Outline

1. Reminder
2. Artificial Intelligence
3. Selected Search Algorithms from AI
We have demonstrated with the packing problem how to obtain fast solutions with mathematical insight and theorems.

We looked at greedy algorithms.

We summarized algorithmic patterns.

We looked at complexity theory.

Next we look at some interesting algorithms from Artificial Intelligence (AI)
What is Artificial Intelligence?
And how is this useful for Astronomy?

- In general, AI is about (computer) systems that can think/act “like humans”.
- A more humble (but still ambitious) goal is computer systems which can think/act rationally.
- An important aspect of AI is to perceive the world at a series of hierarchical levels (classification of information).
- To what extent can we replace astronomers with computers?
- E.g. reducing large amount of data, making intelligent decisions about observations (where, what, when, take, biases, flats, etc), finding “interesting” objects, reading and writing papers, etc.
Turing Test

- Turing (of the Turing machine fame) have designed this test.
- Can a human interrogator tell the difference between a machine and a computer?
Can machines think?

- The Turing test shifts the question: can machines behave intelligently?
- Turing originated this subject in 1950.
- Suggested major components of AI: knowledge, reasoning, language understanding, learning.
- Problem: the Turing test is not reproducible, difficult to analyse mathematically; but does express the essence of AI.
AI and Computer Science

- AI is considered part of computer science, however, it has unusually strong connection with cognitive science, neuroscience, psychology, philosophy, linguistics, logic/mathematics, etc.

- For the astronomer it has a lot to offer, especially in terms machine learning, classification, perception, planning, visualization, etc.

- Computer science emphasizes solving specific problems very efficiently

- AI emphasizes solving very general classes of problems

- We will get only taste by looking into some which can be considered variants/refinements of backtracking.
Uninformed Searches

- Reminder: in backtracking, we try all possible solutions for a problem, by expanding a “search tree”.
- In the algorithm we have kept an exploreList; in AI it is called the fringe.
- Depending what data structure is used for the fringe
  - LIFO stack → depth first search
  - FIFO queue → breadth first search
  - priority queue ordered by path cost, lowest first → uniform cost search
  - depth limited search: DFS with depth limit
  - iterative deepening search (IDS): gradually increasing the depth limit.
Best First Search

- Informed search algorithm
- use a priority queue (a queue ordered by “desirability”), and always expand the most desirable element from the end of the queue.
- We defined desirability using what we know about the problem.
- What proper choice the search will be much more efficient then simple backtracking/uninformed search.
Greedy Search
The simplest variant of Best First

- Let $h(n)$ be an *heuristic*, an estimate of the cost to the closest goal.
- Set up a priority queue as above, and use $h(n)$ for the “desirability”.
- E.g. Finding the best route on a map between two cities. The cities and roads connecting them can be represented as a weighted graph. The heuristic for the greedy search could be the straight line distance of each city to the goal.
- Greedy search is not complete (can get stuck in loops), has $O(b^m)$ time and space complexity, and suboptimal. A good heuristic gives dramatic improvement.
A*Search
An improvement over greedy search

- We use a better heuristic: $f(n) = g(n) + h(n)$ the sum of actual cost to the node $n$ plus the estimated cost from $n$ to the goal.
- **admissible** heuristic: $h(n) \leq h^*(n)$, where, $h^*(n)$ is the true cost from $n$. We also require that $h \geq 0$, thus $h(G) = 0$ at the goal.
- For an admissible heuristic, the A*search is optimal.
- $h(n) = 0$ is admissible (corresponds to Dijkstra’s algorithm)
A* is optimal (sketch of proof)

Suppose some suboptimal goal $G_2$ has been generated and is in the queue. Let $n$ be an unexpanded node on a shortest path to an optimal goal $G_1$.

\[
f(G_2) = g(G_2) \quad \text{since } h(G_2) = 0
\]

\[
> g(G_1) \quad \text{since } G_2 \text{ is suboptimal}
\]

\[
\geq f(n) \quad \text{since } h \text{ is admissible}
\]

Since $f(G_2) > f(n)$, A* will never select $G_2$ for expansion.

QED.
Admissible Heuristics
E.g.: 8-puzzle

- \( h_1(n) \) = number of misplaced tiles
- \( h_2(n) \) = total Manhattan distance
- \( h_1(\text{Start}) = 6 \)
- \( h_2(\text{Start}) = 4 + 0 + 3 + 3 + 1 + 0 + 2 + 1 = 14 \)
Typical Search Costs

- IDS = 3,473,941 nodes
- $A^*(h_1) = 539$ nodes
- $A^*(h_2) = 113$ nodes
- IDS $\approx 54,000,000,000$ nodes
- $A^*(h_1) = 39,135$ nodes
- $A^*(h_2) = 1,641$ nodes

- If $h_2(n) \geq h_1(n)$ for all $n$ (both admissible)
- $\rightarrow h_2$ dominates $h_1$ and is better for search.
How to Find Heuristics?
Look at relaxed problems and possible combine

- The exact cost of a *relaxed* version of the problem will create heuristic.
- If tiles could move anywhere/any adjacent square, we would get $h_1/h_2$.
- Given any admissible heuristics $h_a$, $h_b$,
  $h(n) = \max(h_a(n), h_b(n))$ is also admissible and dominates $h_a, h_b$
- Thus using $h$ will result in a more efficient search.
Best First Searches: Summary

- Heuristic functions estimate costs of shortest paths
- Good heuristics can dramatically reduce search cost
- Greedy best-first search expands lowest $h$ – incomplete and not always optimal
- A* search expands lowest $g + h$
  - it is complete and optimal
  - also optimally efficient (up to tie-breaks, for forward search)
- Admissible heuristics can be derived from exact solution of relaxed problems
In many optimization problems, \textit{path} is irrelevant;

The goal state itself is the solution.

Then state space = set of “complete” configurations;

Find \textit{optimal} configuration, e.g., TSP or, find configuration satisfying constraints, e.g., timetable.

In such cases, can use iterative improvement algorithms; keep a single “current” state, try to improve it.

They are constant space complexity.
E.g. Traveling Sales Representative Problem

- Start with any complete tour
- Perform pairwise exchanges
- This is a famous NP-complete problem, yet
- Variants of this approach get within 1% of the optimal solution very quickly with thousands of cities
**Hill Climbing (gradient ascent)**

- Simply take the highest valued neighbour
- Random-restart: overcomes local maxima
- Random sideways moves escape from shoulders but loop on flat maxima
Simulated Annealing
A physically motivated improvement on hill climbing

- Idea: include bad moves to escape local minima, but gradually decrease their size and frequency (=cooling)
- Metropolis-algorithm: accept any move, which decreases the energy; also accept moves which increase the energy with probability $e^{-\beta \Delta E}$.
- If $T = 1/\beta$ is decreased slow enough, it will always reach the best state, since the probability of stepping away $p \to 0$ for small $T$.
- This is a widely used randomized algorithm for solving otherwise difficult problems.
- Note that in CS often “energy maximum” is sought, the exact opposite of the physical motivation.
Local beam search

- **Idea**: keep $k$ states instead of 1; choose top $k$ of all their successors
- **Not the same as $k$ searches run in parallel**: Searches that find good states recruit other searches to join them
- **Problem**: quite often, all $k$ states end up on same local hill
- **Improvement**: choose $k$ successors randomly, biased towards good ones.
- **Close analogy to natural selection. Next**: genetic algorithms.
Genetic algorithms
Conceptual outline

- stochastic local beam search
- generate successors from *pairs* of states
Genetic algorithms: Example
N-queens problem

- GAs require states encoded as strings
- Crossover helps iff substrings are meaningful components
- NB: GAs ≠ evolution: real genes encode replication machinery.
Note on continuous state spaces

- The previous algorithms are applicable to many problems with continuous space states.
- Typically we discretize the configuration space, and consider steps in each direction.
- Gradient methods compute $\nabla f$ to increase/reduce $f$, e.g., by $x \leftarrow x + \alpha \nabla f(x)$.
- Sometimes can solve for $\nabla f(x) = 0$ exactly.
- The Newton–Raphson method iterates $x \leftarrow x - H_f^{-1}(x) \nabla f(x)$ where $H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$.
Summary

- Artificial Intelligence has potential applications in astronomy.
- It is producing algorithms and methods which solve very general problems.
- We looked at a series of interesting search algorithms which are at the boundary of AI and CS.
- Other possible applications include machine learning, classification.
Homework # 9

(E1) Solve the $N$-puzzle problem (the $N \times N$ generalization of the 8-puzzle) using A* search. Using (at least 3) different heuristics, count the number of nodes searched. The programs should have an option to display the steps of the search and the steps leading to a solution.

(P1) Solve the $N$-queens problem problem with any algorithm mentioned in class. The program should display the solution it arrives when finished. (Extra point)