Informed Search Last Thoughts

David Kauchak
CS51A – Spring 2022

Admin

Assignment 9

Assignment 10

Midterm 2 next Monday
- Dictionaries (2/21) through today
- “cheat” sheet – two pages
- Sample problems posted

Informal search

Order to visit based on some knowledge of the world that estimates how “good” a state is
- $h(n)$ is called an evaluation function

Best-first search
- Rank to visit based on $h(n)$
- Take the most desirable state in to_visit first
- Different approaches depending on how we define $h(n)$

from: Claremont to: Rowland Heights

We’d like to bias search towards the actual solution
Heuristic function: $h(n)$

An estimate of how close the node is to a goal

Uses domain-specific knowledge!

Examples
- Map path finding?
  - straight-line distance from the node to the goal (“as the crow flies”)
- 8-puzzle?
  - how many tiles are out of place
  - sum of the “distances” of the out of place tiles
- Foxes and Chickens?
  - number of people on the starting bank

Two heuristics

Which state is better?

Two heuristics

What is the “distance” of the tiles that are out of place?

Two heuristics

6
Next states?

Which would DFS choose

Completely depends on how next states are generated.
Not an "intelligent" decision!

Which would you do?

Best first search: out of place tiles?
Best first search: distance of tiles?

Which next for best first search?
Informed search algorithms

Best first search is called an “informed” search algorithm.

Why wouldn’t we always use an informed algorithm?
- Coming up with good heuristics can be hard for some problems
- There is computational overhead (both in calculating the heuristic and in keeping track of the next “best” state)

Informed search algorithms

Any other problems/concerns about best first search?
- Only as good as the heuristic function

Informed search algorithms

Any other problems/concerns about best first search?
- What would the search do?
Informed search algorithms

Any other problems/concerns about best first search?

- Only as good as the heuristic function

Best first search using distance as the crow flies as heuristic

What is the problem?

Best first search is called an “informed” search algorithm

There are many other informed search algorithms:

- A* search (and variants)
- Theta*
- Beam search

Sudoku

Fill in the grid with the numbers 1-9

- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)
Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

How can we pose this as a search problem?
- State: 9 x 9 grid with 1-9 or empty
- Start state:
- Goal state:
- State space/transitions

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints
Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

How many next states?
What are they?

1, 6, 7, 9
Sudoku

Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

What are the next states?

2, 6, 7, 8, 9
Sudoku

Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

Generate next states:
- pick an open entry
- try all possible numbers that meet constraints

Now what?
Try another branch, i.e. go back to a place where we had a decision and try a different one
Sudoku

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3</td>
<td></td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td>1</td>
<td>2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fill in the grid with the numbers 1-9
- each row has 1-9 (without repetition)
- each column has 1-9 (without repetition)
- each quadrant has 1-9 (without repetition)

DFS and BFS will choose entries (and numbers within those entries) randomly

Pick the entry that is **MOST** constrained

People often try and find entries where only one option exists and only fill it in that way (very little search)

Best first Sudoku search

Heuristics for best first search?

Representing the Sudoku board

SudokuEntry: Class to keep track of an entry on the Sudoku board

- Keep track of whether or not a value has been placed in that entry (fixed)
- If one hasn't, keep track of what numbers are still valid
Representing the Sudoku board

- Board is a matrix (list of lists)
- Each entry is a SudokuEntry:
  - “fixed” if it has a number place
  - if not, then keeps a list of values still available

Which is the most constrained (of the ones above)?

Remove 1 from all entries in the quadrant

What other parts of the board need to be updated?

What would the state look like if we add pick 1?

Board will consist of 81 SudokuEntry s
- Each quadrant will have 9 entries.
### Representing the Sudoku board

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>3</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

- Board is a matrix (list of lists)
- Each entry is a SudokuEntry:
  - "fixed" if it has a number place
  - if not, then keeps a list of values still available

Remove 1 from all entries in the quadrant

Remove 1 from all entries in the same column

Remove 1 from all entries in the same row