2. State-space search

Inquiry

• Can many problems solved using intelligence may be reduced to the search of an exponential-sized state space?

• Is it AI’s task to give approximate solutions to NP-hard problems?
2. State-space search

Objectives

2a. Define goal-based state-space search
2b. Construct a game tree and perform a goal-driven analysis of it
2c. Explain how heuristics are used to provide adequate solutions to hard search problems

1. State-space search

- *State space*: A set of possible arrangements of values, e.g.:
  - Board configurations in board games
  - Paths in a graph
  - Arrangements of items in a knapsack
  - Assignments of truth values in a formula
- *Search* may be accomplished by *transitions* from state to state, which we can express as a *tree*
- *Strategy*: Reduce size of the state-space tree explored by the algorithm, by pruning branches that cannot lead to a solution
2. State-space search

Goal-driven search

• Goal: the set of environment states in which the goal condition is satisfied
• Reaching a goal state may consist of following a series of transitions between states
• State-space search by goal-driven agents: a search for a path consisting of edges in the state-transition graph from start state to a goal
• Optimizing search compares costs of paths in weighted graph

Reformulating problems

• Problem solving may include converting the form of a problem to a standard form, such as a graph problem
• Example: convert maze to a graph; see Assignment 2]
Examples of state-space search

- *Tic-tac-toe*: goal is a state with three of player’s symbols in a row, column, or diagonal
- *8-puzzle* (tiles move left, right, up, down in an 8 x 8 square): goal is some ordering of tiles, such as ascending numerals from 1 to 63
- *Traveling salesperson* problem: goal is a minimum path through all vertices of a graph; state includes path
- *Chess*

Decision trees for search

- The decision tree for a search of an array must examine each element
- Searching an *unsorted* array, only one element is considered per step
- Hence the tree depth is O(n)
- Searching a *sorted* array, considering the middle element enables a decision to search only the right or left half of the array
- Hence the decision tree depth is O(lg n)
Generate and test

A general-purpose algorithm for state-space search

To solve a given problem:

1. Design:
   a. a test of proposed solutions
   b. a generator of possible solutions

2. While (¬problem-solved ∧ ¬time-expired)
   - generate a possible solution
   - test it

State-space search strategies

• Data-driven (forward chaining): apply rules to facts to generate new facts, searching for a path to goal
• Goal-driven (backward chaining): work backward from goal through subgoals to original facts
2. State-space search

Uninformed search

- Uninformed search is blind, as opposed to heuristic or informed search
- \textit{BFS}: large memory requirements; finds short-path solutions
- \textit{DFS}: small memory requirements; may run infinitely
- \textit{Depth-limited (DLS)}: modification of DFS

Two search approaches

- \textit{Depth-first search}: Label each vertex of graph with a number in order visited, using the strategy of exhausting one path before backtracking to start at a new edge
- \textit{Breadth-first search}: Label each vertex, using the strategy of visiting each vertex adjacent to the current one at each stage
State-space search example

- 8-puzzle, where state space is the possible arrangements of tiles, numbered from 1 to 8, in a 3 x 3 grid, leaving one space empty
- Goal is to move one tile at a time into the empty space, reaching goal state from start state
- [pic of sample start, goal]

Depth-first search of graph

\[ \text{GDFS}(G) \]

\[ G = (V, E) \]

> **Pre:** there is a vertex \( v \) in \( V \) s.t. \( \text{Mark}[v] = 0 \)

\[ \text{count} \leftarrow 0 \]

for each vertex \( v \in V \) do

if \( \text{Mark}(v) = 0 \)

\[ \text{vdfs}(v, \text{count}) \]

return \( \text{Mark} \)

> **Post:** Vertices are marked in order of some DFS traversal

Running time: _____
2. State-space search

Depth-first search from vertex

\( \text{vdfs}(v, V) \) // recursive

\[
\begin{align*}
\text{count} & \leftarrow \text{count} + 1 \\
\text{Mark}[v] & \leftarrow \text{count} \\
\end{align*}
\]

For each vertex \( w \in V \) adjacent to \( v \) do

\[
\begin{align*}
\text{if Mark}[w] = 0 & \\
\text{vdfs}(w, V) & \\
\end{align*}
\]

> Post: \( v \)'s mark is set

**Challenge**: show termination

[See URL found by B. Grozier]

Breadth-first search

- Checks all vertices 1 edge from origin, then 2 edges, then 3, etc., to find path to destination
- If \( G = \langle \{a..i\}, (a,b), (a,c), (a,d), (c,d), (b,e), (b,f), (c,g), (g,h), (g,i), (h,i) \rangle \), and
  - BFS input is \( G, a, f \), then
  - Order of search is \( (a,b), (a,c), (a,d), (b,e), (b,f) \)

[pic]
Breadth-first search

\[
BFS(G) \quad G = (V, E)
\]
\[
\begin{align*}
\text{count} & \leftarrow 0 \\
\text{for each vertex } v \in V \text{ do} & \\
\quad & \text{if Mark}(v) = 0 \\
\quad & \quad \text{bfs}(v)
\end{align*}
\]
\[
\text{bfs}(v) \quad \text{count} \leftarrow \text{count} + 1 \\
\quad \text{Mark}(v) \leftarrow \text{count} \\
\quad \text{Queue} \leftarrow (v) \\
\text{While not empty(queue) do} \\
\quad \text{For each vertex } w \in V \text{ adjacent to front(queue) do} \\
\quad \quad & \text{if Mark}(w) = 0 \\
\quad \quad \quad \text{count} \leftarrow \text{count} + 1 \\
\quad \quad \quad \text{append } w \text{ to queue} \\
\quad \text{Remove front of } \text{queue}
\]

- Iterative
- Proceeds concentrically
- Uses queue rather than DFS’s stack
- See example (pic)

Backtracking and branch-and-bound

- These approaches solve some search problems in large state spaces
- It is very hard to predict whether these approaches will solve a particular problem in reasonable time
- Backtrack example: Finding an English word that satisfies a certain definition [why?]
Partially informed search

- If environment is partially observable, or effects of actions are uncertain, then agent must act on *contingencies* provided by new percepts
- *Exploration* may be required; i.e., actions that generate useful percepts
- *Belief states* are sets of states of the environment

Backtracking
Branch-and-bound strategy

- Prunes tree by maintaining a *bound* (minimum path found so far), eliminating the need to search paths that exceed the bound
- *Example*: Finding best-score word in a turn at Scrabble

### Backtrack \((X[1..i])\)

If \(X[1..i]\) is a solution
- return \(X[1..i]\)
else
- for each state \(q\) in next(\(X[1..i]\))
  - \(X[i+1] \leftarrow q\)
  - return \(\text{Backtrack}(X[1..i+1])\)

Return \(\text{fail}\)

- How does this differ from depth-first search?
- Why does recursive call have *larger* parameter than original parameter?
Tree search algorithm

\[
path \leftarrow \lambda
\]
do
    if cannot expand any node
        return FAIL
    \( v \leftarrow \) node in tree chosen for expansion
    if \( v \) contains goal state
        return \( path + v \)
    else
        expand \( v \)

- Nodes contain state, parent, action, path cost, depth
- Nodes available to expand are those on fringe of expanding tree

Search problems and solutions

- **Problem:**
  - Initial state
  - Set of actions
  - Goal test function (defines set of goal states)
  - Path cost
- **Solution:** path from initial state to goal state
- **Algorithm evaluation criteria**
  - Completeness
  - Optimality
  - Time and space complexity
Iterated local search (hill climbing)

```plaintext
Search (S)
repeat
  c ← random (S)
repeat
  changed ← false
  for each t ∈ T
  c' ← t(c)
  if eval(c') > eval(c)
    c ← c'
    changed ← true
  until ¬changed
until eval(c) is acceptable
return c
```

- Explores search space \( S \) using randomization, transformation set \( T \), and fitness function \( \text{eval} \)
- Addresses \( n \)-queens, TSP, SAT
- Similar algorithm, simulated annealing, solves independent-set problem

2. Hard problems in constraint and optimization

- Constraint problem: To find some value that satisfies a set of constraints or conditions
- Constraint problem examples:
  - Search
  - Pattern matching
  - Sort of a set of records
- Optimization problem: to find maximum or minimum valued solution to a constraint problem, among all solutions
**Combinatorial explosion**

- Suppose we want to choose the best $n$ things to buy or do, of $k$ choices, or the best plan, with $n$ steps and $k$ options at each step, to accomplish something
- Then the space of combinations is $C(k, n)$, and plans, $k^n$; this is the number of overall choices to consider
- *Combinatorial explosion*: as $n$ goes to 100 (i.e., state space goes to $2^{100}$), no computer could consider all possibilities

**Constraint satisfaction problems**

*Problem:*
- A set $x$ of variables $x_1, x_2, ..., x_n$
- A set $C$ of constraints $C_1, C_2, ..., C_m$, that each specifies acceptable combinations of values for a certain subset of $x$
- A variable assignment to $x$ that does not violate any of $C$ is *consistent* or legal
- A variable assignment to all of $x$ is *complete*
- *Solution*: a complete consistent variable assignment
General form of CSPs and solutions

- *Advantage* of formulating problems as CSPs: standard state representations enable generic transition functions, goal tests, heuristics
- *Form of state*: set of partial variable assignments
- *Initial state*: no assignments
- *Successor function*: assigns value to one variable while violating no constraint
- *Goal test*: variable assignments are complete
- Order-independence of variable assignments makes *backtracking* helpful

Example: satisfiability (SAT)

- Given the constraint of a formula $\phi$ in propositional logic (logic with $\neg$, $\wedge$, $\vee$, $\Rightarrow$, no predicates, no quantifiers), does a set of variable assignments exist that satisfies $\phi$ (makes $\phi$ true)?
- *Examples:*
  (a) $p \land q \land r$
  (b) $(p \land q \lor \neg q)$
  (c) $p \land \neg(q \lor \neg q)$
Knowledge, goals, and rationality

• Levels of a computer system: device, circuit, symbol, … knowledge (Newell, 1982)

• Principle of rationality: An agent will select an action if it has knowledge that the action will lead to a system goal

• Knowledge: Whatever an agent has that enables it to compute its actions so as to reach goal

Bounded rationality

• Notion suggested by Herbert Simon, 1972, as alternative to classical rationality assumption of economic theory

• Argument: Humans have limited knowledge and resources for decision making

• Alternative goal to optimality: satisficing (good enough)

• Rational agent: one that chooses actions that yield maximum expected reward averaged over all outcomes
Reducing difficulty of constraint-satisfaction problems

- *Local search* with randomization, i.e., making small adjustments to reduce constraint violations, works well when solutions are densely distributed through the state space.
- *Decomposing problems* into subproblems can greatly reduce solution time.
- *Tree-structured CSPs* can be solved in $O(n)$ time.

Optimization problems

**Examples:**

- *Closest pair*
  Given a set of ordered pairs, points on a Cartesian plane, find two points that are closest together.

- *Shortest path:* For vertices $u$, $v$, in weighted graph $G$, find shortest path from $u$ to $v$.

- *Bin packing:* Find a packing that minimizes the number of bins.
The function-optimization problem

- Let \( f : \mathbb{N}^k \rightarrow \mathbb{R} \) for some \( k \) (the *arity* of \( f \))
- **Problem:** Find some \( x \in \mathbb{N}^k \) s.t. \( f(x) \) is maximal
- **Example:** Suppose \( x \) is the set of proportions of ingredients in a fuel mixture, \( f(x) \) is fuel efficiency under this mixture
- **Optimizing** \( f(x) \) means finding the most efficient mixture
- For an *algorithm* to optimize a function we must have \( f : X \rightarrow Y \) with \( X, Y \) finite

3. Intractability

- We analyze the complexity of *problems*, as opposed to *algorithms*
- Complexity of a problem is complexity of the most efficient algorithm that solves it
- We prove complexity of hard problems by *reducing* a hard problem of known complexity to a problem of unknown complexity
- Problems may be categorized as *tractable* (\( P \), polynomial-time) or *intractable* (NP-hard)
Intractable problems

• Some problems have no known polynomial time ($O(n^k)$) solutions for any constant $k$
• These are considered intractable because for sufficient $n$ they may take “forever” in practice (note this differs from Levitin definition)
• Exponential-time examples: Hanoi, password guessing, understanding English
• Others are called $NP$-complete problems: solutions are checkable, but not known to be obtainable, in polynomial time

Satisfiability (SAT)

• Given a formula $\phi$ in propositional logic (logic with $\neg$, $\land$, $\lor$, $\Rightarrow$, no predicates, no quantifiers), does a set of variable assignments exist that satisfies $\phi$ (makes $\phi$ true)?
• Examples:
  (a) $p \land q \land r$
  (b) $(p \land q \lor \neg q)$
  (c) $p \land \neg(q \lor \neg q)$
The function-optimization problem

- Let \( f : \mathbb{N}^k \rightarrow \mathbb{R} \) for some \( k \) (the *arity* of \( f \))
- **Problem**: Find some \( x \in \mathbb{N}^k \) s.t. \( f(x) \) is maximal
- **Example**: Suppose \( x \) is the set of proportions of ingredients in a fuel mixture, \( f(x) \) is fuel efficiency under this mixture
- **Optimizing** \( f(x) \) means finding the most efficient mixture
- For an *algorithm* to optimize a function we must have \( f : X \rightarrow Y \) with \( X, Y \) finite

Polynomial time complexity

- \( P = \bigcup_{k \in \mathbb{N}} DTIME(n^k) \)
- That is, \( P \) is the set of problems decidable in \( O(n^k) \) time (polynomial time), where \( n \) is the size of the problem and \( k \) is a constant
- **Examples of problems in class \( P \)**
  - Searching a collection is in \( DTIME(n) \)
  - Sorting an array, \( O(n \log n) \)
  - Searching a BST, \( O(\log n) \)
  - Generating a graph reachability matrix, \( O(n^3) \)
Exponential time

- $EXPTIME = DTIME(2^n)$

- $SAT$ seems to be in $EXPTIME$ but not $P$, because it seems that $O(2^n)$ candidate solutions need to be generated and tested.

The $NP$ complexity class

- $NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$

- That is, $NP$ is the set of problems decidable in polynomial time by nondeterministic algorithms.

- Intuition: $NP$ problems are those for which a particular candidate solution, once found, can be verified in polynomial time.

- Question: Is $P = NP$?
**NP completeness**

- For many problems in NP, e.g., SAT, no polynomial time solution is known
- Problems to which SAT or similar problems are reducible are called *NP-complete*
- *Example:* Traveling Salesperson
- If *any* of these has a polynomial-time solution, then *all* do, hence \( P = NP \)
- Most researchers think \( P \neq NP \), i.e., NP-complete problems are believed to have no polynomial-time solutions

**SAT, NPC, and \( P =? NP \)**

- *Theorem* (Cook): *Every* problem in NP is reducible to SAT
- *Proof:* Construct a nondeterministic program that decides the problem, convert its computation steps to a CNF formula
  - [Explain]
- *Theorem:* \( P = NP \) iff SAT \( \in P \)
Intractability, NPC, EXPTIME

• Certain problems are thought or known to have only exponential-time, $O(2^n)$, solutions
• \( \text{EXPTIME} \) and NPC are the \( \text{NP-hard} \) problems, considered intractable
• Problems of planning, scheduling, routing, drawing inferences, understanding language, etc., are in general intractable
• What to do: Replace intractable problem with a simpler one, e.g., one with a probabilistic or approximate solution

Approximation and probabilistic algorithms

• Intractable (NP-hard) problems are considered not worth solving exactly
• Some algorithms can approximate an optimal solution very closely
• Randomization of data can help
• Optimization example: Evolutionary computation, based on natural evolution
Approximation algorithms

- Many use *heuristics*: rules of thumb based on human experience; e.g., putting largest item in box first
- **Example of heuristic**: putting largest item in knapsack first
- **Accuracy ratio**: $f(s_a) / f(s^*)$, where heuristic $s_a$ solves the problem of minimizing $f$ and $s^*$ is an exact solution
- **Performance ratio**: lowest upper bound on accuracy ratio
- **$c$-approximation algorithm**: one whose performance ratio is at most $c$, i.e., $f(s_a) \leq f(s^*)$ for any instance
- **Continuous functions**: Newton’s method
  
  \[-x_{n+1} = (x_n + f(x_n) + f'(x_n)) \div 2\]
  gives approximate sequence; e.g., for square root

Randomized algorithms

- Use of probabilistic control usually leads, with sufficient time, to good approximate solutions
- Often used for optimization problems
- **Examples**: natural evolution; evolutionary computation, which generates a population of solutions stochastically, tests them, and uses test results to generate more
2. State-space search

• Problem class: Optimization under constraints
• Strategy: Find a feasible solution, improve it by successive steps
• Obstacle: Global maxima/minima (the objective) may differ from local ones
• Cases:
  – Simplex method for linear programming
  – Maximal flow
  – Maximal bipartite matching
  – Stable marriage
  – Hill climbing

4. Heuristics

• Definition: rules that guide a system to choices in a state-space search that are likely to lead to a satisfactory solution
• Necessary if:
  – Problem lacks exact solution due to ambiguity of problem statement or data, or lack of information
  – Problem is intractable; solution may encounter combinatorial explosion requiring exponential time to solve
Informed search

- *Greedy best-first* using state-evaluation function to decide which is *believed* best
- *Evaluation function* $h$ is called a *heuristic* and is based on domain knowledge
- $h(v)$ is estimated cost of least-cost path from node $v$ to a goal node
- $A^*$ search: estimates cost to reach goal through node $v$ as sum of cost to reach $v$ and heuristic cost to reach goal from $v$
- *Theorem:* $A^*$ is optimal if $h(v)$ is admissible, i.e., never overestimates cost

Triangle inequality

- Suppose we wish to go from state $A$ to state $B$
- Then $\text{cost}(A, B) \leq \text{cost}(A, C) + \text{cost}(C, B)$
- If we can get from $A$ to $C$ and from $C$ to $B$ for a total cost $x$, then the cost from $A$ to $B$ can be no greater
- *Intuition:* No two sides of a triangle can sum to more than the length of the third
Simulated annealing and local beam search

- *Simulated annealing*: random choice is used to “shake up” the state transitions out of local optima
- *The intensity of shaking-up* is progressively reduced (analogous to lowering of temperature in metal annealing)
- *Beam search* uses a population of states, choosing the best performers for re-generation

Game theory

- Originated in economic theory
  - Mathematical methods for study of decisions
  - Includes notion of *rational* behavior (acting in own interests)
  - Includes notion of *utility*
  - Generalizes the notion of game strategy
- Variants (Von Neumann-Morgenstern, 1947):
  - zero-sum, non-zero-sum
  - 2-player, multi-player
  - perfect-information, imperfect-information
Adversarial search

- Some games are two-player, turn-taking, zero-sum, deterministic, with perfect-information
- Early AI research addressed such games because they are simple to represent and hard to solve
- Minimax algorithm implements DFS assuming optimal opponent move
- Alpha-beta pruning effectively reduces branching factor to its square root
- Cutoff tests using position-evaluation functions enable rational choices without looking ahead to end of game

Example: Playing a game

- To choose a move by Generate-and-test, generate possible moves, test them with an evaluation (utility) function (e.g., allocating points for pieces won by the move)
- Finding good moves often entails lookahead and backtrack in the game tree
The minimax algorithm

- Used for problems with an adversary; e.g., two-player games
- Assuming an optimal opponent, let the best move be the one that would yield the best-valued situation if the opponent replies with his/her best move
- To determine that, apply the same algorithm from opponent’s viewpoints

Example: Tic-tac-toe

- State space: the set of all possible board positions
- Let a position’s value be 1.0 if we win, 0.0 if we lose
- Also, if we can force a win (as at right), value is 1.0
Heuristics in chess

- *Example:* value of Queen is 9, rook 5, bishop 3, knight 3, pawn 1

Game trees

- In a game tree, vertices are board positions
- Edges are possible moves (transitions between board positions)
Aspects of heuristics

• **Limitations:** simple games are ideal for heuristics because
  – Representation is simple
  – All nodes have common representation

• Note that inference systems use heuristics as data; e.g., “people with savings and income should invest in stocks”

• **Confidence levels** for inference are heuristics

**Concepts**

A* search  depth-limited search  inference
backtracking  evaluation function  local search
backward chaining  exploration  minimax
belief state  forward chaining  path
best-first search  goal condition  state
breadth-first search  goal test  state transition
constraint  goal test function  state-space search
  satisfaction  goal-driven search  tree search
  problem  heuristic  triangle inequality
data-driven search  hill climbing  uninformed search
depth-first search
References
