Informatics 2D. Coursework 1: Search and Games

Stefanie Speichert, Claudia-Elena Chirita, Valerio Restocchi

Deadline: 3 PM, Tuesday 10th March 2020

1 Introduction

The objective of this assignment is to help you understand the various search algorithms and their applications in path-finding problems. You will implement the search algorithms using *Haskell* and show your understanding by answering some theory-related questions.

You should download the following file from the Inf2D web page:

http://www.inf.ed.ac.uk/teaching/courses/inf2d/coursework/Inf2D-Ass1-20.zip

You will submit two files:

1. a version of the Inf2d1.hs file containing your implemented functions. **Remember to add your matriculation number to the top of the file**.
2. a PDF titled s<matric>-answers.pdf. You will write all your answers that do not require coding into this file. We do not provide a template. You are free to use any text-editor you like (e.g. Latex, Word,...). Please do not use scans of your handwriting. [1]

Submission details can be found in the Submission section.

The deadline for the assignment is:
3pm, Tuesday 10th March 2020.

Some basic commands of Haskell are listed below[2].

- `ghci` GHCi is the interactive environment, in which Haskell expressions can be evaluated and programs can be interpreted. You can fire up GHCi with the command `ghci` in a terminal. All of the following commands work in GHCi environment.
- `:help` Get information if you are stuck.
- `:cd <dir>` You can save *.hs files anywhere you like, but if you save it somewhere other than the current directory, then you will need to change to the right directory, the directory (or folder) in which you saved *.hs.
- `:show paths` Show the current directory.
- `:load Main` To load a Haskell source file into GHCi. You can also use the short version `:l Main`, where Main is the topmost module in our assignment.
- `main` Run function 'main' (defined in the Main.hs file).

If your Haskell is a bit rusty, you should revise the following topics:

- Recursion
- Currying
- Higher-order functions
- List processing functions such as map, filter, foldl, sortBy, etc
- The Maybe monad

The following webpage has lots of useful information if you need to revise Haskell. There are also links to descriptions of the Haskell libraries that you might find useful:

[1] You will loose marks if anything is illegible for the marker.
[2] From `http://downloads.haskell.org/~ghc/latest/docs/html/users_guide/ghci.html`

Haskell resources: http://www.haskell.org/haskellwiki/Haskell

In particular, to read this assignment sheet, you should recall the Haskell type-definition syntax. For example, a function `foo` which takes an argument of type `Int` and an argument of type `String`, and returns an argument of type `Int`, has the type declaration `foo :: Int → String → Int`. All of the main functions you will write have their corresponding type-definitions already given. However, if you declare your own helper functions you will have to write these yourself.

2 Help

There are the following sources of help:

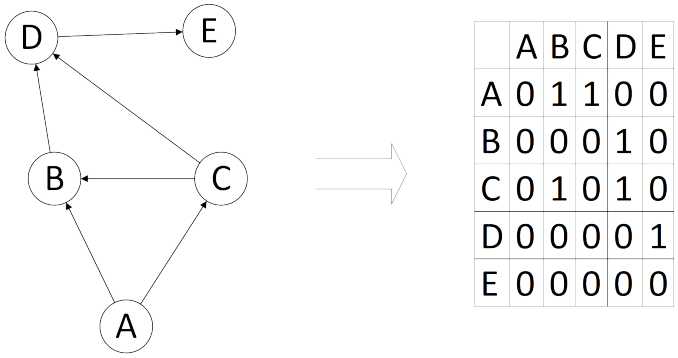
- Attend the lab sessions.
- Check Piazza for peer support and clarifications.
- Read "Artificial Intelligence: A Modern Approach" Third Edition, Russell & Norvig, Prentice Hall, 2010 (R&N) Chapters 3 and 5.
- Email the TA: Stefanie Speichert (s.speichert@ed.ac.uk) She'll also be happy to meet with you if you have a longer (but concrete!) question.
- Read any emails sent to the Inf2D mailing list regarding the assignment.

Do not be afraid to ask for help! While we are not able to give you solutions to the problems we know how to point you in the right direction or help you out when you are stuck!

3 Uninformed Search (35%)

In this section, you will implement some uninformed search procedures (see Chapter 3 of R&N) on graphs.

Problem Statement: You will traverse a directed acyclic graph (DAG). Given a start node n_s and a goal node n_g, your task is to implement different uninformed search procedures to reach the goal node from the start node.



	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	1	0
C	0	1	0	1	0
D	0	0	0	0	1
E	0	0	0	0	0

Figure 1: The graph representation for a graph with uniform cost of 1 per edge.

Computationally, a graph is represented in the form of its adjacency matrix (see fig 1 for an example). The matrix was flattened rowwise and is, therefore implemented as a list of Nodes.

A Node is simply an Integer. The first Node of a graph is indexed by 0.

We use a list of nodes, defined as type `Branch`, to represent the search *agenda*: the set of nodes which are on the fringe of the search tree. In order to write search algorithms in Haskell, we need to represent the status of the search as we expand new nodes to search branches.

As a search problem we have:

- Initial state: `[start]`, where `start` is `Integer`.
- Successor function: the function `next` which extends branches.
- Goal test: the function `checkArrival` checks whether the goal node has been found.

Each time a node is taken from the search agenda, you will have to check if it is the destination node. If it is a solution, the branch is returned as the desired path. Otherwise, the node is expanded using the successor function and the new nodes are added to the search agenda in the appropriate order. Note that you might not reach the goal state at all!

Currently, the start and end node, as well as the graph itself (and the heuristic for A* Search) are defined in the module GraphSearch.hs. While you should not change any helper functions, you can change these variables for testing purposes to allow for different graph sizes, node objectives and heuristics.

Careful! During testing time we will test your implementation with many different DAGs. These will have different number of nodes, different costs per transitions and might include paths where there is no solution at all! Also note that start and end node can be any two nodes in the graph (e.g. your goal node could be node 0).

3.1 Uninformed Search- Basics (6%)

You must first implement the following functions:

- `next :: Branch → Graph →[Branch]`. The first argument is an input search branch. The second is the graph. The `next` function should expand the input branch with its possible continuations, and then return the list of expanded search branches. For example, if your input node is 0, then you can reach node 1 and node 2 . Therefore the return of the next function should look like this: [[1,0],[2,0]].
- `checkArrival :: Node → Node → Bool`. The first argument of this function is the destination node. The second argument is the current node in the graph. The `checkArrival` function is for determining whether the graph has arrived at the destination or not.

3.2 Breadth-first Search (6%)

Breadth-first search expands old nodes on the search agenda before new nodes. You will implement the function and return a search path if the goal was reached or an error otherwise.

- `breadthFirstSearch :: Graph → Node → (Branch → Graph → [Branch]) → [Branch] → [Node]→ Maybe Branch`
- The first item is the graph in the form of a list of Nodes (= the flattened adjacency matrix).
- The second argument is the destination position of type `Node`, based on which the `checkArrival` function determines whether a node is the destination position or not. The branch reaching the destination node is a solution to the search problem.
- `(Branch →Graph → [Branch])` is the type of the `next` function which expands a search branch with new nodes.
- `[Branch]` argument is the search agenda.
- `[Node]` is the list of explored nodes. Before expanding a search branch, you should check whether the current node of the search branch is an old node or not. You should not search for continuations of a repeated node.
- The function returns a value of type `Maybe Branch` which is either `Nothing`, if a solution is not found, or `Just Branch` solution, if one is found. Haskell will automatically throw an error if a function returns `Nothing`. As long as you account for the goal not being found you can simply return `Nothing` and not worry about error declarations.

3.3 Depth-limited Search (7%)

Depth-limited search solves a possible failure of depth-first search in infinite search spaces by supplying depth-first search with a pre-determined depth limit. You will have to implement this procedure. Branches at this depth are treated as if they have no successors. *Hint: First try to implement Depth First Search then modify it to fit the problem description.*

- `depthLimitedSearch :: Graph → Node → (Branch → Graph → [Branch]) → [Branch] →Int → [Node] → Maybe Branch`
- The `Int` argument is the maximum depth at which a solution is searched for.

3.4 Uninformed Search - Questions (16%)

Consider the following graph:

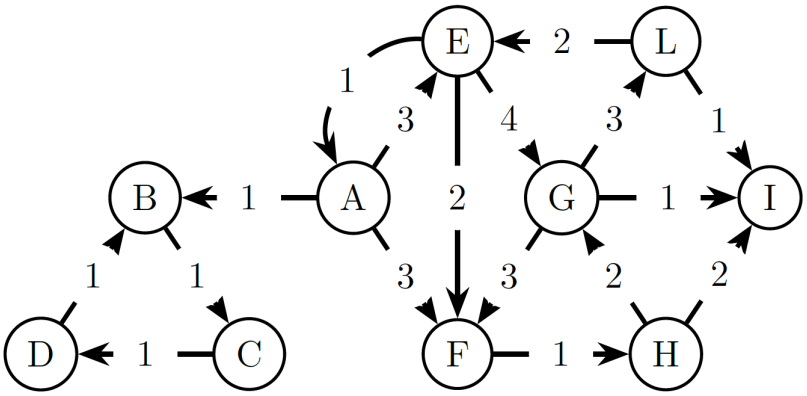


Figure 2: A graph.

All searches performed in this section are with reference to this graph.

1. Breadth First Search (BFS) and Depth First Search (DFS)
   (a) (Manually) perform Breadth First Search *with* elimination of repeated states. The ordering of the nodes is lexicographic. If you get stuck in a loop, indicate the loop. Full points will only be awarded if you show each step of the algorithm instead of only the end result.
   (b) (Manually) perform Depth First Search *with* elimination of repeated states. The ordering of the nodes is lexicographic. If you get stuck in a loop, indicate the loop. Full points will only be awarded if you show each step of the algorithm instead of only the end result.
   (c) Draw a graph that favours BFS. Briefly explain why you choose this graph.
   (d) Draw a graph that favours DFS. Briefly explain why you choose this graph.
   (e) Compare BFS and DFS. What are the biggest differences? (Be concise. Character Limit: 750 characters.)
2. Iterative Deepening Search
   (a) What is the optimal depth for the graph in Figure 2?
   (b) Compare Depth First Search and Iterative Deepening Search. When would you use Iterative Deepening Search over Depth First Search? (Be concise. Character Limit: 500 characters.)

4 Informed Search: The A-Star Algorithm (30%)

In this section you will implement some heuristic search functions. See Chapter 3 of R&N. For simplicity, we will assume now that **the goal is always reachable!**

4.1 A-Star Helper Functions(5%)

You have to first implement two helper functions:

- The heuristic function `getHr:: [Int] → Node → Int` : In this exercise you can assume that the heuristic function for each node to the target has been given in the form of a list. The list is sorted by node index. This means that the heuristic for Node 0 to the goal node is at index 0 and so on.
- The cost function `cost :: Graph → Branch → Int`: The cost function calculates the current cost of a trace. The cost for a single transition is given in the adjacency matrix. The cost of a whole trace is the sum of all relevant transition costs.

4.2 *A** Search (15%)

*A** search ranks nodes on the search agenda according to a heuristic and cost function and selects the one with the smallest combined value.

- `aStarSearch :: Graph → Node→(Branch→Graph → [Branch])→([Int]->Node->Int)→[Int]→ (Graph→Branch→Int)→[Branch]→[Node]→ Maybe Branch`
- The first item is the graph in the form of a list of Nodes (= the flattened adjacency matrix).

- The second argument is the destination position of type `Node`, based on which the `checkArrival` function determines whether a node is the destination position or not. The branch reaching the destination node is a solution to the search problem.
- `(Branch → Graph → [Branch])` is the type of the `next` function which expands a search branch with new nodes.
- `([Int] → Node → Int)` is the heuristic function.
- `(Graph → Branch → Int)` is the cost function.
- `[Branch]` is the search agenda (same as uninformed search).
- `[Node]` is the list of explored nodes (same as uninformed search).
- `Maybe Branch` is the value the function returns (same as uninformed search).
- Both heuristic function and cost function return an `Int` type which makes it possible to get a combined ranking of nodes.

4.3 Informed Search - Questions (10 %)

1. Heuristics
   (a) Why is the straight line distance a valid heuristic? Name the criteria and why they apply in this case.
   (b) Describe (1 sentence) *three* more problems that can be solved with A-Star search and name at least one valid heuristic for each. [3]
2. Best-First and A-Star Search Comparison
   (a) What are the differences between the Best-First and A-Star search strategies?
   (b) How would you change your A-Star implementation to be a search instead? (You do not need to implement these changes, simply state them here.

5 Games: Connect Four with a Twist (35%)

In this section you will implement minimax search with alpha-beta pruning (see Chapter 5 of R&N) for the two-player game **Connect Four** with a slight twist. The board size is 4 columns and 4 rows.

Rules introduction:

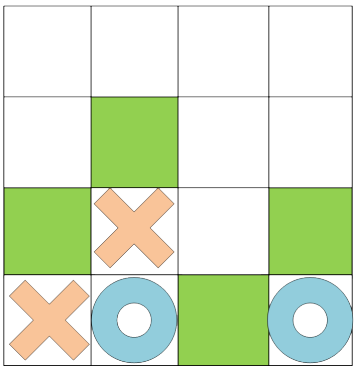
1. Two players take turns adding one piece at a time.
2. Each turn consists of two actions: Placing a piece and choosing to rotate the game to the left; the player first makes their move and can then decide to leave the board as is or if they want to turn the board to the left.
3. If a player has won the game is determined after a full turn (placing a piece and deciding if the board should be rotated).
4. For each column, only the lowest unoccupied cell is the available space for a player to add a piece. An example is given in Figure 1a.
5. When the game board is turned the places automatically fall to the lowest cell to form a valid game board. (See Figure 1b)
6. The winner is the player who is the first to form a horizontal, vertical, or diagonal line of four of one's own pieces. If the game board fills before either player achieves four in a row, then the game is a draw.

5.1 Game Functions

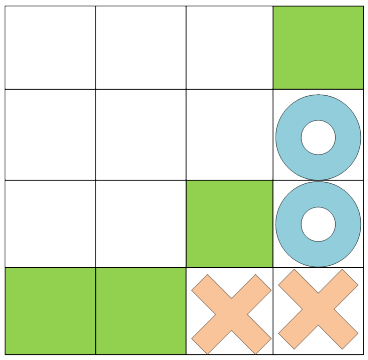
We will represent the 4 by 4 grid for the game board as a list type with 16 elements. Your game player will interact with the game by providing the (row, column) pair for his/her moves. Several useful functions for the Connect Four game are defined in ConnectFour.hs.

- `Game, Cell` and `Role` types: The `Game` type is a list of `Int` types. It represents the 16 cells on the game board. The `Cell` type represents positions on the board as the `(row,column)`, an `Int` pair. The `Role` type describes which player it is, the human player or computer player.
- `getLines` provides all possible lines on the board (rows, columns and diagonals) for winning the game.

[3] You are **not allowed** to use examples from the lectures or the book.



(a) A game state and the possible moves in green.



(b) The same game state after a rotation to the left.

Figure 3: Two possible Connect 4 game states in our version.

- `terminal` function takes a `Game` state and returns `True` if it is a terminal state or `False` if otherwise.
- `movesAndTurns` takes a `Game` and a `Role` as arguments and returns a list of possible game states that the player can move into, that is a combination of all moves either with or without the rotation to the right.
- `checkWin` takes a `Game` and a `Role` and determines if the game is a winning position for the player.
- `switch` allows you to alternate roles of players, e.g. MAX and MIN.

5.2 Evaluation Function (5%)

For this section, you will implement an evaluation function for terminal states. The function determines the score of a terminal state, assigning it a value of +1, -1 or 0 as defined below:

- `eval :: Game → Int`
- `eval` takes a `Game` as an argument, and should return an `Int` type so that game states can be ranked.
  - A winning position for MAX should be +1.
  - A winning position for MIN should be -1.
  - A draw has a 0 value.
  - The `terminal` and `checkWin` functions defined in ConnectFour.hs will be useful.

5.3 Alpha-Beta Pruning (15%)

Alpha-beta finds the minimax value of a state, but searches fewer states than the regular `minimax` function (see RN and lecture notes). It does this by ignoring branches of the search space which cannot affect the minimax value of the state. A range of possible values for the minimax value of the state is calculated. If a node in a branch has a value outside this range then the rest of the nodes in that branch can be ignored, as the player can avoid this branch of the search space (see Figure 2).

Note, we assume that the MAX player plays moves which maximise the value and MIN plays moves which minimise the value. The alpha-beta algorithm from R&N is shown in Figure 2, however note that your function should return the best **evaluation value** reached from the input game state, not the action that achieves this value.

You will have to choose an initial range for alpha-beta pruning. We can't represent infinite numbers in Haskell, so you should use the range (-2, +2) for your `alphabeta` function. Since the maximum evaluation of a game position is +1 and the minimum is −1, this range covers all possible minimax values for the state.

- `alphabeta :: Role→ Game → Int`
- `alphabeta` takes a type `Role` and a `Game` as arguments.
- `alphabeta` returns an `Int` since the terminal evaluation of a game is either +1, a win for player MAX; 0, a draw; or -1, a win for player MIN.
- Your `alphabeta` function should use the `eval :: Game → Int` function to evaluate terminal states.

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a), α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figure 4: The alpha-beta search algorithm (From R&N)

Note, if you find Alpha-Beta pruning too difficult you can instead implement the minmax algorithm for a maximum of 10 % instead of 15.

5.4 Connect Four - Questions (15%)

Consider the game Quadrio `https://www.boardgamegeek.com/boardgame/243498/quadrio`. It is a full version of our simplified Connect 4 with a Twist! You can turn the game in every direction and you can insert pieces from every side now (instead of only from the top).

1. Familiarise yourself with the game and its rules.
2. Given that you can rotate, place a piece and then rotate again, how many possible actions can a player perform during one term?
3. What are the main challenges for implementing Quadrio over Connect Four with a Twist for alpha-beta pruning? Would such an implementation be practical? Give reasons for you decision. (Be concise. Character Limit for this question: 1000 characters.)

6 Notes

Please note the following:

- We mark coding style as well as correctness, make sure you comment your code accordingly!
- To ensure maximum credit, make sure that your code can be properly loaded on GHCi before submitting.
- All search algorithms in this assignment are expected to terminate within 2 minutes or less. If any of your algorithms take more than 2 minutes to terminate, you should make a comment in your code with the name of the algorithm, and how long it takes on average. You may lose marks if your algorithms do not terminate in under 3 minutes of runtime for any of the search problems.
- You are free to implement any number of custom functions to help you in your implementation. However, you must comment these functions explaining their functionality and why you needed them.
- Do not change the names or type definitions of the functions you are asked to implement. Moreover, you may not alter any part of the code given as part of the assignment other than specific variables for testing purposes. You may only edit and submit your version of the **Inf2d1.hs** file.
- You are strongly encouraged to create your own tests to test your individual functions (these do not need to be included in your submitted file).

- Ensure your algorithms follow the pseudocode provided in the books and lectures. Implementations with increased complexity may not be awarded maximum credit.

7 Submission

Create a directory in which you keep the files you submit for this assignment. This directory should be called `Inf2d-ass1-s<matric>` where `<matric>`is your matriculation number (e.g 1234567).

In this directory, write answers to all questions that do not require code into a single PDF file that you will call `s<matric>-answers.pdf` . For everything else use the `Inf2d1.hs` file. This file should contain all your code. You should *not* change code in any other file. [4]

You will submit through LEARN. For this purpose you need to create a single file of your submission directory. On a DICE machine you can use the following command:

```
tar -cvzf Inf2d-ass1-s<matric>.tar.gz Inf2d-ass1-s<matric>
```

Make sure that this file contains the correct versions of two submission files.

Submit that file via LEARN. You do this by using the LEARN interface for our course (Inf2D)

Your file must be submitted by **3 pm, Tuesday 10th March 2020**.

You can submit more than once up **until the submission deadline**. All submissions are timestamped automatically. Identically named files will overwrite earlier submitted versions, so we will mark the latest submission that comes in before the deadline.

If you submit anything before the deadline, you *may not resubmit afterward.* (This policy allows us to begin marking submissions immediately after the deadline, without having to worry that some may need to be re-marked).

If you do not submit anything before the deadline, you may submit **exactly once** after the deadline, and a *late penalty* will be applied to this submission, unless you have received an *approved extension.* Please be aware that late submissions may receive lower priority for marking, and marks may not be returned within the same timeframe as for on-time submissions.

For additional information about late penalties and extension requests, see the School web page below. Do **not** email any course staff directly about extension requests; you must follow the instructions on the web page.

`http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests`

Good Scholarly Practice: Please remember the University requirement as regards all assessed work for credit. Details about this can be found at:

`https://www.ed.ac.uk/academic-services/students/conduct/academic-misconduct`

and at:

`http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

Furthermore, you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository then you must set access permissions appropriately (generally permitting access only to yourself, or your group in the case of group practicals).

[4] If you change some of the code in any other file for testing purposes make sure that your implementation still works with the default as this is what you will be marked on. You should *not* submit any other file even if you modified it.