE SAME KB AND THE SAME ALGORITHM!!**
Summary

- One possible agent architecture: knowledge + inference
- Logics provide a formal way to encode knowledge
  - A logic is defined by: syntax, set of possible worlds, truth condition
- A simple KB for Pacman covers the initial state, sensor model, and transition model
- Logical inference computes entailment relations among sentences, enabling a wide range of tasks to be solved