6601 - Assignment 1: Search

Daniel Kohlsdorf, Titilayo Craig, Thad Starner

WARMUP (20%)

As a warmup you will explore some theoretical properties of search algorithms with exercises from your text book:

Assignment 3.18: Describe a state space in which iterative deepening search performs much worse than depth-first search (for example, $O(n^2)$ vs $O(n)$).

Assignment 3.21: Prove each of the following statements, or give a counterexample:

a. Breadth first search is a special case of uniform-cost search
b. Depth first search is a special case of best-first tree search
c. Uniform-cost search is a special case of A* search.

PRACTICE (20%)

The following two questions will help you to gain an insight into scaling large search problems using data structures.

1) **Queues**: Figure 3.14 in your textbook shows the uniform cost search algorithm. Argue in terms of computational complexity why a priority queue such as a fibonacci heap is superior to searching the max in an unordered queue or to sorting. HINT: you have to use both INSERT and POP for your argument. You do not need to prove your result but construct a clear argument.

2) **BDD**: If you search large spaces, keeping track of nodes you visited (explored queue in the algorithm) already might be expensive in terms of memory. How could you use binary decision diagrams (BDDs) to keep track of visited nodes with the goal to save memory? How do you you have to represent states?

IMPLEMENTATION (60%)

In the three city problem you want to visit three cities. The order of your visit does not matter. The goal is to find the shortest path through these three cities. In this assignment we will search a real street map of Atlanta. In the attachment you will find a map in open street map format (osm) and a script to parse it into a street graph. Furthermore, you will find a breadth first search

implementation. You can start coding your own searches right away. However, if you do not like Python and you want to do it on your own, go ahead. We accept solutions in: C, C++, Java, Python, Ruby, Lisp, Prolog. If you want to use something else talk to us first. The book's webpage contains helpful code snippets for multiple languages, too.

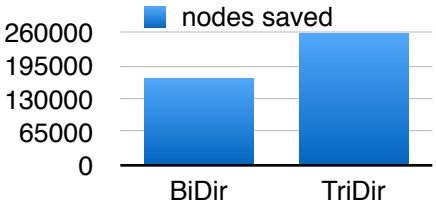
1) **Implement Search algorithms:**

- **uniform cost search**
- **bidirectional search**
- **tridirectional search.**

Tridirectional search is similar to the bidirectional algorithm, except that you initialize three simultaneous searches instead of two. **Implement all three algorithms**. In the report shortly **describe each of the three implementations**. Also include in the report if the algorithm is **complete and / or optimal**. Solve the **three city problem with EACH of the three algorithms.**

Figure 1: Tridirectional Search

2) Evaluation: With your implementations compare the number of nodes touched when you perform three separate uniform cost searches vs the number of nodes expanded with the bidirectional search method and tridirectional search method. Run a search for multiple starting conditions (pick three positions at random and run all three search methods, repeat this a **100** times). Count the number of nodes expanded for all three search methods. Write a paragraph about your results and explain your observations as well as your experimental setup. Also discuss the relation to the theoretical bounds using O notation. Report the number of nodes touched for bidirectional and tridirectional search and report the difference vs uniform cost search. Graph the number of nodes saved on average for bidirectional and tridirectional search compared to uniform cost search. We expect a plot similar to the one shown in blue.



LATE

Repeat your experiment comparing triddirectional search and A*. Therefore you have to implement A* too. Write a paragraph about your results and explain your observations as well as your experimental setup.