PARALLEL A-STAR

Salvatore Licata
Lorenzo Ferro
Alessandro Zamparutti

August 2022

1 Introduction

Implementation of the A-Star algorithm both parallel and sequential versions on large and weighted benchmark graphs. Study on performance and comparison between the 2 implementations.

The project implies:

- reading of large weighted benchmark graphs
- run of A-Star parallel, sequential and standard Djkstra algorithm
- comparison considering computation time and memory usage

2 Reading

Starting from a .map file (standard format for maps rappresentation, using ascii characters to represent walls and passable terrain) the `graph_generator.py` python script is able to create a .txt with the format required by our A-Star implementation.
In the benchmark section we have:

- huge-graphs: graphs with millions of nodes, used for comparisions on large dimensions
- maze: graph with a small number of solutions (few path available from a point to another)
- street-maps: true city grid maps with different size
- small-graphs: generated by us used for testing

The weight of the edges can be randomized by the script in order to raise a level of difficulty in finiding a path (all the grids benchmark have only edge with weight 1)

3 Basic structures

The project started implementing all the basic structures needed by the algorithm. All the structures are as generic as possible to encourage reuse and are dynamically allocated and are implemented using lists to have no static limitations.
Each structure have a create fucntion for allocation and a destroy one for disallocation of the memory.

3.1 hash table

Is implemented with multiplicative modular method with golden ratio:

$$golden\_ratio = (\sqrt{5} - 1)/2 \tag{1}$$

$$P = 8191 \tag{2}$$

$$hash(key, module) = ((key * golden\_ratio)\%P)\%module \tag{3}$$

The structures re-allocates itself when reaches the 3/4 of the maximum capacity, in case of collisions concatenate the elements in a list (keeps in the first place the last inserted item).

3.2 heap

The heap structures uses hash table for faster addressing and is used as a priority queue with the possibility to have MAX or MIN priority as first element

3.3 queue

A generic list with Head and Tail pointers with a generic item. FIFO queue with head extraction and tail insertion

3.4 stack

LIFO implementation with both insertion and extraction on the head

3.5 graph

Generic Graph with possibility to be UNDIRECTED or DIRECTED, have a pointer to generic data (possibility to manipulate passing custom funcitons eg. 2d data).
Each vertex has a true_cost (the cost of the path from start to that vertex) and a heuristic_cost (exstimation of the cost from that node to the destination node). The graph also contains a Hash table of all the nodes used to efficiently find a node (a good improvement in the graph generation).

4 Approach

Put your approach text here.

If you are working with others on the paper, you can use custom footnotes like this one [1]

5 Summary

Put your summary text here.

References

6 Appendix

Put your appendix text here.

[1] Itzik: I think the template is coming along nicely, good job!