Computing Science (CMPUT) 455

Search, Knowledge, and Simulations

James Wright

Department of Computing Science
University of Alberta
james.wright@ualberta.ca

Fall 2021

Part II

Search and Knowledge

455 Today - Lecture 7

Today's Topics:

- Introduction to Search
- Blind search and random sampling
- Heuristic search and heuristic functions
- Examples of heuristics

Coursework

- Quiz 4 (state spaces and sequential decisions)
- Read Greenemeyer, 20 years after Deep Blue
- Lecture 7 Activities
- Coming up:
  - Assignment 2 - GoMoku endgame solver - preview on Tuesday

Example Code

- Sample Python codes for search and minimax

Introduction - Search in Computing Science

Overview of Part Two - Search and Knowledge

- Next five lectures (7-11)
- Very quick review of search in general
- Blind vs heuristic search, treasure hunt demo
- Search for two player games
  - Solving games by search
  - Minimax and Alphabeta
  - Some search enhancements
- Knowledge for heuristic search
  - Evaluation function
  - Move ordering
  - Reducing branching factor and depth of search

Outline of Search Topic

- Introduction to Search
- Brief overview of some major types of search
- Blind search, heuristic search
- Graph search strategies, depth-first, breadth-first, best-first
- Single agent, two player, multiplayer
- Some sample problems and search-based solutions

Search in Computing Science

- Search is a very large part of Computing Science
- Many important kinds of search algorithms
- Use search to *lookup* existing information
- Use search to *discover* new information
- Information may *change during search* (think internet)
- Data may be stored in a *structured* or *unstructured* way
- The topic of search is very broad and varied

Examples of Search in CS

- Linear search in an unsorted array or a list
- Binary search in a sorted array
- Search a tree data structure
- Search a database using SQL queries
- Search a pattern in a (large) text string
- Search the minimum or maximum of a mathematical function $y = f(x)$
- Search the minimum or maximum of a high-dimensional mathematical function
- Search the internet using a search engine
- Search for suspects in a video stream
- Many more. Wikipedia has over 100 entries in category "Search algorithms"

State Space Search

- In this course, we only look at search in a given, known state space
- Often the state space is given but only *implicitly*, in a *generative model*
  - We create (generate) the states by a forward search process from a start state
  - Each *reachable* state is reached by a valid action sequence from the start state
  - Examples: Go, chess, ...
- Sometimes we can understand the whole state space, even if it is huge
- Examples: b^d model, DAG model when we know the states at each depth, and the actions
- We can *enumerate* all states in smaller state spaces, e.g. TicTacToe

Single Agent Search vs Two Player Search

- We study mostly two player games in this course
- In many search problems there is only a single player (also called an agent)
- Agent can choose every action, no opponent
- Simpler setting than two players
- Some concepts are very similar, some are very different
- Some optional extra slides in the resources
- Not discussed in class, not on exams or quizzes

Blind Search

Blind Search

- One important distinction is
  blind search vs *heuristic* search
- Blind search: no extra information that helps us search
- We assume only that we recognize *goal states*
  when we visit them
- Popular blind search algorithms on graphs
  (covered e.g. in Cmput 204):
  - Depth-first search (dfs)
  - Breadth-first search (bfs)
  - Optional activity: review related resources,
    such as slides on *Blind Search Extras*

Example: Blind Treasure Search

- Given: a state space with a single goal state which contains the treasure
- All states are equally likely to be the goal
- Claim: we can expect to visit about half of all states before hitting the goal
  - No matter which blind search algorithm we use . . .
  - No matter which (connected) type of graph the state space is, e.g. tree, DAG, DCG
- Proof: next slide

Analysis of Blind Search

- State space with a total of n states
- Exactly one goal state
- States are visited in some order $s_0, s_1, ...s_{n-1}$
- All states s_i are equally likely to be the goal state
- Probability that s_i is goal is $1/n$
- To reach s_i, we visit a total of $i + 1$ states
- Expected number of visits to reach goal:

$$\frac{1}{n}(1 + 2 + ...n) = \frac{1}{n}\frac{n(n+1)}{2} = \frac{n+1}{2} \approx \frac{n}{2}$$

Comments

- The shape of the state space did not matter here
- E.g. b^d model, some other tree, DAG, even cyclic graph
- We do assume that each node is only visited once
- The details of the search algorithm
  also did not matter!

More Comments

- If we have some information about where the goal is
  - We can use that information in a heuristic
  - It is no longer blind search
- In practice, some states may be more costly to visit than others
  - Example: states stored on hard disk vs solid state disk vs memory vs cache
  - Our simple model does not account for that
  - We can visit the cheapest ones first
  - But then, we are using some knowledge...

Sample Code for Blind Search - Treasure Search

- `blind_search_on_tree.py`
- Generate tree with branching factor *b*, depth *d*
- Randomly place the treasure in the tree
- Run different blind search algorithms to find it
- Repeat experiment many times with same tree, different treasure locations
- Blind search algorithms implemented:
  - depth-first search (dfs) - see extras
  - breadth-first search (bfs) - see extras
  - random sampling - discussed next
- Activity: experiment more with this program

Random Sampling

- Idea: go down random path in tree
- Check at each step if treasure is found
- Stop at treasure, or at leaf if treasure not found
  - If not found: restart, try another random walk
- This search is incomplete, probabilistic
  - We may be lucky and find the treasure soon
  - We may be unlucky and miss the treasure every single time
- Sample code tries 1000 times, then gives up
- Activity: experiment with changing that limit

Random Sampling for Treasure Search on Tree

Repeated random sampling on tree
Returns tuple `(found, numNodesSearched)`

```
def sample(tree, start, treasure):
  totalNodesSearched = 0
  for _ in range(1000):
    found, numNodesSearched =
      sampleRandomPath(tree, start, treasure)
    totalNodesSearched += numNodesSearched
    if found:
      return True, totalNodesSearched
  return False, totalNodesSearched
```

Single Random Sample

Single random sample on tree
Returns `(found, numNodesSearched)`

```
def sampleRandomPath(tree, start, treasure):
  numNodesSearched = 1
  current = start
  if current == treasure:
    return True, numNodesSearched
  while tree[current]:
    current = random.choice(tree[current])
    numNodesSearched += 1
    if current == treasure:
      return True, numNodesSearched
  return False, numNodesSearched
```

Experiment in `blind_search_on_tree.py`

- Create uniform tree with $b = 3$, $d = 6$
  - Each node represented by a number
  - See `generate_tree.py` sample code for details
- Repeat experiment for 1000 different random locations of treasure
- Run all three search algorithms
  - Dfs, bfs
  - Random sampling up to 1000 times
- Measure:
  - How often is the treasure found
  - Average number of search steps

A Typical Run

Each run will be different.
The treasure is put into different random locations.

```
mmueller% python3 blind_search_on_tree.py
Tree with 1093 Nodes.
Search with Dfs
1000 Runs 1000 Successes
542.823 Average nodes searched
Search with Bfs
1000 Runs 1000 Successes
542.267 Average nodes searched
Search with Random sampling
1000 Runs 829 Successes
3028.408 Average nodes searched
```

Discussion: Dfs and Bfs vs Random Sampling

- Dfs and bfs are *complete* search methods
  - They find the treasure if it is there
  - They visit each node exactly once
- Random sampling is incomplete
  - It may keep missing the treasure in each random walk
  - The probability of finding it increases towards 1 with more samples
  - Some nodes may never be visited in any finite run
  - Some nodes are visited many times
  - Question: Are nodes equally likely to be visited?

Discussion: Dfs and Bfs vs Random Sampling

- Dfs and bfs are *complete* search methods
  - They find the treasure if it is there
  - They visit each node exactly once
- Random sampling is incomplete
  - It may keep missing the treasure in each random walk
  - The probability of finding it increases towards 1 with more samples
  - Some nodes may never be visited in any finite run
  - Some nodes are visited many times
  - Question: Are nodes equally likely to be visited?
  - Very strong bias - nodes closer to the root are sampled much more frequently

Optional Activity: Expected Number of Steps for Random Sampling

- We saw that for dfs, bfs,
  the expectation is about $n/2$ steps
- How about random sampling?
- More steps expected because of re-visiting the same nodes
- How much more? depends on b and d
- Hint: analyze level by level as when we estimated the DAG state space
- You can start with the case $d = 1$. One root, b children

Discussion and Optional Activity: Random Sampling Without Repetition?

- Q: Could we eliminate the duplication in random sampling?

Discussion and Optional Activity: Random Sampling Without Repetition?

- Q: Could we eliminate the duplication in random sampling?
- Yes, with extra book-keeping
- Keep track of which subtrees still have unexplored nodes
- Only sample among children that are roots of such subtrees
- When a subtree is completely explored
  - Check if parent is also fully explored
  - If yes, recursively check grandparent,...
- Stop algorithm when treasure found, or root is marked as fully explored
- Optional activity: implement this version of sampling

Discussion and Optional Activity: Random Sampling Without Repetition?

- Q: Could we eliminate the duplication in random sampling?
- Yes, with extra book-keeping
- Keep track of which subtrees still have unexplored nodes
- Only sample among children that are roots of such subtrees
- When a subtree is completely explored
  - Check if parent is also fully explored
  - If yes, recursively check grandparent,...
- Stop algorithm when treasure found, or root is marked as fully explored
- Optional activity: implement this version of sampling
- Question: Why is this not done in practice?

Summary of Blind Search

- Blind search uses no heuristics
- Single goal in random location:
  on average, explores half the state space
- Examples: standard graph search algorithms dfs, bfs
- State space too large: cannot complete search
- Random sampling also works
  - Not quite competitive on these examples
  - Re-visits nodes more than once

Blind Search and Two Player Games

- Most of what I said is relevant for the two player case
- More complex because of opponent actions
- There is not a single “goal state”
- Much more on the two player case in future lectures
- Next: how does having a heuristic change single agent search?

Heuristic Search

From Blind Search to Heuristic Search

- We can solve small problems by brute force, systematic search
- We can sometimes solve large problems by luck (e.g. sampling)
- We can sometimes solve large problems by knowledge, without any search
- What can we do in all other cases?
- Answer: use heuristic to *guide a search process*
- Combine power of search with power of knowledge

Example - Treasure Hunt with Heuristic

- Heuristic can help direct the search
- Which action is most likely to lead to treasure?
- If heuristic is good:
  enormous reduction in search effort possible
- Extreme case - perfect heuristic:
  - Always picks a correct action
  - Goes directly to goal

Example - Treasure Hunt with Heuristic

- Heuristic can help direct the search
- Which action is most likely to lead to treasure?
- If heuristic is good:
  enormous reduction in search effort possible
- Extreme case - perfect heuristic:
  - Always picks a correct action
  - Goes directly to goal
- Python demo `heuristic_search_on_tree.py`
- With probability p: follow path to treasure
- Demo: vary p and see what happens!

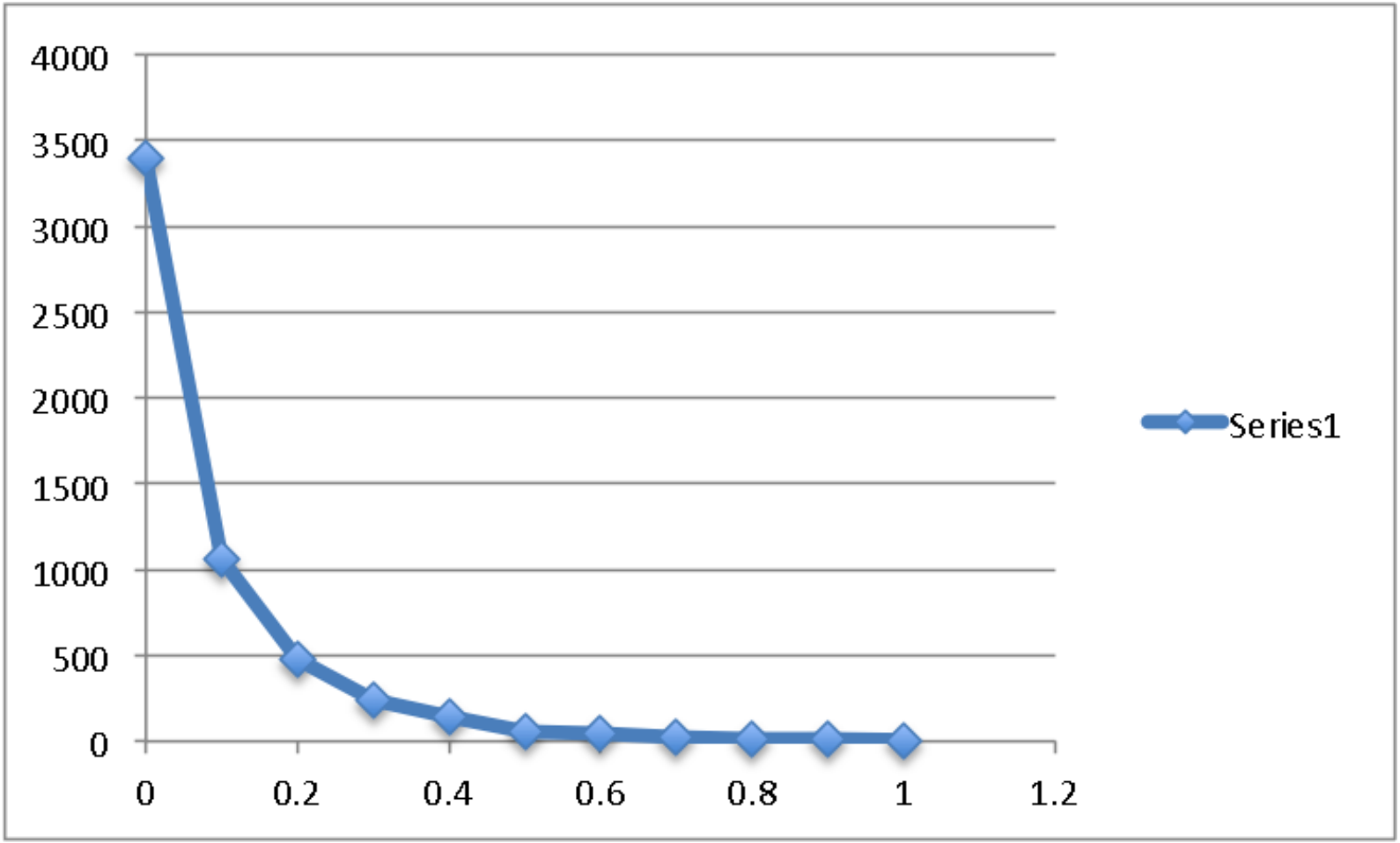
A Typical Run

```
mmueller% python3 heuristic_search_on_tree.py
Tree with 1093 Nodes.
Search with Random sampling
Heuristic accuracy 0
100 Runs 82 Successes 3394.23 Average nodes searched
Search with Random sampling
Heuristic accuracy 0.1
100 Runs 98 Successes 1056.3 Average nodes searched
Search with Random sampling
Heuristic accuracy 0.2
100 Runs 100 Successes 476.94 Average nodes searched
Search with Random sampling
Heuristic accuracy 0.3
100 Runs 100 Successes 241.09 Average nodes searched
Search with Random sampling
Heuristic accuracy 0.4
100 Runs 100 Successes 140.49 Average nodes searched
```

A Typical Run

```
Search with Random sampling
Heuristic accuracy 0.5
100 Runs 100 Successes 60.0 Average nodes searched
Search with Random sampling
Heuristic accuracy 0.6
100 Runs 100 Successes 49.92 Average nodes searched
Search with Random sampling
Heuristic accuracy 0.7
100 Runs 100 Successes 22.69 Average nodes searched
Search with Random sampling
Heuristic accuracy 0.8
100 Runs 100 Successes 14.33 Average nodes searched
Search with Random sampling
Heuristic accuracy 0.9
100 Runs 100 Successes 8.92 Average nodes searched
Search with Random sampling
Heuristic accuracy 1
100 Runs 100 Successes 6.6 Average nodes searched
```

Quality of Heuristic vs Number of Simulations



- X-axis: *p*, Y-axis: average nodes searched

Examples: Applications of Heuristic Search

- Solve the game of checkers
- Play chess, backgammon, Go better than any human
- Find a good path for a character in a video game
- Solve a puzzle such as Rubik's cube
- Plan a tour for the Traveling Salesman Problem
- Control an unmanned vehicle in traffic
- Plan the motion of a robot arm
- Land a spacecraft on a comet
- Control mobile robots in an automated warehouse
- Optimize the location of warehouses for delivery

What are Heuristics Good For?

- They help guide the search through a (large) state space
- Following heuristic is often (much) better than blind search
- Besides state evaluation,
  we can use other forms of heuristics
  - Example: heuristic for move ordering in games
  - Evaluate moves, not states

When Do We Need Heuristic Search?

Typical scenario for modern heuristic search:

- Huge state space
- Heuristics are good, but far from perfect
- Goals:
  - Use heuristic when it works
  - Have a robust method,
    works even when heuristic fails
- Important tool for search: exploration

Why Exploration?

- Classical search methods:
  - Some search algorithm
  - Heuristic evaluation
  - Search is **greedy**, always trusts the heuristic
  - Can get into deep trouble
- Exploration can help
  - Look at other parts of the state space,
    not only where the heuristic leads us
  - Examples: simulation, random walks, many other methods
  - Much more later in this course

Search, Heuristic Evaluation and Simulation

- Many modern heuristic search methods combine all three elements
- **Search** - systematically look ahead into the future
  - *Exploit* the heuristic
  - *Explore* other states as well
- **Heuristic Evaluation** - how good is a state, or an action?
  - Use to guide the search
  - Do not blindly trust it
- **Simulation** - look ahead into the future by *sampling* sequences of future states
  - An example of *exploration*

Heuristic Function

- Heuristic function $h(s)$ defined for each state s
- $h(s)$ estimates the "goodness" of s
- Two *very different* meanings for:
  - Single agent search
  - Two player search
- Heuristic is usually not perfect, but "useful"
- Many ways to define a heuristic function
  (many examples later)

Heuristic Evaluation Function - Single Agent Search

Single agent search meaning of heuristic $h(s)$:

- $h(s)$ estimates the **distance** from s to a goal
- Examples of goals:
  - Reach the treasure
  - Reach the destination in path planning
  - Solve Rubik's Cube
- Goals correspond to terminal states in the state space
- Requirements:
  - $h(s) = 0$ if s is a goal
  - $h(s) > 0$ if s is not a goal
- Intuition: state with smaller h is likely better
- In this course, we will not use this type of heuristics

Heuristic Evaluation Function - Two Player Games

Two player zero sum games:

- What does a heuristic position evaluation mean here?
- Three different popular measures:

1. How likely is a player to win from here?
2. What is player's expected score,
   if win = 1 and loss = -1?
3. In point-scoring games:
   - What score will player get at the end?
   - Example: $h(s)$ = +14.5 points

1. Heuristic as Winning Probability

- $h(s)$ = probability of winning from state s
- For one specific player
  - Example: for Black
  - Example: for the *current* player (toPlay)
- Requirements:
  - $h(s) = 0$ if s is a sure loss for the player
  - $h(s) = 1$ if s is a sure win for the player
  - $0 < h(s) < 1$ for all other s
- Winning probability for *opponent*: $1 - h(s)$
  - This assumes one player has to win (no draws)

2. Heuristic as Payoff in Win/Loss Games

- Requirements:
  - $h(s) = -1$ if s is a sure loss for the player
  - $h(s) = 1$ if s is a sure win for the player
  - $-1 < h(s) < 1$ for all other s
- Often, a draw has value $h(s) = 0$
- Popular translation between winning probability and payoff:
  - Winning probability p
  - Payoff v
  - Translate by $v = 2p - 1$
  - Linear mapping
  - Matches requirements for sure wins and losses
- Payoff for *opponent*: $-h(s)$

3. Heuristic in Point-Scoring Games

- Requirement:
  - $h(s) =$ true score
    if s is a terminal position
  - Payoff for *opponent*: $-h(s)$ (zero sum)
- In games, no real requirements for non-terminal states
  - Just try to predict final score as well as possible
- Compare with single-agent case (see resources)
  - many special types of heuristic, e.g. admissible, consistent,...

Using a Heuristic vs Constructing a Heuristic

1. *Search* part of course (now):
   - How to use a heuristic in search?
2. *Knowledge* and *machine learning* parts (later):
   - How to construct or learn a good heuristic?

Examples - Sources of Heuristics

- Human knowledge from books
- Mathematical analysis of the problem
- Domain knowledge
  - Example: Euclidean or Manhattan distance for path-finding on a map (ignores obstacles)
- Rule-based systems
- Abstractions, such as *pattern databases*
- Solution of *relaxed*, simplified problem
- Trial and error

Learning a Heuristic

Learn a heuristic evaluation function:

- From examples in master games (supervised learning)
- From self-play (unsupervised)
- Construct function from *simple features* of a state
  - Learn weights of those features
- Design a deep neural network for evaluation
  - Learn weights of the network

Example of Misleading Heuristic

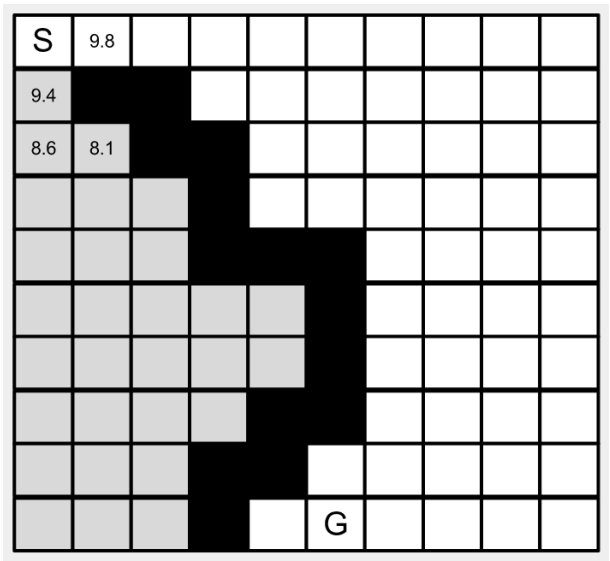


Image source: Fan Xie

A single agent problem:

- Find path from S to G
- Dark squares are obstacles
- Heuristic: Euclidean distance to G
- Misleading heuristic at start prefers to go down, not right
- Imagine making the grid finer and finer...
- ... more and more search effort wasted on the left side

Follow the Heuristic, or Explore?

- Blind search is hopeless in large state spaces
- Focusing only on the heuristic is too risky
- Need to find a balance
  - Follow heuristic when it works well
  - Do not trust heuristic blindly
  - Use exploration when stuck, or heuristic is misleading
  - Always use *some* exploration
    to guard against biases in heuristic
  - How much exploration is best?
- Big questions: when, where, and how to explore?
- Today, we have many case studies, but no full theory

Blind Search as Heuristic Search

- Remember blind search assumptions:
  - "Know the rules" -
    can generate all successors of a state
  - Recognize a goal state once we reach it
  - No other knowledge
- Equivalent in single agent heuristic search:
  "trivial heuristic"
  - $h(s) = 0$ if s is a goal state
  - $h(s) = 1$ if s is not a goal state
  - Search with this heuristic is like blind search,
    no extra information

Summary

- Discussed heuristic vs blind search
- Exploit heuristic knowledge
- Explore to avoid over-reliance on less than perfect heuristics
- Big question: how to find a balance?
- Next topics: solving two player games, minimax and alphabeta