AI S20 Assignment 1 Report

By Sergey Semushin (BS18-05)

Table of contents

General information

- The code is tested only on SWI-Prolog (threaded, 64 bits, version `8.1.21-198-gd129a7435`).
- The code uses a `clpfd` library for convenient working with integers.
- The heuristic function I used: firstly, try to make steps, then pass the ball diagonally, then pass in a straight line. The motivation for this function was that passing is not needed in most cases, but passing the ball diagonally will help to move it to a further distance. Also, this function considers distance to the touchdown point if it is visible.
- Example of using the code:
    - For additional info about available parameters
      `swipl -s main.pl -g main -- -h`
    - For running *random search* indicating max path length
      `swipl -s main.pl -g main -- --map maps/map.pl --alg random_search --max 500`
    - For help running *heuristic search* with increased vision
      `swipl -s main.pl -g main -- --map maps/map.pl --alg heuristic_search -v 2`

Assumptions

- I assumed that more than one orc, human and/or touchdown point can't be at the same coordinates.
- I assumed that a human can pass the ball on a free move.
- I assumed that random search should be runned one time and can make 100 moves maximum.

Structure of the code

On the "lowest level" of my program I defined following fact:
`move_types(ID, FUNCTION, DX, DY, H).`

- `ID` - unique number of move type
- `FUNCTION` - prolog rule which, given `DX`, `DY`, current `X` and current `Y` will return `NEW_X` and `NEW_Y` (or test that a move to these coordinates is possible)
- `DX` and `DY` - for defining direction
- `H` - priority in which moves are made in heuristic search

Then, there is a general `search/6` rule, which implements pure backtracking with no optimization. This rule, as a first argument accepts particular method of search, one of the following:

- `random_search` - defines which move to make at random
- `backtracking_search` - just gives an optimization by cutting branches with too much cost
- `heuristic_search` - changes the order of traversing of the branches and optimizes by the same method as `backtracking_search`

There is also `iterative_deepening_search`, but it is implemented differently

The `main/3` uses `search/6` to find the path, and `search/6` uses `move_types/5` to define which moves are possible is order to split to multiple branches (except `random_search`)

Proposed test maps

Symbols explanation

- `#` - end of the map
- `S` - starting point
- `H` - human
- `O` - orc
- `T` - touchdown
- `/` - symbol to mark that map can have different sizes

Maps

- **`assignment.pl`**
  Example from the assignment description

```
# # # # # # #
# . . . . . #
# . T . . . #
# H O . . . #
# . . . . . #
# S . . . . #
# # # # # # #
```

- **`emptyN.pl`**
  `N` can be 5, 10 or 20
  This map is supposed to test how good algorithms are on simple maps of different sizes

```
# # # / # # #
# . . / . T #
# . . / . . #
/ / / / / / /
# . . / . . #
# S . / . . #
# # # / # # #
```

- **`human_path.pl`**
  To test how algorithms handle free move (and to make iterative deepening search's life easier)

```
# # # # # # #
# . T . . . #
# . H H H . #
# . . . H . #
# H H H H . #
# S . . . . #
# # # # # # #
```

- **`labyrinth.pl`**
  To test algorithms in “natural” environment

```
# # # # # # # # # # # #
# . . . O . . . O T . #
# . O . . . O . . O . #
# O O O O O O O . O . #
# . . . . . . . . O . #
# . O O O O O O O . O . #
# . . O . . . . . . . #
# O . O O O O O O H O #
# . . . . . . . . O . #
# . O O O . O O O O . #
# S O . . . . . . . . #
# # # # # # # # # # # #
```

- **`no_touchdown.pl`**
  To test algorithms on unsolvable maps and to test time it takes detect that it is impossible to solve

```
# # # # # # # # # # # #
# . . . O . . . O . . #
# . O . . . O . . O . #
# O O O O O O O . O . #
# . . . . . . . . O . #
# . O O O O O O O . O . #
# . . O . . . . . . . #
# O . O O O O O O H O #
# . . . . . . . . O . #
# . O O O . O O O O . #
# S O . . . . . . . . #
# # # # # # # # # # # #
```

- **`pass_required.pl`**
  To test how good algorithms are at passing the ball (and to make heuristic search’s life harder)

```
# # # # # # #
# . . . O T #
# . . . O H #
# . . . . O #
# . . . . . #
# S . . . . #
# # # # # # #
```

- **`vision.pl`**
  To test how increasing vision distance will improve heuristic algorithm

```
# # # # # # #
# . . . . . #
# . . . . . #
# . . . . . #
# . . . . . #
# S . T . . #
# # # # # # #
```

Part 1. Running tests

In the following table every map-algorithm pair has:

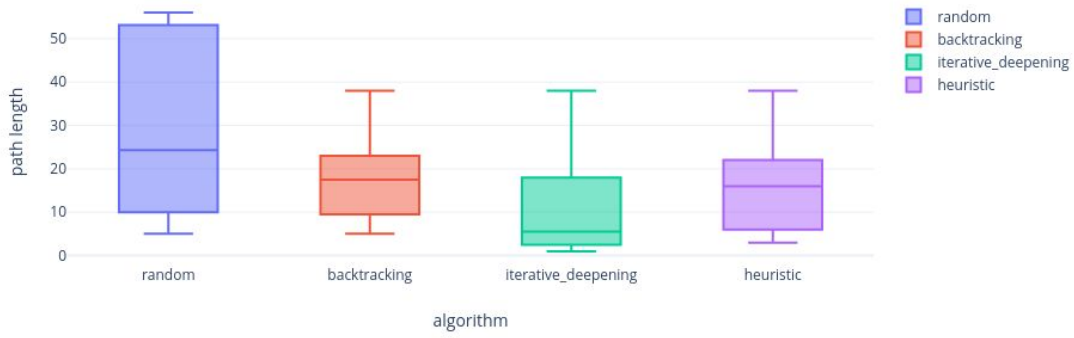
- percentage of failure (when algorithm could not find path when it exists or when algorithm was running for an unreasonable amount of time). If it is not 0%, average path length and time are taken only from successful runs
- average path length (when path is found)
- average time of running (in milliseconds, if not stated otherwise)

Every map-algorithm pair was tested 20 times.

Map \ Algorithm	**random**			**backtracking**			**iterative_deepening**			**heuristic**		
`assignment.pl`	60%	5.0	10.02	0%	17	6.126	0%	3	4.791	0%	3	12.10
`empty5.pl`	25%	53.14	27.03	0%	8	1.738	0%	8	44.84	0%	8	14.31
`empty10.pl`	95%	56	28.0[1]	0%	18	3.283	0%	18	862.6	0%	18	20.81
`empty20.pl`	100%	-	53.22	0%	38	9.201	0%	38	16.9**s**	0%	38	35.21
`human_path.pl`	10%	24.3	24.11	0%	5	0.553	0%	1	7.031	0%	4	13.67
`labyrinth.pl`	100%	-	10.35	0%	26	8.320	0%	18	142.8	0%	25	38.23
`no_touchdown.pl`	0%	-	8.938	0%	-	29.81	0%	-	2.84**s**	0%	-	55.84
`pass_required.pl`	80%	10.0	11.42	0%	11	1.208	0%	3	5.122	0%	19	24.13
`vision.pl`	30%	24.3	15.94	0%	20	3.412	0%	2	2.002	0%	14	17.43
Overall mean	55.56	28.79	21.00	0	17.88	7.072	0	11.38	2.31**s**	0	16.13	25.75
Overall variance	1609	458.5	203.8	0	112.4	82.14	0	163.4	3.1**e**7	0	136.9	213.8

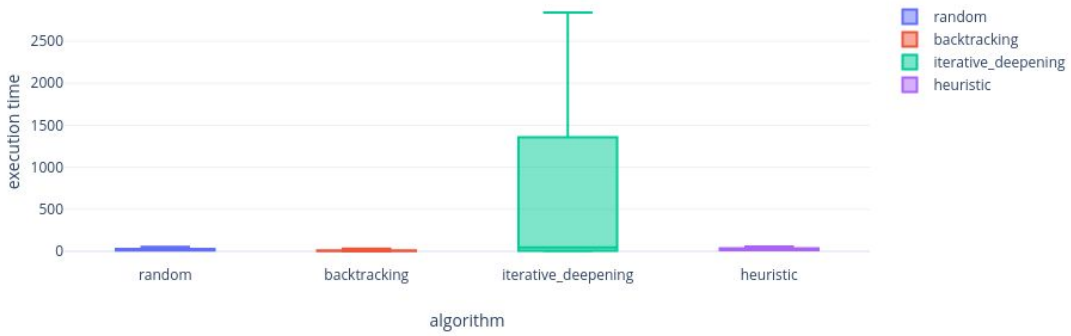
Visualization

On the following plot you can see the visualization of sets of path lengths found by different algorithms



[1] If we consider not only one successful run, then average time is 47.134 msec

And the visualization of execution time:



As iterative_deepening_search's outliers are too far away, here is zoomed in version:

Conclusions

By comparing different algorithms, we can conclude several points:

- *Random search* fails to find existing paths too frequently even for small maps.
- *Iterative deepening* method is performing better on small maps. Despite it always returning the best solution, for big maps it runs too long.
- *Backtracking* and *heuristic* produce very similar results in terms of path length, but heuristic is mostly better. *Heuristic search* was worse than *backtracking* only on specially designed for this map.
  Note: "better" and "worse" is about path length here, not about time. Path length is not random, so it is valid to claim such a sentence without performing statistical tests.
- With my heuristic, search finds better path lengths, but runs for longer time on average. This is not what I expected. Probably, it is because it is hard to invent good heuristic function when agent has limited vision and we can't use distance-based heuristic functions.

Part 2. Bigger field of view

Making field of view 2 instead of 1 can not make a solvable map to become unsolvable, because it only increases the abilities of the agent.

Doing so can not make an unsolvable map to become solvable, because it does not give any new possible moves. However, it can make some searches to run faster or find better paths.

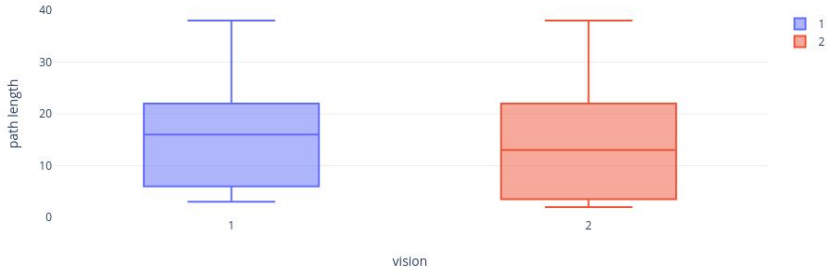
Changing vision does not change anything for *backtracking, iterative deepening* and *random* searches, because *random* search performs random moves which does not depend on vision, and the other two implemented algorithms just bruteforce all possible solutions, also not depending on vision. As for the *heuristic* search, it can change order of traversal depending on vision.

So, let's compare *heuristic search* from the previous table and *heuristic search* when we give it more vision (first value - average length over 20 runs, second value - average time over 20 runs)

Map \ Vision	**1**		**2**	
`assignment.pl`	3	12.10	3	11.82
`empty5.pl`	8	14.31	8	14.01
`empty10.pl`	18	20.81	18	20.92
`empty20.pl`	38	35.21	38	35.85
`human_path.pl`	4	13.67	4	13.74
`labyrinth.pl`	25	38.23	25	38.15
`no_touchdown.pl`	-	55.84	-	55.79
`pass_required.pl`	19	24.13	19	24.03
`vision.pl`	14	17.43	2	10.59
Overall mean	16.13	25.75	14.63	24.99
Overall variance	136.9	213.8	162.3	235.6

Visualization

On the following plot you can see the visualization of sets of path lengths found by different algorithms



And the visualization of execution time:

T-test

Although vision clearly improved path length (from 14 to 2) and average time (from 17.43 to 10.59) on the `vision.pl` map for *heuristic search*, it is worth checking if there is overall performance difference in terms of time of *heuristic search* on all maps (possibly, it might be even worse for increased vision as we have to compute more stuff). So, assuming that distribution of execution time on different maps is normal and variances does not depend on vision distance, I decided to perform t-test on these sets.

Firstly, to perform a t-test, we need to formulate our hypotheses:
H_0 = There is no significant difference in execution times between 1 cell vision and 2 cell vision
H_a = There is significant difference in execution times between 1 cell vision and 2 cell vision

Then, we need to find the t-value:

$$\overline{X_1} \approx 25.7478$$

$$\overline{X_2} \approx 24.9889$$

$$S_1^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(X_{1i} - \overline{X_1}\right)^2 \approx 213.7529$$

$$S_2^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(X_{2i} - \overline{X_2}\right)^2 \approx 235.6479$$

$$S_{\overline{X_1}-\overline{X_2}} = \sqrt{\frac{S_1^2}{N_1} + \frac{S_2^2}{N_2}} \approx 7.0665$$

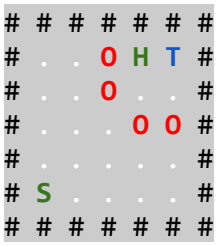
$$t = \frac{\overline{X_1} - \overline{X_2}}{S_{\overline{X_1}-\overline{X_2}}} = \frac{29.4193 - 28.7759}{7.0665} \approx 0.0910$$

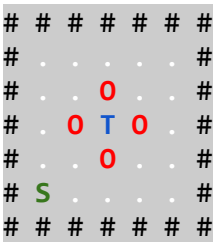
Lastly, with t = 0.0910 and 8 degrees of freedom, we can calculate p-value: p = 0.92973…
So, we can conclude that the difference is considered to be not statistically significant.

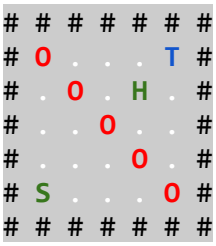
Part 3. Hard to solve and unsolvable maps

Unsolvable maps

In all unsolvable maps there are either no touchdown points at all (`no_touchdown.pl`) or touchdown points are separated from the initial position by a wall of orcs. Some examples of such maps:







Any other map is solvable, but it might take multiple tries for *random search* or longer time for any algorithm. Algorithms can also produce worse solutions on some maps depending on size, particular arrangement of orcs, touchdown points and humans on the map.

Hard to solve maps

For *random search* and *backtracking search*, the hardness of the map depends only on the amount of paths that neither fail immediately nor lead to the goal. So, for *random search* and *backtracking search* `empty20.pl` is harder than `empty5.pl` and even harder than `labyrinth.pl`.

For *iterative deepening search*, map is harder the bigger minimal path is. For example, it runs for a relatively long time on `empty20.pl` (16.9**s** on average), but it's very good on `human_path.pl` and `vision.pl` as it finds the best solution at an average of 7ms and 2ms correspondingly.

As for *heuristic search*, the hardest maps are those which differ from the most of maps and therefore they make heuristic function to work against reaching the goal. Particularly, on `pass_required.pl` map heuristic performed worse than *backtracking* and *iterative deepening* algorithms.