2. State-space search

1. Constraint and optimization problems
2. Goal-driven search
3. Exhaustive search and intractability
4. Heuristics

Inquiry

- How do you solve problems?
- 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?
- How might (a) NAO or (b) Siri operate as a goal-based agent?
2. State-space search

**Topic objective**

2. Explain how heuristics offer ways to pursue goals in exponentially large search spaces

**Subtopic outcomes**

2.1 Explain what constraint and optimization problems are
2.2a Explain goal-based state-space search*
2.2b Perform a goal-driven analysis of a problem with a game tree*
2.3 Apply the definition of *intractability* to a computational problem
2.4 Explain how heuristics are used to provide adequate solutions to hard search problems*
2. State-space search

1. Constraint and optimization problems

- How do you choose a career?
- A friend?
- A partner?
- A plan for the weekend?

---

**Constraint and optimization problems**

- *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
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

Examples

• Constraint:
  – Search of an array of integers for a key value; for a value greater than 5; for a perfect square
  – Search of a database table for a record satisfying specification for three fields
• Optimization:
  – Search for the maximum perfect square
  – Search for the lowest-priced two-bath house
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 $\neg$, $\land$, $\lor$, $\Rightarrow$, 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) $p \land \neg(q \lor \neg q)$
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

---

Single-source shortest path

- In any weighted graph, from any source vertex, a tree exists composed of the set of shortest paths from the source to each other vertex

- The problem of a single shortest path reduces to building this tree (which is *not* the MST)

- *Dijkstra’s algorithm* builds this tree of shortest paths, starting from the source vertex
A shortest path’s subpaths are minimal

- Optimal-substructure property applies to shortest-path problem as follows
- If path \((u, ..., u', ..., v', ..., v)\) is minimal, then \((u', v')\) is minimal too (why?)
- So to minimize path \(u..v\), find shorter segments \(u'..v'\)

\[\begin{tikzpicture}
\node (u) at (0,0) {$u$};
\node (u') at (1,0) {$u'$};
\node (v) at (2,0) {$v$};
\node (v') at (1,0) {$v'$};
\draw (u) -- (u');
\draw (u') -- (v');
\draw (v') -- (v);
\end{tikzpicture}\]

- Note: more than one shortest path may connect two given vertices

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
2. State-space search

Subtopic outcome

2.1 Explain what constraint and optimization problems are

2. Goal-driven search

• What are some of your goals?
• How do you pursue them?
Goal-based problems

- **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

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
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

• **Planning** is defining a sequence of actions to achieve a goal

• **Algorithm evaluation criteria:** Completeness; optimality; time and space complexity

Planning

• **Classical planning environments:** fully observable, deterministic, finite state, discrete

• Much planning relies on partial decomposition of problems

• Actions occur when preconditions are met

• **Plan construction** consists of search in space of possible actions for such a sequence

• **Challenge:** the frame problem, which consists of deciding what exactly are the changes in the situation caused by an action
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
- *Hamiltonian path problem*: goal is a path through all vertices of a graph back to start
- *Traveling salesperson* problem: goal is a minimum such path
- *Chess*: goal is checkmate by player

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, such as ascending numerals from
State-space search strategies

- **Data-driven** (forward chaining): apply rules to facts to generate new facts, seeking a path to goal
- **Goal-driven** (backward chaining): work backward from goal through subgoals to original facts
- **Generate and test:**
  - Design a test of proposed solutions and a generator of possible solutions
  - While \((-problem-solved \land \neg time-expired)\)
    - generate a possible solution
    - test it

Uninformed search

- We may represent state spaces as graphs, with transition rules for going from state to state
- Uninformed search is blind, as opposed to heuristic or informed search
- **Depth-first search:** deep, searching one full path first
- **Breadth-first search:** shallow, searching one-step paths first
- **Depth-limited (DLS):** modification of DFS
Depth-first search of graph

\[ \text{GDFS}(G) \]
\[ G = (V, E) \]
\[ > \text{Pre: there is a vertex } v \text{ in } V \text{ s.t. } \text{Mark}[v] = 0 \]
\[ \text{count} \leftarrow 0 \]
\[ \text{for each vertex } v \in V \text{ do} \]
\[ \text{if Mark}(v) = 0 \]
\[ \quad \text{vdfs}(v, \text{count}) \]
\[ \text{return Mark} \]
\[ > \text{Post: Vertices are marked in order of} \]
\[ > \text{some DFS traversal} \]

Depth-first search from vertex

\[ \text{vdfs}(v, V) \]
\[ \text{// recursive} \]
\[ \text{count} \leftarrow \text{count} + 1 \]
\[ \text{Mark}[v] \leftarrow \text{count} \]
\[ \text{For each vertex } w \in V \text{ adjacent to } v \text{ do} \]
\[ \text{if Mark}[w] = 0 \]
\[ \quad \text{vdfs}(w, V) \]
\[ > \text{Post: } v \text{'s mark is set} \]

Label each vertex with a number in order visited, exhausting one path before backtracking to new edge.
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 $(a,b), (a,c), (a,d), (b,e), (b,f)$

Breadth-first search

$\text{BFS}(G) \quad G = (V, E)$

\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) \\
\text{bfs}(v) & \\
\text{count} & \leftarrow \text{count} + 1 \\
\text{Mark} \; (v) & \leftarrow \text{count} \\
\text{Queue} & \leftarrow (v) \\
\text{While not empty(queue)} & \text{do} \\
\quad & \text{For each vertex} \; w \in V \; \text{adjacent to} \; \text{front(queue)} \; \text{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} \; \text{queue} \\
\quad & \text{Remove front of} \; \text{queue} \\
\end{align*}
2. State-space search

Subtopic outcome

2.2a Explain goal-based state-space search*

2.2b Perform a goal-driven analysis of a problem with a game tree*

3. Exhaustive search and intractability

- What is the set of all possible solutions to a planning problem like?
- How big is this set for two-step planning solutions? $n$-step?
Algorithm analysis

• **Analysis**: separation into components
• Form of running-time analysis for algorithm $\alpha$, with running time that is big-O or big-$\Theta$ (theta) of $f$, where $f$ is a function of the size of $\alpha$’s input:
  $$T_\alpha(n) = O(f(n))$$
• **Examples:**
  - $T_{\text{Array-append}}(n) = \Theta(1)$
  - $T_{\text{Linear-search}}(n) = \Theta(n)$
  … because appending a new element to an array takes constant time and linear search takes time proportional to the array size

Loops and complexity

• A single loop of $n$ iterations, where each iteration executes $O(1)$ steps, is $O(n)$
• A loop nested to two levels, each with roughly $n$ iterations, where each iteration executes $O(1)$ steps, is $O(n^2)$
• If we start with $n$ items to look at and cut our remaining work in half at each step, then the job will take $O(\log_2 n)$ such steps.
• If our loops are nested to $n$ levels, as in password guessing, then algorithm is $O(2^n)$ — offer job to someone else
Slower part of an algorithm dominates its running time

- **Theorem:**
  \[ (\forall n) \ T_1(n) \in O(g_1(n)) \land T_2(n) \in O(g_2(n)) \Rightarrow \]
  \[ T_1(n) + T_2(n) \in O(\max\{g_1(n), g_2(n)\}) \]

- **Discussion:** The times of two steps of an algorithm, added together, grow as the order of the slower (maximum-time) part

- **Example:** Algorithm that checks for duplicates in an array by bubble-sorting then checking consecutive elements, is \( O(n^2) \)

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
Hard computational problems

- We may 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)

Multiplication rule

- Suppose an operation has $k$ steps
- Suppose steps 1 .. $k$ have $n_1$, … $n_k$ ways to be performed
- Then there are $\prod_{i=1}^{k} (n_i)$ ways to perform the operation
- From this it follows that there are $O(2^k)$ ways to perform it
## 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 plans is of \( k^n \) size; 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.

## 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:**
  1. \( p \land q \land r \)
  2. \( (p \land q) \lor \neg q \)
  3. \( p \land \neg (q \lor \neg q) \)
Polynomial time complexity

- \( P = \bigcup_{k \in \mathbb{N}} \text{DTIME}(n^k) \)

- 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 \( \text{DTIME}(n) \)
  - Sorting an array, \( O(n \log n) \)
  - Searching a BST, \( O(\log n) \)
  - Generating a graph reachability matrix, \( O(n^3) \)

Exponential time

- \( \text{EXPTIME} = \text{DTIME}(2^n) \)

- \( SAT \) seems to be in \( \text{EXPTIME} \) but not \( P \), because it seems that \( O(2^n) \) candidate solutions need to be generated and tested.
The \textit{NP} complexity class

- $\text{NP} = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)$
- That is, \text{NP} is the set of problems decidable in polynomial time by nondeterministic algorithms
- \textit{Intuition:} \text{NP} problems are those for which a particular candidate solution, once found, can be verified in polynomial time
- \textit{Question:} Is $P = \text{NP}$?

\textit{NP} completeness

- For many problems in \text{NP}, e.g., SAT, no polynomial time solution is known
- Problems to which SAT or similar problems are reducible are called \textit{NP-complete}
- \textit{Example:} Traveling Salesperson
- If any of these has a polynomial-time solution, then all do, hence $P = \text{NP}$
- Most researchers think $P \neq \text{NP}$, i.e., \text{NP}-complete problems are believed to be exponential-time
SAT, NPC, and $P = \text{NP}$

- **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:** $P = \text{NP}$ iff $\text{SAT} \in P$

Intractability, NPC, EXPTIME

- Certain problems are thought or known to have only exponential-time, $O(2^n)$, solutions
- *EXPTIME* and *NPC* are the *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
2.3 Apply the definition of intractability to a computational problem

4. Heuristics

- What do you do when you plan?
- Do you consider every contingency?
- How can you please everyone?
### Heuristics

- **Definition:** rules that guide a system to choices in a state-space search that are likely to lead to a satisfactory solution in acceptable time

- **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

### 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, hard to solve
- Minimax algorithm implements DFS assuming optimal opponent move
- Alpha-beta pruning effectively reduces branching factor to its square root
- Cutoff tests evaluate positions 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
2. State-space search

Game trees

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

Heuristics in chess

- Example: value of Queen is 9, rook 5, bishop 3, knight 3, pawn 1
- Seek to force checkmate if possible in a few moves
- Checkmate: a state in which opponent’s king is threatened, with no escape
Informed search with heuristic

- *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 it is useful to know that for state $C$, $\text{cost}(A, B) \leq \text{cost}(A, C) + \text{cost}(C, B)$
- 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 than $x$
- *Intuition*: No two sides of a triangle can sum to more than the length of the third
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

```plaintext
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 Backtrack (X [1 .. i + 1])
Return fail
```

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

\[ \text{path} \leftarrow \lambda. \]
\[ \text{do} \]
\[ \text{if cannot expand any node} \quad \text{return FAIL} \]
\[ v \leftarrow \text{node in tree chosen for expansion} \]
\[ \text{if } v \text{ contains goal state} \]
\[ \quad \text{return } \text{path} + v \]
\[ \text{else} \quad \text{expand } v \]

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

Iterative improvement

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

**Search (S)**

\[
\text{repeat} \\
\quad c \leftarrow \text{random (S)} \\
\text{repeat} \\
\quad \text{changed} \leftarrow \text{false} \\
\quad \text{for each } t \in T \\
\quad \quad c' \leftarrow t(c) \\
\quad \quad \text{if eval}(c') > \text{eval}(c) \\
\quad \quad \quad c \leftarrow c' \\
\quad \quad \quad \text{changed} \leftarrow \text{true} \\
\text{until } \neg \text{changed} \\
\text{return } c
\]

- 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

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
2. State-space search

Knowledge, goals, and rationality

• Levels of a computer system: device, circuit, symbol, … knowledge (Newell, 1982)
• Knowledge: Whatever an agent has that enables it to compute its actions so as to reach goal
• Principle of rationality: An agent will select an action if it has knowledge that the action will lead to a system 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
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

Subtopic outcome

2.4 Explain how heuristics are used to provide adequate solutions to hard search problems*
Summary

• We solve problems by searching a space of possible solutions
• The size of these state spaces is often exponential in the size of the data involved
• This makes most state-space search very hard
• We address this difficulty with rules of thumb based on experience

References
