death

Rosia E Evans

November 21, 2025

1 Introduction

2 Modelling

2.1 Initial Thoughts And Attempts

My first thoughts were that this was either something that could be reduced to a easier special case of the Satisfyability Problem or to a Maximum Flow Problem. I spent an hour on each and reached some conclusions.

2.1.1 Max Flow Problem

My initial draw to Max Flow was due to the fact we'd dedicated a lot of time to it in lectures and the blocking elements in the problems input leant themselves well to a node with no capacity. I tried to map the problem out as a grid of nodes with each node representing a cell and edges representing their adjacencies. From this I noticed that edges also fit with the problems 1x2 "tiles" as each would cover 2 nodes. This gave me confidence that Max Flow could be the solution. However, on further modelling I hit issues as I tried to model flow across the graph and found the rules of flow conservation hindered finding a solution rather than helped. I also struggled to work out where a source or sink would connect to, I considered attaching the source to all nodes but this didnt help either.

At this point I also realised that I could create a row of blocking tiles, cutting the board in half, and the flow could no longer reach the whole board. I considered multiple sources or multiple runs of a solver with different starting locations but felt this was enough of a problem that Maximum Flow couldn't be the solution.

This would later prove to be wrong.

2.1.2 Special Case of The Satisfyability Problem

From this I moved onto the Satisfyability problem. Immediately I could see that edges could be represented as boolean values and a satisfyability clause could be used to relate them and describe a tile. I created a set of clauses to describe

a single tile and decided that if this could be reduced to a 2-SAT, Horn-clause or dual-Horn clause problem, this could be easily solved in linear time. I then set aside two 5 hour sessions to solve this.

After the 2 sessions I had failed to reduce the problem but had worked out it couldnt be 2-SAT or Dual-Horn-SAT in the mapping I had chosen, but could possibly be reduced to Horn-Sat. I had also found another way to map the problem which allowed me to represent cells as two variables rather than four by using one variable to represent whether the connection was vertical or horizontal and another to represent whether it was positive or negative. I got this whilst reading a paper I had found through the wikipedia page for 2-SAT, it described using 2-SAT to generate conflict-free placement of shapes in a space[4]. This paper had futher convinced me that The Satisfyability problem could be used.

At this point I had spent a lot of time on the problem and made little progress towards creating an implementation. I felt I must be missing something so fell back to ask for advice from the lecturer running the module.

2.2 A Change In Direction

Upon visiting the lecturer, he was relatively confused as to why I had focused on The Satisfyability Problem so much and turned me in the correct direction. He mentioned the problem was an edge selection problem and that I should look for an edge selection algorithm, which we had only studied one of. Looking back through my lecture notes I found the problem was incredibly similar to a Maximum Matching problem[1], an edge selection problem that could be solved as a Maximum Flow problem.

Seeing the Maximum Matching problem grouped nodes into two groups, I realised I had been very close previously with the Maximum Flow problem when I thought to apply the source to all cells. Had I thought a bit further about the problem and realised I could alternate the source and sink between cells, I would have had a solution

My initial solution for this then, involved splitting the board into two groups using a checker pattern, then using those two groups as nodes in a bipartite graph for a Maximum Flow problem with capacities of 1 on all edges. During this planning however, I fell into the idea that the graph for a Maximum Matching problem had an infinite capacity on all edges from source or the sink. This was an assumption I made until the final implementation produced some peculiar results. I'm unsure where this confusion came from but my assumption is that I accidentally took the capacities from the Maximum Weight Closure problem.

Once I had my graph designed, I needed to decide an implementation to solve the Maximum Flow problem. Running short on time and confidence due to research wasted on the SAT problem, I decided to implement an Edmonds-Karp solver[3]. This was decided on as it would allow me better efficiencies than the Ford/Fulkerson algorithm with no added complexity and a decent runtime on the small inputs I was providing it.[2]

3 Implementation

The first point I made to myself was that I wouldn't need weighted edges, since all edges had a capacity of one apart from the source and sink edges (as far as I was aware), I could not take capacity into account at all, and simply write edge cases for the source and sink in the residual network calculations. Because of this, I also knew that edges didnt need to hold any infomation so I decided not to create representation of them in my standard graph. I created an Edge class for representing edges in the flow, where I needed to be able to add and remove them individually, but outside of this, the graph was entirely nodes holding references to each other.

Another point I decided on early was that rather than creating copies of networks to represent residuals, I could instead have each node store two sets of references, one to its regular children and one two its residual children. Rather than creating copies, I would instead alter the data in the nodes, understanding I would never need more than one residual network at a time and also understanding that the nodes in both networks would never change. The only major issue this design caused me was a lack of references to parents, meaning parent child relations had to be stored in a Map later, which in the end may have been less memory efficient.

Outside of nodes in the graph the other major concepts I needed to be able to define were flows and paths. Flows I represented as LinkedLists, since I would regularly be adding and removing things from them.

4 Discussion And Conclusion

5 References

References

[1] Thomas Jansen. *Slide 194, Lecture 4, An Example Application: Maximum Matching.*

[2] Thomas Jansen. *Slide 254, Lecture 7, Overview of Maximum Flow Algorithms with Worst Case Runtimes.* A slide showing worst case runtimes.

[3] Thomas Jansen. *Timestamp 37:15 in Lecture 6 is the first mention of this.*

[4] Chi Poon Zhu. *A polynomial time solution for labeling a rectilinear map.* `https://www.sciencedirect.com/science/article/pii/S0020019098000027` .