Discussion Section

Matthew Tong
HW #1

• 1) Define
  – Intelligence
  – AI
  – Agent
2) Intractable/unsolvable => no AI
   - Big difference between the two
     - Intractable leads open possibility of better machines or algorithms
       - Moore’s Law
       - Better algorithms
     - Unsolvable means cannot be computed. Period. No bigger guns.
       - But problems that are unsolvable for people
         » “This is a statement that Matt cannot know is true.”
       - Heuristics
• 3) Only what told
  – Search can lead to solutions a person couldn’t find
  – Learning
  – Self-modifying code
  – We’ll see more examples later on
• 4a) Robot soccer player
  – Partially observable (will not have full 4D representation of field)
  – Stochastic ( anyone who says deterministic must kick a ball 100 yards and know to the nearest mm where it will end up)
    • Not competitive since more unknowns than other players
  – Sequential
  – Continuous
  – Multi
• 4b) Internet shopping agent
  – Lot of theoretically possible interpretations on this
  – I pictured something where you would pass a list of books and it would try to buy them at the cheapest possible prices (including possibly lumping orders for cheaper shipping, etc).
• 4b) Book shopper
  – Partially - Will have to query each vendor
    • Enough said fully that I accepted - Depends on whether queries are actions or whether net is memory
  – Deterministic - Results of actions are known!
  – Episodic
  – Static - Prices won’t change over single session
  – Discrete - Books are units, prices are rationals, etc.
  – Single agent (unless bidder)
• 4c) Mars Rover
  – Partially
  – Stochastic
  – Sequential
  – Static! (That was key to why Mars - could spend minutes deliberating or communicating knowing nothing would change)
  – Sing agent
• 4d Theorem prover
  – Fully (Knows all facts has to work with)
  – Deterministic (Proofs)
  – Sequential/Episodic - One theorem, but multiple steps
  – Static (But semi since mathematician wants semi-quick answers)
  – Single
• 5) Successor function $\leftrightarrow$ actions, results
  – Successor is set of action state pairs
  – Successor-fn(s)
    • For each a in Legal-Actions(s)
      – $S' \leftarrow$ Results(a, s)
      – Add (a, s’) to list
    • Return list
• 5) Successor function $\leftrightarrow$ actions, results
  – Results(a, s)
    • Successors = Successors-Fn(s)
    • For $i = 1$:\text{length(successors)}
      – If a in successors(i)
        » Return s in successors (i)
    • Error, return null, etc
• 5) Successor function $\Leftarrow\Rightarrow$ actions, results
  – Results($a, s$)
    • Successors = Successors-Fn($s$)
    • For $i = 1$:length(successors)
      – Add $a$ in successors($i$) to list
    • Return list
6) Spell out problem with initial state, etc.
   - Initial state - Needed to spell out conditions fully
     (where are boxes, where is banana, where is monkey, etc)
   - Goal test - Needed to solve problem (ie monkey has banana)
   - Successor function
     - Lot of points lost here.
     - Return sets of pairs, not a single successor state
     - Needed to capture list of actions described
   - Cost - Needed to be relevant and implementable
• 6c) Bad record problem
  – Lots of people misinterpreted this or confined to inefficient solutions
  – A file contains some number of records
  – The current file is bad because one record is bad
  – Submit a sequence of files to find the bad record
  – Often people confined to linear search, removing the optimal solution (depending on cost function)
  – Binary search!
Programming Assignment

• Wonderful API in Java
  – Stack
  – Queue
  – Priority queue

• Object oriented language, use it
  – Puzzle class
  – Node class
  – Generic Solver class
  – All your various solver class inherit from generic Solver.
  • I believe could essentially just change which List subtype was used (aside from ID which of course could just call DF many times).
Constraint Satisfaction Problems

Chapter 5
Constraint satisfaction problems (CSPs)

Standard search problem:
  state is a “black box”—any old data structure
  that supports goal test, eval, successor

CSP:
  state is defined by variables $X_i$ with values from domain $D_i$

  goal test is a set of constraints specifying
  allowable combinations of values for subsets of variables

Simple example of a formal representation language

Allows useful general-purpose algorithms with more power
than standard search algorithms
Example: Map-Coloring

Variables $WA, NT, Q, NSW, V, SA, T$

Domains $D_i = \{\text{red, green, blue}\}$

Constraints: adjacent regions must have different colors

- e.g., $WA \neq NT$ (if the language allows this), or
- $(WA, NT) \in \{(\text{red, green}), (\text{red, blue}), (\text{green, red}), (\text{green, blue}), \ldots\}$
Example: Map-Coloring contd.

Solutions are assignments satisfying all constraints, e.g.,
\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}
Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints

General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!
Varieties of CSPs

Discrete variables
finite domains; size $d \Rightarrow O(d^n)$ complete assignments
  ◇ e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)
  infinite domains (integers, strings, etc.)
  ◇ e.g., job scheduling, variables are start/end days for each job
  ◇ need a constraint language, e.g., $Start.Job_1 + 5 \leq Start.Job_3$
  ◇ linear constraints solvable, nonlinear undecidable

Continuous variables
  ◇ e.g., start/end times for Hubble Telescope observations
  ◇ linear constraints solvable in poly time by LP methods
Varieties of constraints

Unary constraints involve a single variable,
  e.g., \( SA \neq green \)

Binary constraints involve pairs of variables,
  e.g., \( SA \neq WA \)

Higher-order constraints involve 3 or more variables,
  e.g., cryptarithmetic column constraints

Preferences (soft constraints), e.g., \( red \) is better than \( green \)
  often representable by a cost for each variable assignment
    \( \rightarrow \) constrained optimization problems
Constraint Graph for Sudoku

- Still expressable as binary constraints
- Fairly redundant \((9 \times 8 \times 3)/2\)
Example: Cryptarithmetic

\[
\begin{array}{c}
\text{TWO} \\
+ \text{TWO} \\
\hline
\text{FOUR}
\end{array}
\]

Variables: \(F, T, U, W, R, O, X_1, X_2, X_3\)
Domains: \(\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}\)
Constraints

\(\text{alldiff}(F, T, U, W, R, O)\)
\(O + O = R + 10 \cdot X_1, \text{ etc.}\)
Real-world CSPs

Assignment problems
  e.g., who teaches what class

Timetabling problems
  e.g., which class is offered when and where?

Hardware configuration

Spreadsheets

Transportation scheduling

Factory scheduling

Floorplanning

Notice that many real-world problems involve real-valued variables
Standard search formulation (incremental)

Let’s start with the straightforward, dumb approach, then fix it

States are defined by the values assigned so far

◊ **Initial state:** the empty assignment, `{}`

◊ **Successor function:** assign a value to an unassigned variable that does not conflict with current assignment.
  ⇒ fail if no legal assignments (not fixable!)

◊ **Goal test:** the current assignment is complete

1) This is the same for all CSPs! 😊
2) Every solution appears at depth $n$ with $n$ variables
  ⇒ use depth-first search
3) Path is irrelevant, so can also use complete-state formulation
4) $b = (n - \ell)d$ at depth $\ell$, hence $n!d^n$ leaves!!!! 😞
Backtracking search

Variable assignments are commutative, i.e.,
\[\text{[WA} = \text{red then NT} = \text{green]} \text{ same as } \text{[NT} = \text{green then WA} = \text{red]}\]

Only need to consider assignments to a single variable at each node
\[\Rightarrow b = d \text{ and there are } d^n \text{ leaves}\]

Depth-first search for CSPs with single-variable assignments is called backtracking search

Backtracking search is the basic uninformed algorithm for CSPs

Can solve \(n\)-queens for \(n \approx 25\)
Backtracking example
Improving backtracking efficiency

*General-purpose* methods can give huge gains in speed:

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?
This kind of bookkeeping helps with many of the heuristics as well, so we’re having you do it from the start.
Minimum remaining values (MRV):
choose the variable with the fewest legal values
Degree heuristic

Tie-breaker among MRV variables

Degree heuristic:
choose the variable with the most constraints on remaining variables
Least constraining value

Given a variable, choose the least constraining value:
the one that rules out the fewest values in the remaining variables

Combining these heuristics makes 1000 queens feasible
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for every value $x$ of $X$ there is some allowed $y$
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff
for every value $x$ of $X$ there is some allowed $y$
Arc consistency

Simplest form of propagation makes each arc consistent

\( X \rightarrow Y \) is consistent iff
for every value \( x \) of \( X \) there is some allowed \( y \)

If \( X \) loses a value, neighbors of \( X \) need to be rechecked
Arc consistency

Simplest form of propagation makes each arc consistent

\( X \rightarrow Y \) is consistent iff

for every value \( x \) of \( X \) there is some allowed \( y \)

If \( X \) loses a value, neighbors of \( X \) need to be rechecked

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment
Strong k-consistency

• A CSP is strongly k-consistent if for any set of k-1 variables and any constraint assignment to those, a consistent value can be assigned to any kth variable
• 1-consistency would just check nodes
• 2-consistency is arc consistency
• 3-consistency (path consistency) - Any pair of variables can always be extended to a third neighboring variable
  – Would catch that WA=red, NSW = red is inconsistent
Strong $k$-consistency

• If $n$ variables and doing $k$-consistency, no search is necessary
  – But exponential!!
  – Middle ground is key

• General point is general CSP. But this is where domain knowledge can figure in.
  – In Sudoku, X-wing configuration is a kind of 3-consistency