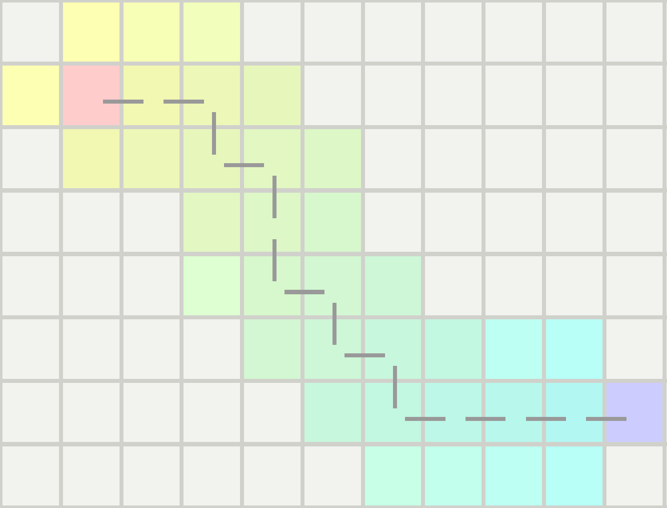
Jamie Haddow 0705082
CMP201 Data structures and algorithms
Comparing Lee and Astar pathfinding algorithms



Introduction

The size of complexity of AAA games is staggering. Kilometers of worlds to explore.

Thats all good having this large, crazy realistic world,but when it comes down to it...

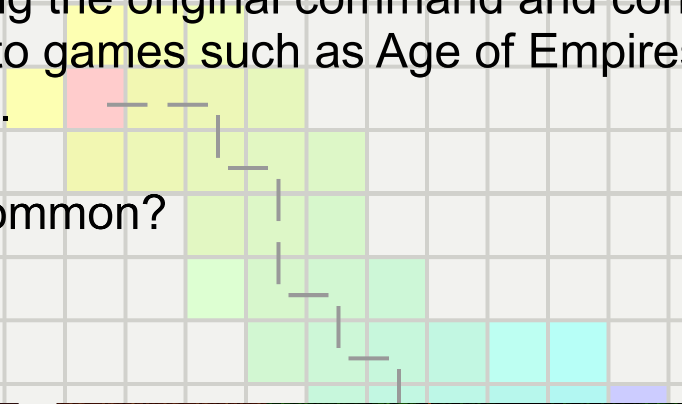
How do you get from A to B to C? Pathfinding!!

Strategy games are my jam.

My first strategy game being the original command and conquer and then evolving to the X-Com series, then to games such as Age of Empires, Warcraft 1-3, Civilization, Advance wars.

What do they all have in common?

A grid system.

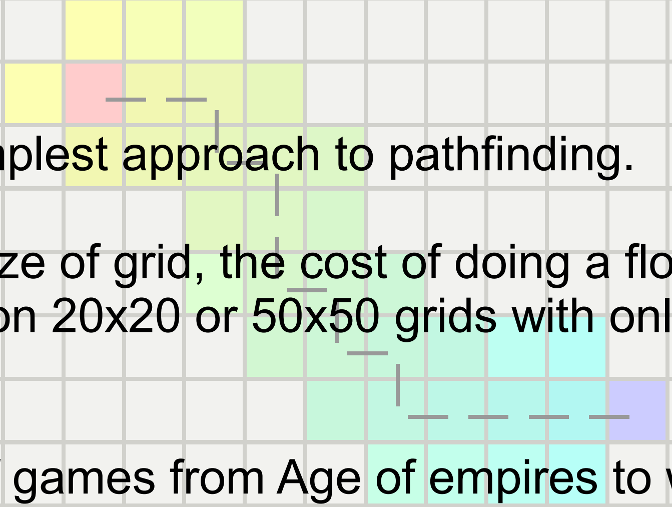


What algorithm to use

With that in mind, a number of my own game ideas involve a grid system.
From a dungeon crawler in a procedurally generated map to a large map
of say 100x100 tiles where humans need to defend a world from an enemy
made of liquid. To make this, I need a pathfinding algorithm that suits
my needs. Someting that works well on a small scale, not resource intensive but also
not overly complicated to implement.

The two pathfinding algorithms I decided to use:

Lee / Astar.



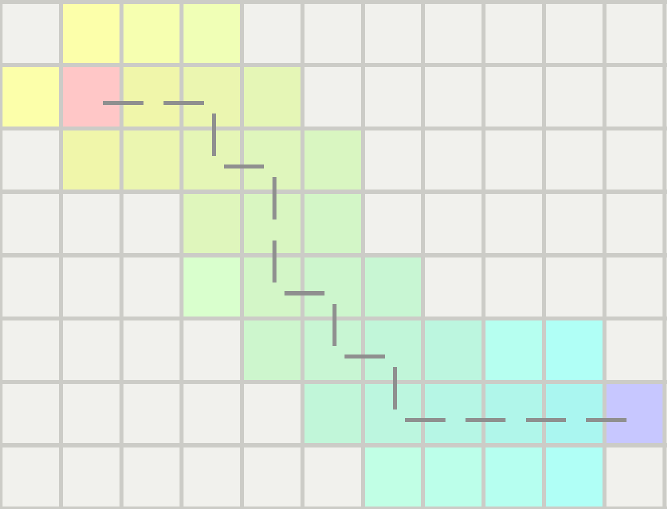
Lee, due to it being the simplest approach to pathfinding.

Until you reach a certain size of grid, the cost of doing a flood search is not an issue.
Especially smaller games on 20x20 or 50x50 grids with only 1 unit moving at a time
in a turn based game.

Astar being used in a lot of games from Age of empires to warcraft
as a more advanced pathfinding method where large maps and obstacles are an issue.

'People underestimate the challenge of making your game do something not stupid.
Also known as 'Artificially idiocy' Its not about writing the perfect algorithm
but more one that isnt stupid.'

As an example on how the pathfinding works on astar in sfml for the graphical aid....

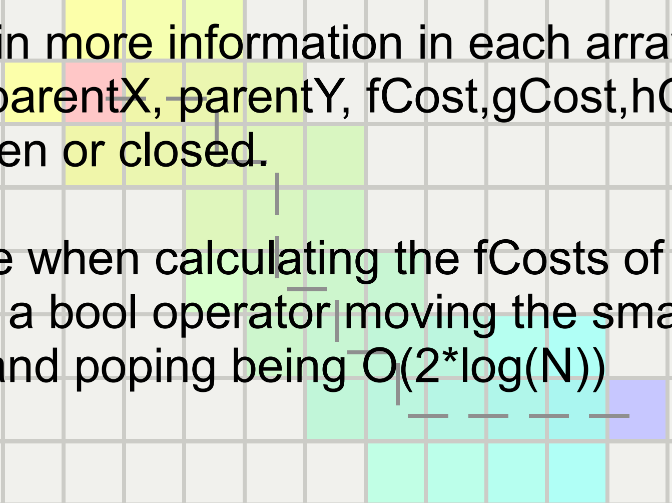


How I did it + complexity

In both my Lee and Astar, I used Arrays for creating the grid,and pathfinding.
I used a nested for loop to create the grid, populating it with relevent numbers
Memory complexity being O(n^2) n being the height x width.

With lee‘s, I only had to worry about increasing a number as I progressed through the grid. O(n)

I then used a list to track the path from the end to the start at the end.
This being a linear constant.



Astar also needed to contain more information in each array element so I used a struct
which contained the X, Y, parentX, parentY, fCost,gCost,hCost and a bool to check
if an element/node was open or closed.

For deciding where to move when calculating the fCosts of neigbouring elements,
I used a priority queue with a bool operator moving the smallest fcost to the front of the queue.
Pushing being o(LOG(N)) and poping being O(2*log(N))

Why use an array?

With the game concept i'm following, using a grid, I chose Arrays due to it being easy to update,control and edit in the sense of the game world im trying to create.

With the size of my game concept, The array would be more memory efficient than using a vector. Knowing that the size of the level will be static, using a dynamic structure would be overkill.

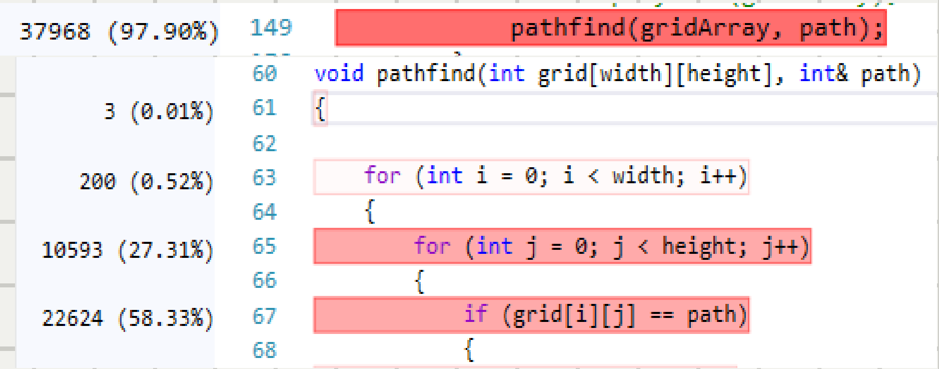
For accessing character positions, using an array has a constant time operation for any location on the grid.

I could have used vectors or a nested pair for Astar but I was unsure how to use them properly. This is something I would like to look into over the winter holiday.

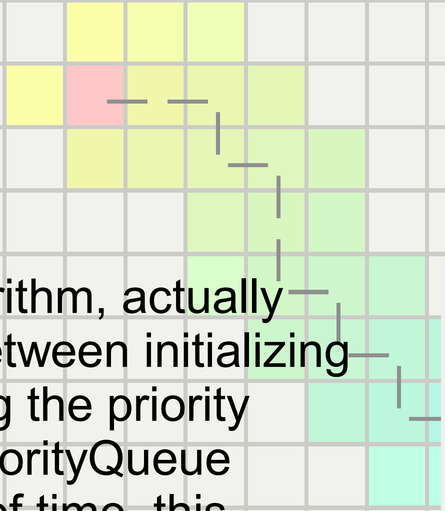
Performance profiling

Running the performance profiler on both algorithms using 500x500 grids gave me these results.

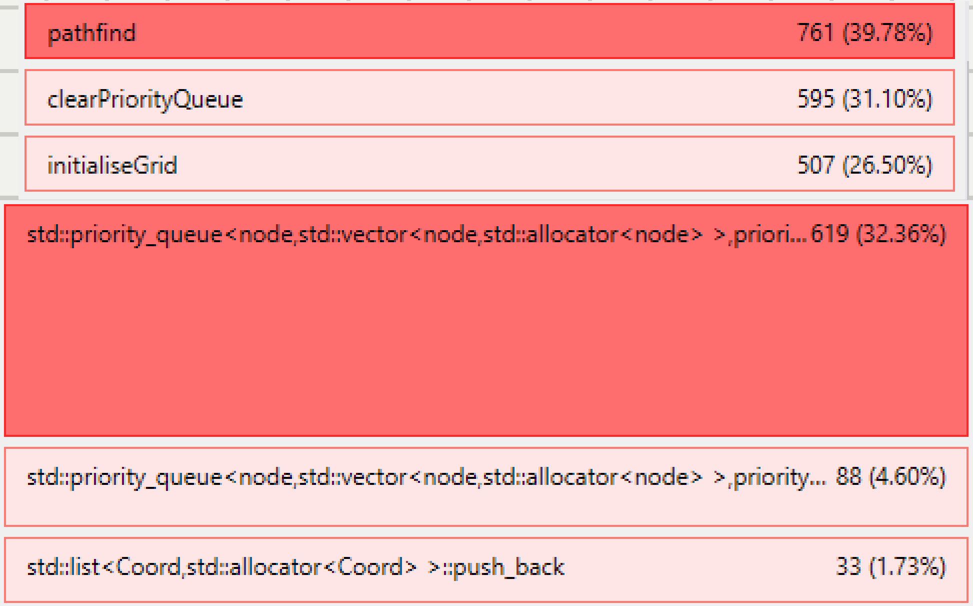
With the Lee algorithm, the pathfind function took up nearly all the computation time with the majority of the time looking through the inner nested loop.



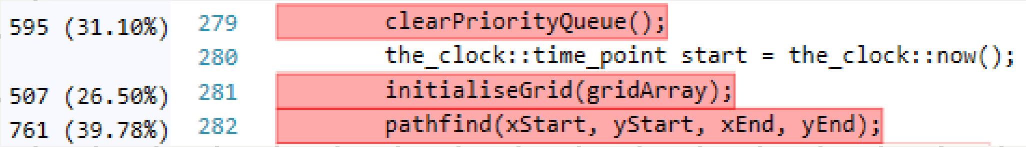
```
37968 (97.90%)        pathfind(gridArray, path);
               void pathfind(int grid[width][height], int& path)
     3 (0.01%) {

   200 (0.52%)     for (int i = 0; i < width; i++)
                   {
 10593 (27.31%)        for (int j = 0; j < height; j++)
                       {
 22624 (58.33%)            if (grid[i][j] == path)
                           {
```



However with the Astar algorithm, actually a lot of the time is split up between initializing the grid and pushing/popping the priority queue. Although the clearPriorityQueue function does take up a lot of time, this was just added for performance checking multiple iterations.



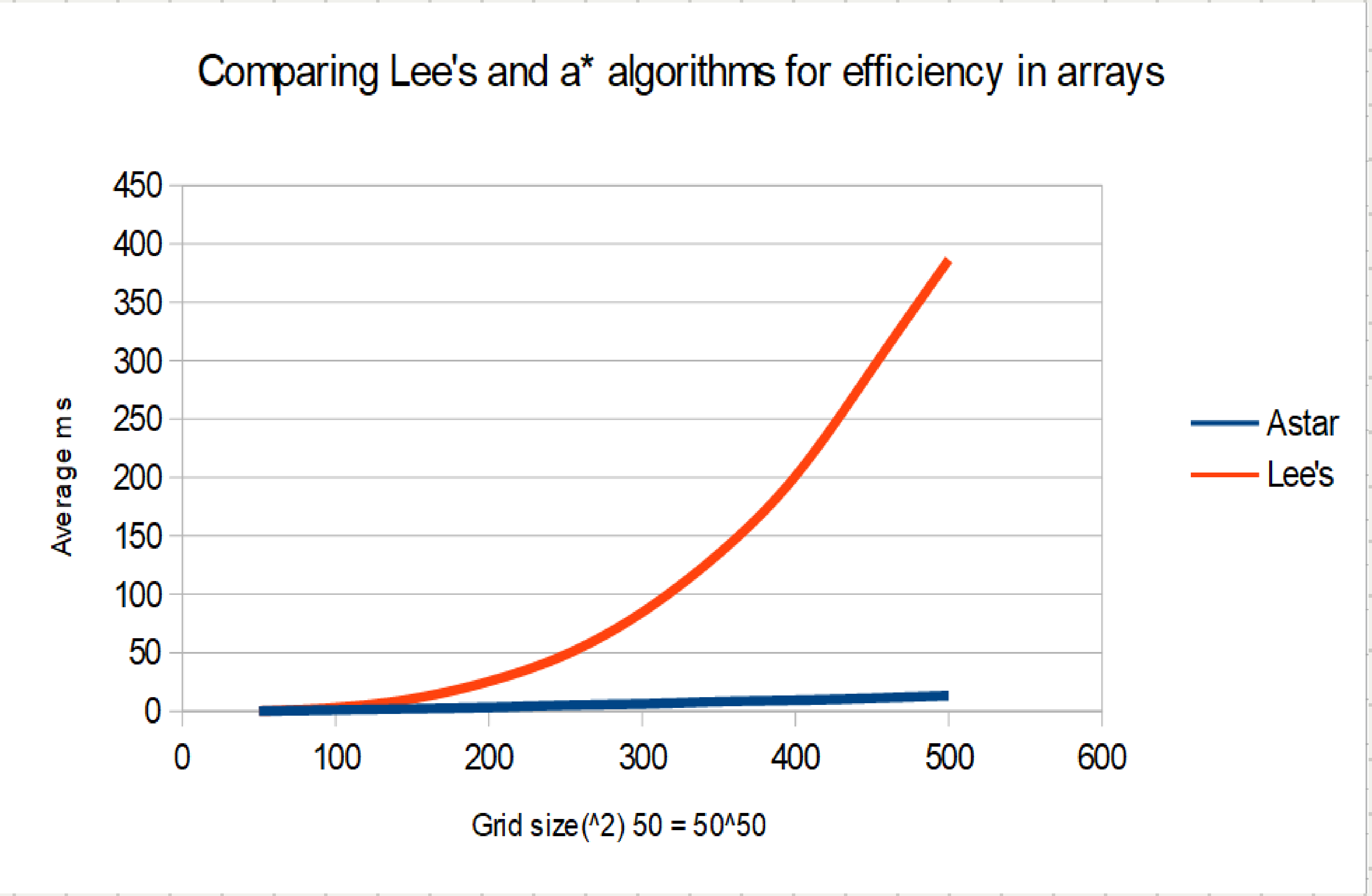
```
pathfind                                                                      761 (39.78%)
clearPriorityQueue                                                            595 (31.10%)
initialiseGrid                                                                507 (26.50%)
std::priority_queue<node,std::vector<node,std::allocator<node> >,priori... 619 (32.36%)
std::priority_queue<node,std::vector<node,std::allocator<node> >,priority... 88 (4.60%)
std::list<Coord,std::allocator<Coord> >::push_back                           33 (1.73%)
```



```
595 (31.10%)        clearPriorityQueue();
                    the_clock::time_point start = the_clock::now();
507 (26.50%)        initialiseGrid(gridArray);
761 (39.78%)        pathfind(xStart, yStart, xEnd, yEnd);
```

To measure the performance of both algorithms, I used a simple bar chart.

Here, you can see that until the grid size increases past 150x150 the performance Both lee's and Astar are very similar to begin with. However, this quickly increase as the grid size increases. Again, with the game concept I have in mind for my Pathfinding, I would never be using a grid above 100x100

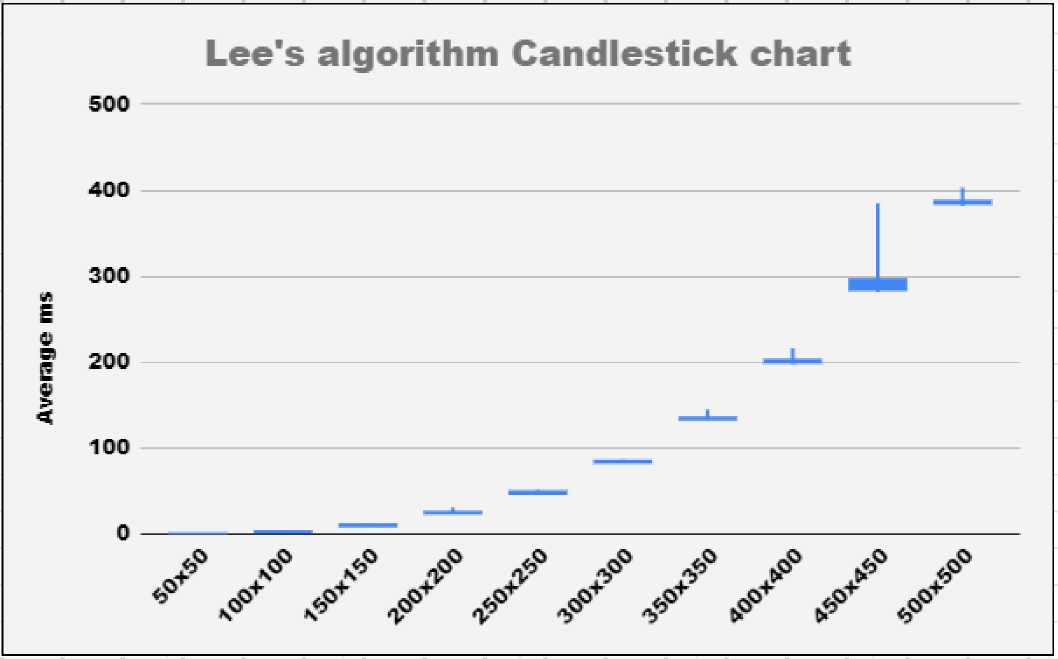


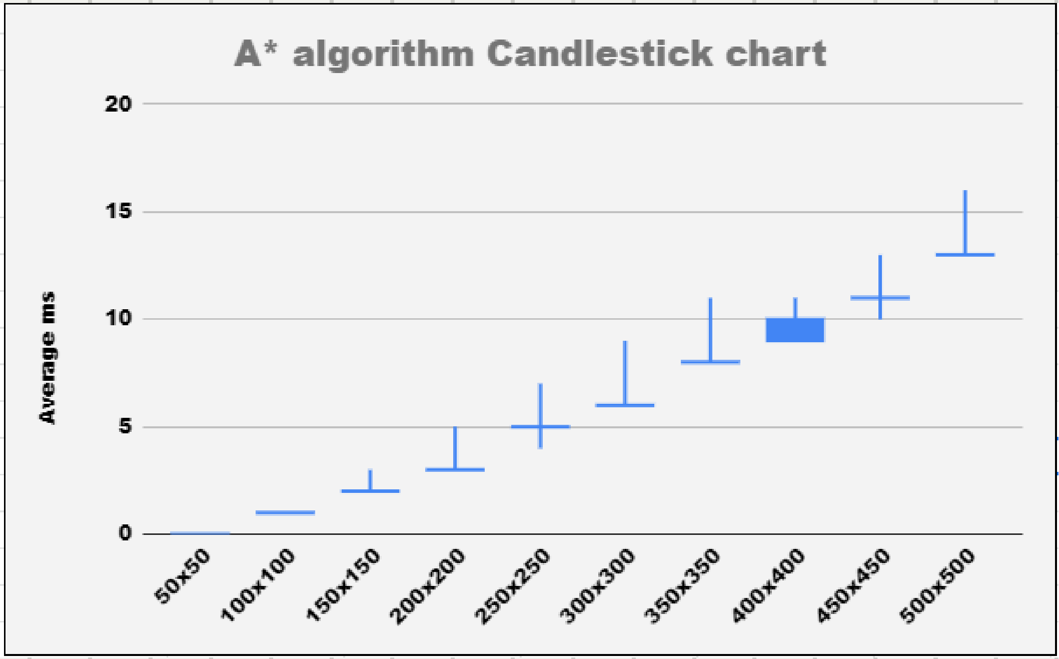
Here, I used a candlestick chart recommended by Adam during one of the lectures.

With lees algorithm, you can clearly see a logarithmic curve as the grid size increases. O(n^2)

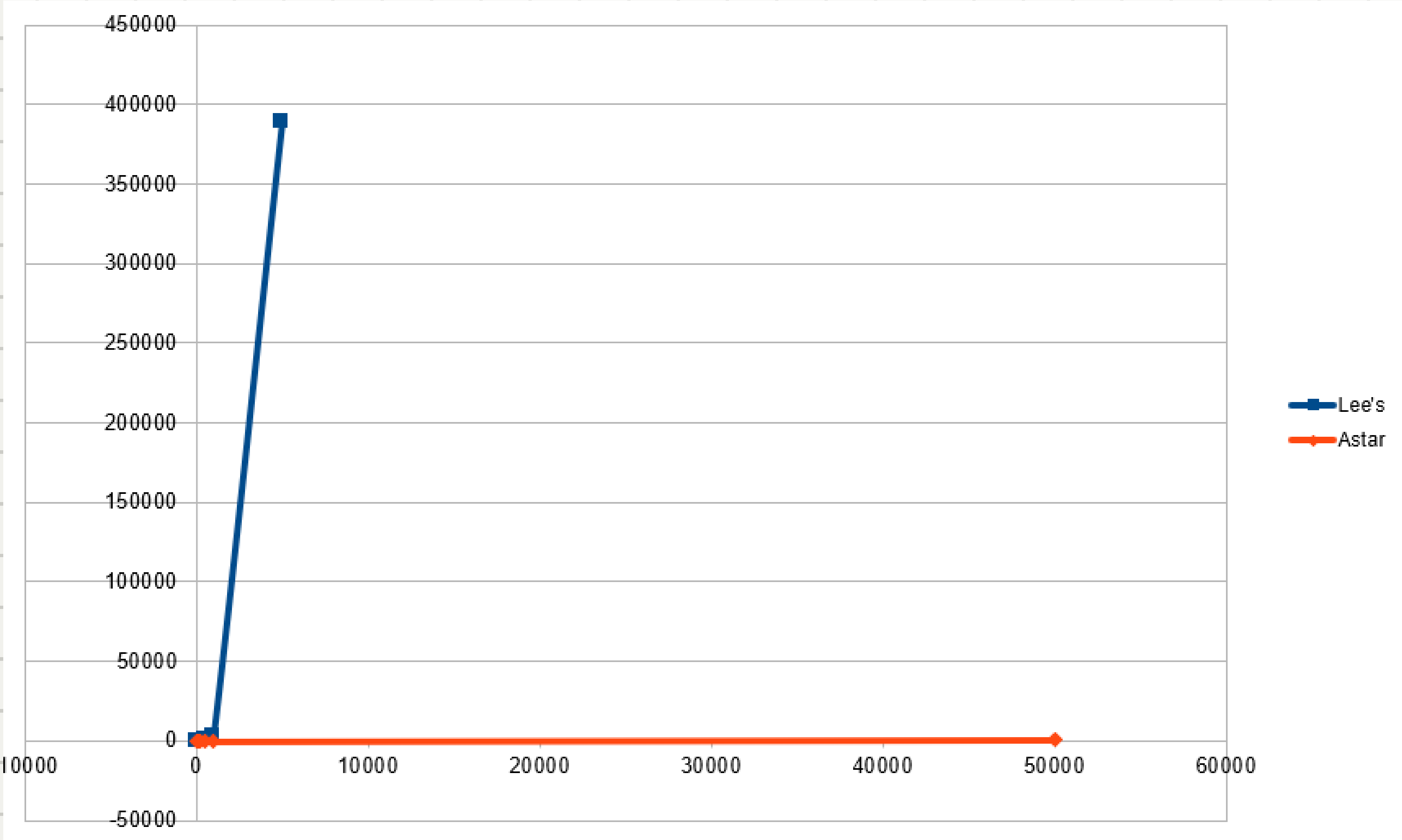
Astar on the other hand, the average ms follows the grid size in a steady pattern. O(nlogn)

Like i mentioned on the previous slide, not going above a grid size of 100x100, using either would result in a negligible difference.





Originally i tested the grid sizes up to 5000x5000 but realized that there was such a huge difference in efficiency after a point that the graphs I would be using would actually be useless



Conclusion

For the purpose of my game, in the end, from the results, it wouldn't matter too much in terms of which one to use.

The difference being, 1ms for Astar and 3ms for lee's.

If I was increasing the grid to a 3d map then Astar would be the clear winner as we would then be using O(n^3) for lee's