Search

Artificial Intelligence Course
Chapter 4

Search Problem

- Search problem
  - Find a particular object from a large number of such objects
    (search through many possibilities for one that satisfies
    specified requirements).
- Examples
  - Travel arrangements problem: search for the best possible
    combination of flights, car rentals, and hotels.
  - Robot vehicle routing: search for a route to a given
    destination.
  - Board games (chess): search for a sequence of moves to
    vanquish the opponent.
  - Theorem proving: search for a proof whose last line is the
    formula to prove, given axioms and inference rules.

Search Techniques

- Blind Search
  - no guidance
- Heuristic Search
  - some form of guidance
- Optimization
  - find the best object or at least a very good one.
- Adversary Search
  - search methods for two-person games

4.1 Basic Search Issues

Search methods try to find particular objects in a set of such
objects by enumerating and testing some subsets of those
objects.
- Most search methods proceed by systematically applying
  operators and checking after each transformation whether the
  resulting object is an element of the goal set.
- Search space and operators
- Goal and metric
Search Spaces and Operators

- **Search space**: the set of objects that we are interested in searching among. Examples of objects:
  - Board game: a configuration of pieces
  - Class scheduling: a registration form with zero or more classes filled in.
  - Theorem proving: a proof

- **Operator**: transforms one object into another in a search space. Examples of operators:
  - Board game: a legal move
  - Class scheduling: signing up for an add/d drop course.
  - Theorem proving: a rule of inference, e.g. modus ponens

Goal

- **Goal**: a subset of the set of all objects in the search space.
  - Board game: the set of all configurations of pieces in which you win the game.
  - Class scheduling: the set of all registration forms that allow you to graduate having never taken a class that begins earlier than noon.
  - Theorem proving: the set of all proofs whose last line is the formula to prove.

Metric

- **Metric**: some measure of the distance between two objects in the space or some measure of the value of a given object in the space.
- **Metric is implemented as a comparison function or an evaluation function**

- **Comparison function**: estimates, given two objects, which is closest to the goal or which has greater value.
- **Evaluation function**: estimates, given a single object, the distance to the goal or estimates the object’s value.

Heuristic Algorithms

- **Algorithms that are based on techniques that are supposed to perform well in practice but provide no iron-clad guarantees.**
- **Basic steps of heuristic search algorithms**:
  1. Create a representation for the objects and operators.
  2. Define a metric on the search space.
  3. Design an efficient method for comparing or evaluating objects consistent with the metric.
  4. Design an efficient method for selecting the next object to consider in searching the space.
Appliance Assembly Example

- Work cells: wiring connections, positioning parts, fastening parts.
- Task: arrange the robotic work cells to form an assembly line to perform operations accounting for the dependencies among them.
- Exploiting structure to expedite search
  - If we are clever in exploiting the available useful structure, we may be able to reduce the amount of search considerably or even do away with it altogether.

Two Methods for Representation

- Method 1: Search the permutation space
  - Search space: all permutations of work cells
  - Operator: swap adjacent work cells

- Method 2: Search the sequence space
  - Search space: all sequences of work cells
  - Operator: extend a sequence by adding a cell
4.2 Blind Search

- Tree-structured search space
  - Node $n_i$: an object in the search space
  - Arc: operator that transforms $n_i$ to $n_j$ ($n_i$: parent of $n_j$, $n_j$: child of $n_i$)
  - Outdegree of a node: the number of arcs emanating from the node
  - Branching factor $b$ of a tree: average outdegree of nodes in the tree
  - Depth $d$: the length of the path from the root of the tree to the node corresponding to the goal.

Depth-First Search (DFS)

1. $N :=$ a list of initial nodes
2. If $N = \{\}$, exit and signal failure.
3. $n :=$ first node in $N$; $N := N - \{n\}$
4. If $n$ is a goal node, exit and signal success.
5. Otherwise, $N := \{$children of $n\} + N$; goto step 2.

Depth-First Search

- The number of nodes visited if the goal is on the far left:
  \[ L = d + 1 \]
- The number of nodes visited if the goal is on the far right:
  \[ R = 1 + b + b^2 + \ldots + b^d = \frac{(b^{d+1} - 1)}{(b - 1)} \]
- Average number of nodes visited by DFS is
  \[ \frac{(L + R)}{2} = \frac{b^{d+1} + db + b - d - 2}{2(b - 1)} \]
Storage Requirement of DFS

- At each node $n$, DFS has to keep track of the root node and the children of each node in the path from the root to $n$ excepting $n$.
- In the worst case: $d(b - 1) + 1$
- DFS is asymptotically optimal in its use of space:
  - the amount of space required by DFS is within an additive or multiplicative factor of the minimum amount of space required for the task.

Breadth-First Search (BFS)

1. $N := \text{a list of initial nodes}$
2. If $N = \emptyset$, exit and signal failure
3. $n := \text{first node in } N$; $N := N - \{n\}$
4. If $n$ is a goal node, exit and signal success.
5. Otherwise, $N := N + \{\text{children of } n\}$; goto step 2.

Remark: BFS differs from DFS in step 5.

Time Complexity of BFS

- The sum of all the internal nodes visited at depth $d$:
  \[ A = 1 + b + b^2 + \ldots + b^{d-1} = \frac{(b^d - 1)}{(b - 1)} \]
- Average number of nodes visited at depth $d$:
  \[ B = \frac{(b^d + 1)}{2} \]
- Average number of nodes visited by BFS:
  \[ A + B = \frac{b^{d+1} + b^d + b - 3}{2(b - 1)} \]
- BFS is computationally more expensive in time than DFS by a factor of approximately $(b+1)/b$.
- Remark: $(b+1)/b$ is approximately 1 for large $b$. 

BFS and DFS in Comparison

- BFS has to keep track of the entire set of unexplored nodes, which is roughly the set of all nodes at depth $d$; it uses space proportional to $b^d$ (bigger than $d(b - 1) + 1$ of DFS).
- BFS is guaranteed; BFS will find a goal node if such a node exists even if the tree has infinite depth.
- With an infinite tree, DFS may not find a node satisfying the goal and, worse, it may not terminate.
- Neither BFS nor DFS is asymptotically optimal in both time and space.

Iterative-Deepening Search (IDS)

- An asymptotically optimal algorithm.
- Iterative deepening search works by making repeated DFS to a fixed depth, increasing the depth on each search until a goal node is found.
- IDS revisits the same nodes in the search tree many times.
- But, the cost of these repeated visits is negligible from an asymptotic perspective.

IDS Compared with DFS and BFS

- IDS is an asymptotically optimal blind search procedure.
- Space requirements for IDS are the same as for DFS.
- Time requirements for IDS are roughly the same as BFS.
- IDS guaranteed to find a goal node if one exists.
- Although IDS is asymptotically optimal, there are plenty of occasions in which we might prefer either DFS or BFS.
- DFS is preferred for searching a finite tree-structured search space with goal nodes at the leaves of the tree.
- BFS is preferred in cases with a small branching factor, operators that are expensive to apply, and goal nodes expected at a reasonable depth.
Searching in Graphs

- Many search spaces correspond to graphs instead of trees.
- In searching a graph, it is possible to arrive at the same node using different paths corresponding to different sequences of operators.
- To avoid considering nodes more than once, we keep a list of the nodes already visited.
- Or we can transform a graph-structured search space into tree-structured space.

4.3 Heuristic Search

- A **heuristic** is any rule or method that provides guidance in decision making: A useful heuristic need not always improve decision making, but should improve decision making more often than not.
  - E.g.: Choosing to stand in the shortest checkout line at the supermarket.
- **Heuristic search**: we take advantages of information about the structure of the space.
- We assume a **metric** on objects in the search space that allows us to estimate the distance from a node to a goal.
- We use this metric to compare two nodes in determining which nodes to explore next during search.

Best-First Search

- Best-first search attempts to find a goal node quickly by searching nodes that are estimated to be close to a goal.
  1. \( N := \) a list of initial nodes
  2. If \( N = \) \{\}, exit and signal failure.
  3. \( n := \) first node in \( N \); \( N := N - \{n\} \)
  4. If \( n \) is a goal node, exit and signal success.
  5. Otherwise, \( N := N + \{\text{children of } n\} \);
     \( N := \text{SortByEstimatedDistanceFromGoal}(N) \)
     goto step 2.

Best-First Search: An Example

[Diagram showing a graph with nodes A, B, C, D, I, J, N and distances from A to each node labelled 1, 6, 7, 2, 9, 3, 2, and 4.]
Evaluation Functions

- Best-first search relies on the use of an *evaluation function* to compare objects in the search space.
- The time and space complexity of best-first search depends on how well the evaluation function performs.
- A search algorithm is said to be **heuristically adequate** if it always finds a goal that is nearest to the start node.

Admissible Evaluation Functions

- Consider the following form of evaluation function
  \[ e(n) = g(n) + h'(n) \]
  
  - \( g(n) \): the distance from the root to \( n \)
  - \( h'(n) \): estimate of the distance from \( n \) to the nearest goal
- If \( h'(n) = 0 \), best-first search reduces to BFS.
- An evaluation function, \( h' \), is **admissible** if \( h'(n) \) is less than or equal to \( h(n) \) for all \( n \), where \( h(n) \) is the actual distance to the nearest goal.
A* Algorithm

- In other words, an evaluation function $h'(n)$ is admissible if it underestimates $h(n)$, the actual distance to the nearest goal.

- If $h'$ is admissible, then the resulting variant of best-first search is heuristically adequate and called A*.

Admissible Evaluation Function: An Example (1/2)

- A robot path planning problem
- Searching for path in a grid environment
  - $S$: robot’s location, $G$: destination
  - The Manhattan distance from $S$ to $G$ is four.
  - The length of the shortest unobstructed path is six.

Searching for Path in a Grid Environment

The Manhattan distance provides an admissible heuristic evaluation function for this path planning problem.

(Manhattan distance) $\leq$ (length of the shortest unobstructed path)

Admissible Evaluation Function: An Example (2/2)

Figure 4.12 Searching for paths in a grid environment
Searching for Paths with Different Metrics

- An evaluation function imposes structure on the objects of a search space.
- Three-dimensional surface determined using the **Manhattan distance** to the goal as a metric.
  - Each node is labeled with the additive inverse of the Manhattan distance to the goal. The surface is determined by these distances.
  - Different metrics and hence evaluation functions result in different surfaces.

---

Three-dimensional surface determined using the shortest distance to the goal as a metric for the grid environment.

- The figure shows the surface when the metric is the shortest distance to the goal **accounting for obstructions**.
4.4 Optimization and Search

- Optimization: search for the highest (or lowest) value object in the search space.
- Search techniques can be used to address optimization problems that involve finding an object that maximizes (or minimizes) a particular function.
- Search Methods for Optimization
  - Hill-climbing search
  - Gradient search
  - Simulated annealing
  - Genetic algorithms

Hill-Climbing Search

- A variant of best-first search that requires only a constant amount of storage.
- Does not keep track of any previously encountered nodes except the best encountered so far.
- Procedure:
  1. \( n := \) initial node.
  2. If \( value(n) > value(\text{any children of } n) \), then exit and return \( n \).
  3. Otherwise, \( n := \) highest value of \( n \); goto step 2.
- Hill-climbing search always finds a local optimum.

Local Maxima and Minima

- A local maximum in a search space is an object s.t. all the nearby objects have lower value.
- A local minimum is an object such that no other objects in the search space has higher value.
- Hill-climbing search finds a local maximum (not necessarily a global maximum).

Figure 4.15: Grid environment with a cul de sac

neither explores the same node twice will find \( G \) starting from \( S \), but such an algorithm is impractical for any real optimization problem.
Gradient Search (1/2)

- Gradient search is a hill-climbing method for finding the minima and maxima of continuous, differentiable functions.
- By taking many small steps in the direction indicated by gradient, we are guaranteed to find a local maximum.
- Objective: Find a real value $x$ maximizing an evaluation function $f(x)$

Gradient Search (2/2)

- Gradient search algorithm:
  ```
  do
  
  $x := x + \beta \frac{df(x)}{dx}$
  
  until
  
  $-\varepsilon \leq \frac{df(x)}{dx} \leq +\varepsilon$
  
  $\beta$: step rate (small positive constant)
  $\varepsilon$: stopping criterion (small positive constant)
  ```

Simulated Annealing (1/2)

- Motivation for simulated annealing.
Simulated Annealing (2/2)

- To find a global maximum, *randomization* is necessary.
- Most of the time we take *small* steps in the gradient direction, but *occasionally* we take *large* steps in the gradient direction or in some other direction.
- **Simulated annealing** was inspired by a technique used by metallurgists.
- In annealing metals, the materials are *heated and cooled repeatedly* to achieve a particular crystalline structure.
- Annealing can be used to obtain metals that are malleable and less likely to break when subjected to shock.

Simulated Annealing Procedure

- Procedure SimulatedAnnealing():
  1. Start with $A :=$ initial configuration.
  2. Generate an alternative configuration $A'$
  3. Compute the probability ratio $p = \frac{P_t(A')}{P_t(A)}$
  4. If $(p > 1)$, then $A := A'$
     else if $(p \leq 1)$, then $A := A'$ with probability $p$.
  5. Goto step 2.
- Simulated annealing has a *temperature* parameter $T$ that is adjusted during search: $T_0 > T_1 > T_2 > \ldots > T_N$
- High $T$: *large* steps more probable.
- Low $T$: *small* steps more probable.

Genetic Algorithms

- Genetic algorithms search by simulating evolution.
- *Natural selection* (survival of the fittest) and genetics.
- A population of individuals live in a continent.
- They mate each other to produce offspring.
- Each individual has a fitness value.
- **Problem Solving by a Genetic Algorithm**
  - **Individual**: an object (possible solution) in a search space
  - **Fitness value**: evaluation function
  - **Population**: a set of possible solutions
  - Search is based on a population of individuals

A Simple Genetic Algorithm

1. Initialize the population $G = \{A_1, \ldots, A_M\}$
2. Evaluate fitness $f(A_i)$ for $A_i$, $i = 1, \ldots, M$ in $G$
3. Create a probability distribution
   \[ P(A) = \frac{f(A)}{\sum_{A' \in G} f(A')} \]
4. Create the next generation $G'$ by using *genetic operators* (see the next page).
5. If an acceptable solution found in $G'$, then: **exit** and return the best individual as the solution; otherwise: $G = G'$
**Genetic Operations (Fig 4.18)**

- **Step 4** (in more detail) Using genetic operators for creating new individuals:
  - \( G' = \{ \} \).
  - For \( I = 1 \) to \( |G|/2 \)
    - a. (Reproduction) Randomly choose two parents \( A \) and \( A' \) according to \( P(x) \).
    - b. (Crossover) Randomly swap bits in \( A \) and \( A' \).
    - c. (Mutation) Randomly flip a small number of bits.
    - d. Call the resulting offspring \( B \) and \( B' \).
  - \( G' = G' + \{ B, B' \} \)

**Application to Vehicle Routing**

- **Vehicle routing problem**: find the shortest route for a fleet of delivery trucks that visits all the company's customers.
- **Vehicle routing problem** is a kind of traveling salesman problem (**TSP**): Visit all of \( n \) cities **exactly once** and end up back at the starting city. The objective is to minimize the total distance traveled.
- **Tour**: a permutation of the \( n \) cities, indicating the order in which to visit the cities.
- **Chromosomal representation** of a tour: examples
  - \( (1 \, 2 \, 3 \, 4 \, 5 \, 6 \, 7 \, 8) \)
  - \( (1 \, 3 \, 2 \, 7 \, 5 \, 6 \, 8) \)
  - \( (2 \, 3 \, 6 \, 7 \, 5 \, 1 \, 8 \, 4) \)

**Mutation and Crossover**

- **Mutation**: swap the cities at two sites.
  - \( (1 \, 3 \, 4 \, 2 \, 7 \, 5 \, 6 \, 8) \) \( \Rightarrow \) \( (6 \, 3 \, 4 \, 2 \, 7 \, 1 \, 8) \)
- **Crossover**: exchange substrings between two chromosomes
  - Example of partially matched crossover:
    - \( (134|275|68) \) \( \Rightarrow \) \( (134|751|68) \) \( \Rightarrow \) \( (234|751|68) \)
    - \( (236|751|84) \) \( \Rightarrow \) \( (236|275|84) \) \( \Rightarrow \) \( (136|275|84) \)
    - exchange map
      - \( 1 \Rightarrow 5 \Rightarrow 7 \Rightarrow 2 \)
4.5 Adversary Search

- Search techniques for two-person games in which the two players alternate moves.
- Both players have perfect information about the location of the pieces on the board. (In contrast, many card games involve imperfect information.)
- Search in a two-person game of perfect information involves considering various sequences of alternating moves by the two players to assess the consequences of the immediately available options for moving.

Game Tree

- A game tree is a tree consisting of nodes that represent options for the players to move in a two-person game.
- The nodes alternate between options for the two players (maximizer and minimizer).
- Nodes: states of the game (config. of playing board)
- Root node: the current state of the game
- Terminal nodes: +1 (win of maximizer) or -1 (win of minimizer)

Simple Game Tree (Fig 4.20)

Minimax Search

1. \( N := \{ m \} \).
2. \( n := \text{first}(N) \).
3. If not assigned \( n \) and \( n \) assigned a value \( v \), then exit by returning \( v \).
4. If not assigned \( n \), then \( N := N - \{ n \} \).
5. If not assigned \( n \) and terminal \( n \), then assign \( n \) the value 1, -1, or 0 depending on whether it is a win for the max, a win for the min, or a draw.
6. If not assigned \( n \) and all its children assigned values, then if \( n \) a max node, assign \( n \) the maximum of the values of its children; otherwise, assign \( n \) the minimum of the values of its children.
7. If not assigned \( n \) and all its children have not been assigned values, then \( N := \{ n \} + N \).
8. Return to step 2.
Assigning Values to a Game Tree (Fig. 4.21)

An Evaluation Function for Chess

- To determine which move to make next without searching the entire game tree, we need an evaluation function, $e(n)$, which when applied to a node $n$ returns an estimate of its value.

$$e(n) = \frac{w(n) - b(n)}{w(n) + b(n)}$$

- $w(n)$: sum of the values of all the white pieces in the state $n$
- $b(n)$: sum of the values of all the black pieces

Alpha-Beta Search

- Although it is generally difficult to determine exactly how far ahead to search, there are some cases in which, given the nodes that are already evaluated, there is no use in extending the search at other nodes (pruning).

Example 1
Example 2

![Tree Diagram]

1. \( v := \text{value}(n) \)
2. \( m := \text{parent}(n), u := \text{value}(m) \)
3. If \( \text{maximizer}(m) \), then \( \text{value}(m) := \text{maximum}\{u, v\} \)
4. If \( \text{minimizer}(m) \), then \( \text{value}(m) := \text{minimum}\{u, v\} \)
5. If \( \text{root}(m) \) or \( \text{value}(m) \) unchanged, then \textbf{quit};
   otherwise, \textbf{back up} the value \( m \).

An Example of Backing Up the Value:

![Tree Diagram]

Algorithm for Alpha-Beta Search

1. \( N := \{m\} \)
2. \( n := \text{first}(N) \)
3. If \( n = m \) and \( n \) assigned a value, then \textbf{exit} returning this value.
4. Try to \textbf{prune} as follows.
   If \( \text{maximizer}(n) \), \( v = \text{min}_v\{\text{siblings}(n)\} \),
   \( u = \text{max}_u\{\text{siblings}(\text{min}_a\{\text{ancestors}(n)\})\} \).
   If \( u \geq v \), then \textbf{remove} \( n \) and its siblings and any successors of \( n \) and its siblings from \( N \).
   If \( m \) is a minimizing node, then proceed similarly switching \( \text{min} \) for \( \text{max} \), \( \text{max} \) for \( \text{min} \), and \( \leq \) for \( \geq \).
Algorithm for Alpha-Beta Search

5. If \( n \) cannot be pruned, then if \( n \) is a terminal node or we decide not to expand \( n \), assign \( n \) the value determined by the evaluation function and back up the value at \( n \).
6. Otherwise, remove \( n \) from \( N \), \( N := \{ \text{children}(n) \} + N \), assign the children initial values of \(-\infty\) for max nodes and \(+\infty\) for min nodes.
7. Return to step 2.

4.6 Indexing in Discrimination Trees

• Some applications (e.g. answering database queries) require repeatedly searching through the same set of objects stored in memory.
• The effort required in these searches can often be reduced by appropriately adding structure to the set of objects.

- Storing and retrieving predicate calculus formulas
  (1) Discrimination networks
  (2) Decision Trees

Discrimination Trees

- A discrimination tree or, more generally, a discrimination network is such a data structure used for storing and retrieving large numbers of symbolic objects.
- Basic idea: recursively partition a set of objects where each partition divides the set into subsets
- Examples:
  (1) employees with (salary \( \leq \) $50,000) and those with (salary > $50,000).
  (2) partition salaries into the ranges ($0 - $10,000), ($10,000 - $20,000), ($20,000 - $50,000), ($50,000 - $70,000), and (> $70,000).

A Discrimination Tree for the PC Formulas

- A Set of PC Formulas

  (on block1 block2)
  (on block2 (floor room17))
  (instance block block1) //교과서 정정
  (instance block block2)
  (on (? x) (? y))
  (on (? z) (? z))
Decision Trees

- Decision trees are closely related to discrimination trees.
- A **decision tree** is a representation for a particular sort of procedure.
- **Nonterminal** nodes: questions
- **Arcs**: answers
- **Terminals**: data or decision

Decision procedure: Starting at the root, ask the question posed at a node and traverse the arc corresponding to the answer.

- An example decision tree for deciding what to do when your car fails to start:

  - Does the starter work?
  - Is there gas in the tank?
    - Yes
      - No
    - Yes
  - Do the lights work?
    - Yes
      - No
    - Yes
      - No

Actions:
- Call an expert
- Get some gas
- Call an expert
- Recharge the battery