CS31920 Advanced Algorithms Assignment

Runar Reve

November 2020

Contents

1 Introduction

This is a report about using linear programming(LP) to solve a small single player puzzle game. The game uses a rectangular boards containing square tiles. On the board there is pairs of colours (each colour on the board appears exactly twice). The goal is to make consecutively path between all the pairs of colours simultaneously by following the grid without any of the different paths crossing each other. The rules are: each path can only move directly vertically or horizontally for each tile, and a tile can not be occupied by more than one colours at any time (Therefore no crossing of paths). This problem is similar to the popular game mobile puzzle game "Flow Free"[1].

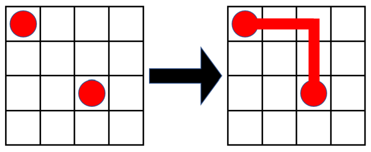
Figure 1: Simple board with valid solution.

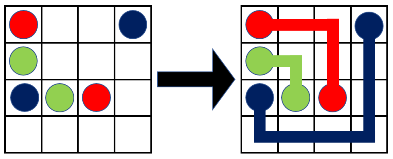
Figure 2: More complicated board with valid solution.

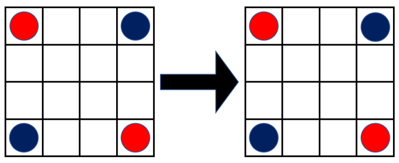
Figure 3: Board with no possible solution.

The data is expected to be given as a raw text file with single digit integers where '0' represent a empty tile, and numbers in the range of '1' to '9' will represent tiles of different colours.

To solve the puzzle with LP I used the programming language C together with the GNU Linear Programming Kit (GLPK)[2, 3] package and the standard libraries of C. Throughout this report I will discuss my modelled and implementation to solve the puzzle, together with discussing my ideas and thought process for my methods.

2 Modelling

2.1 Modelling as a maximum flow problem

To solve this problem for a given input board I have first modeled it as a maximum flow problem. By creating a networks of nodes that represent each tile on the board then connecting the nodes to all the nodes that is directly next to it self with edges. Then by adding a source and a sink to the network and connecting each of them to one of the colours. The capacity over all the edges can be set to anything above '1', but because of the constraint for maximum outgoing flow (2.4.4) the through any edge will never exceed '1'.

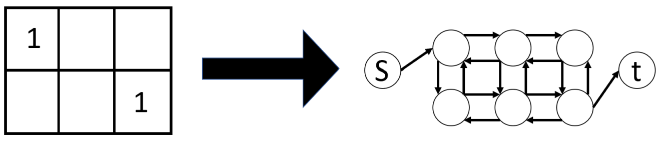

Figure 4: Representation of converting the input board to a network for a maximum flow problem.

2.2 Representing colours

Originally I only had one set of edges for all the colours and using the amount of flow over each edge to identify what colour it represent. This quickly caused problems because the solution often liked splitting the flow between two edges, changing the identifying flow to represent a different colour. There were no way of restricting the value to split or merge colours together, creating a new colour.

To avoid this problem, I used a solution to have one set of edges for each colour on the board for going between nodes of the flow network. Then by restricting that the flow from one colour has to follow the flow conservation over edges of the same set of colour edges(2.4.3). This has solved the problem of two colours directly merging together to create a new colour, but there is no easy way to restrict the flow from splitting in some special circumstances (More information about this splitting problem in 2.8).

When initializing the network I connect each transit node ($node_{transit} \in Nodes \setminus node_{source} \setminus node_{sink}$) with the full set of colour edges. Then I allocate one set of edges for each unique colour and connect from the source to one of the coloured nodes with an edge, and connecting the other same coloured node to the sink node withe the same set of edge.

This makes sure that only the right colour traverses from the start colour node to its end node without ever getting mistaken by another colour. The downside with this is the amount of edges gets multiplied by the amount of different colours. But as the board normally is not extensively large and the amount of colours is caped at 9 colours, which means that we can scarifies some simplicity to make our model more complex by adding more sets of edges between nodes.

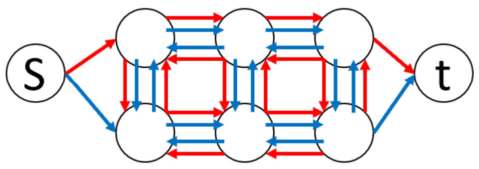

Figure 5: Small graph that visualize a 3*2 input board with two colours as a flow network. Between transit nodes we can see the full set of colour edges, while only one edge representing the connected nodes colour between the source/sink and its corespondent start/end node

2.3 Variables

The only variables used in the constraints for the LP is the edges between the nodes, capacity the capacity for the edges, and the flow over each edge. The constraints will be generated based on the nodes and the structure of the network, but there is no more than the edges, capacity, and flow used for the constraints.

2.4 Constraints

The list underneath of constraints are only set on each of the transit nodes of the given graph. The definition of a transit node is: $node_{transit} \in Nodes \setminus node_{source} \setminus node_{sink}$. The reason source and sink has no constraints is because they are not following any sort of flow conservation as they should be able to generate or destroy flow.

2.4.1 Respects capacities

Each edge in the represented flow network has a upper capacity that the flow over that edge can not exceed. In my current model the upper capacity is set to '1'. This is because the source node is restricted to outputting '1' flow for each colour and there is no need to have more capacity that the edge should not use. This constraint is technically redundant because of the "Max total flow" that restricts the maximum outgoing flow to be '1', but it helps the model create the edges.

$$f_e \leq c(e) \tag{1}$$

2.4.2 Flow conservation

One of the main constraints for a maximum flow problem is that all transit nodes has to conserve the flow of the edges (meaning that the sum of incoming flow has to equal the sum of outgoing flow). The only node that can generate flow is the source node and the sink does can destroy all incoming flow. Because of the next constraint(Colour flow conservation) this constraint is currently redundant. They are both fulfilling the flow conservation of the nodes, but the next constraint is restricting the flow between different colours while this is for all edges connected to each node. I have chosen to not removing it as it is a more comprehensive constraint to represent flow constraint.

$$\sum f(v,\cdot) - \sum f(\cdot,v) = 0 \tag{2}$$

2.4.3 Colour flow conservation

To constrain a flow for a set colour to change to a different colour I have set a flow constraint on all incoming flow from a specific colour edge has to flow out through a edge for the same colour. This is to prevent one colour from "jumping" between different coloured edges.

$$\sum f(v,\cdot) - \sum f(\cdot,v) = 0, \forall f(v,\cdot)_{colour} \cap f(\cdot,v)_{colour} \tag{3}$$

2.4.4 Max outgoing flow

In an attempt to limit the amount of outgoing flow to only contain at most 1 colour, I have this constraint on the outgoing flow of each node to be at most 1. It does not fully prevent that multiple partial flows using a node. Even though this constraint does not fully prohibit two separate colours from through a node, in practise it makes it harder for it to happen and generating more valid solutions.

$$\sum f(v,\cdot) \leq 1 \tag{4}$$

2.5 Objective function

The objective function for the LP is to maximise the flow over all the outgoing edges from the source node. If the maximised flow is equal to the amount of different colours, then that is a indication that it has found a valid solution.

$$Maximise \sum_{v\in V,(s,v)\in E} f_e \tag{5}$$

2.6 Check success

In my model there is two instances it checks if the input has a possible valid solution. The main method is to check if the maximum flow the objective function has maximised is equal to number of different colours on the board.

The second instance will be checked when building the path of the between the sets of colours. If it at any point tries to colour a tile that already contain a path, then it will conclude that there is no valid solution.

2.7 Why this compute correct answer

With the combination of the constraints discussed above, the linear program will compute a valid answer for most inputs, that does not contain too many empty tiles clustered together, and identify when there is no valid solutions. With the colour flow conservation it makes it impossible for one colour to suddenly change. Combining this with the maximum flow out of a node to be no more than 1 limits the ease for two different flows from flowing through the same node simultaneously. These constraint do not fully avoid the flow from splitting over two different edges in some interesting instances as discussed in 2.8. Because of this one problem, LP can cross the paths of two colours and conclude that there is no valid solution, even though there might be a unused path that was not utilized because the linear program used the crossing to calculate the path (As seen in in figure: 7). A possible solution to fix these problems will be discussed in 4.1.1 by running multiple linear programs when crossing is detected.

2.8 Splitting flow problems

In my model there is no way of avoiding the flow from splitting and use the split flow to overlap colours. To fully prevent the flow from splitting I would need to restrict the flow to be binary, either '0' or '1'. Effectively turning it into a integer linear program, which is normally given to be NP-complete[4]. The splitting does not seem to occur for any of the given test input files, as none of these contains enough empty tiles for this to happen.

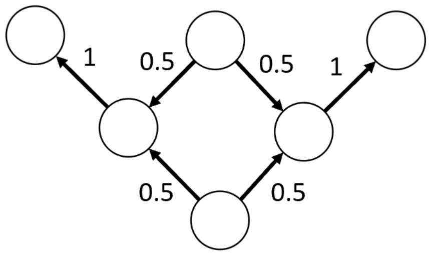

Figure 6: Stripped network example that demonstrates how the linear program can compute a valid solution that follows the constraints are still able to splits and flow through the network.

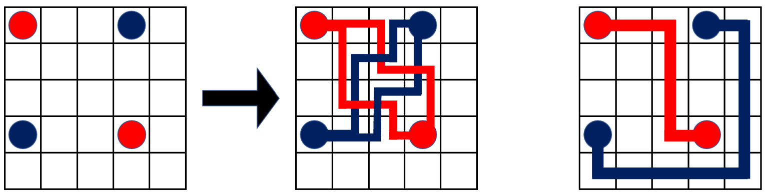

Figure 7: Example of a instance where the model will conclude there is no valid solution because it splits and cross the flow (middle solution), even though there is a valid non crossing path (as seen on the right solution)

3 Implementation

3.1 Board to distance matrix

With the input board that is given to the program I first convert it into a distance graph between each tile of the input and 2 nodes representing the source and sink. The distance in this case will represent the capacity on the edges between two nodes which I have set to 1 for all. The capacity is set to 1 because each source node will only produce 1 flow for each colour and this will make it easier to validate if there exists a valid solution. For every node in the graph the outgoing edges from the node is on the row of the node, while all incoming edges are on the column of the node.

3.2 Set up the problem

With the distance graph created in the previous section, it makes it manageable to iterate through the distance matrix and add the constraints and rules for that GLPK can understand and use to solve the problem. In my implementation I have separated the various stages of formulating the LP to make it more streamlined. This makes it terribly inefficient, because it has to loop through the large distance matrix several times to set up the LP and can be optimised to reduce the time complexity of the program. But in my experience the increased time this results in is minuscule with the data we expect and the structure one get by splits the program into section makes it more comprehensive to understand and editing sections.

3.3 LP solution too board

With the solution that the LP has maximised I loop through all the edges to retrieve the flow over the edge. By observing what colour the edges with flow is corresponding to we can colour the tile the destination node is representing. If at any point it tries to colour a tile that is previously occupied, it will output that there is no valid solution. The reason there can be multiple different flows to same node is because of splitting flows discussed in section 2.8.

4 Discussion and conclusion

After using LP for this assignment I have seen the strengths and weaknesses of converting a problem into LP and solving it that way, instead of writing a algorithm to solve the initial problem. The most difficult thing I found was to wrap my head around how LP actually works and why some restrictions do not work. One of my biggest problem was that there is no way I could find to represent a "if-statement" to restrict split flow from multiple different edges.

4.1 Improvements

There are many implementation improvements that I would have liked to optimise if I had more time to and the know-how to implement. In the current implementation for every time the program does something with a specific edge it has to loop through the distance graph and count the edges to get the index of edge. This could be fixed by storing the index of the edges in a separate array or in the distance graph instead the capacity (since all capacities are set to 1).

4.1.1 Alternative path when crossing

A possible implementation to make sure there is no other possible solution to the problem, seen in figure 8, could be to run multiple LPs. In stead of concluding there is no possible solution when multiple colours tries to occupy a single tile, we create multiple new instances of the graph where we removed a full path from the colliding colour to rerun the new sub-problem. There are multiple possible ways of selecting what path to use to remove nodes, either picking a path arbitrarily, shortest path, or rerun for each possible path. If one of the new runs are able to find a path we can use that path to generate the solution.

In the case where there are more than two colours overlapping I believe this approach can still be used, but would make the implementation a bit more complicated. When creating new instances one should remove one colour at the time until possible solution is found or one is more sure there actually is no solution.

With this method there most likely will still be some small uncertainty when declaring there is no solution. There is a chance that a solution computed paths that block each other, so removing either path would split source and sink. To solve this one would need to alter the path, but at that point you are getting close to brute forcing a solution by randomly changing the path.

4.2 Self evaluation

I am quite happy with my approach and outcome of this assignment. I were able to get a successfully implement more features than I expected. My original goal for this assignment was to be able to understand the basics on how to implement a maximum flow problem in linear programming, but at

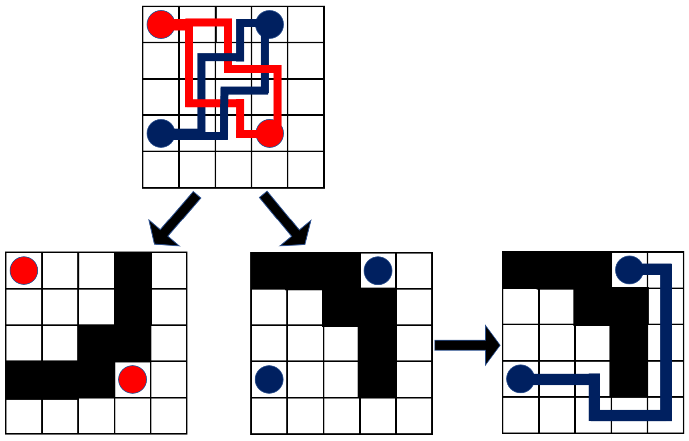

Figure 8: An example to find a non detected exciting path when the multiple paths crossed by running new maximum flow where we remove nodes used for other colours. Start solution is from figure: 8

the end I were able to successfully implement other constraints and improve the output for a more correct answers.

I am still unsure if linear programming is the best method of solving this exact problem, as it is, to my knowledge, not known how to avoid the flow splitting problem without increasing the time complexity drastically. Maybe using a maximum flow algorithm would produce better answers, but I am unsure if our limitations of splitting would fully disappear, or it there would be any new unforeseen obstacles. But I can definitely want to explore other uses for linear programming that I would like to experiment with.

If there is something I am not that satisfied, it would be my implementation. The is not always optimal structured and would like to move many of the steps into separate functions and reduce the amount of loops that is being used to make the code more comprehensive. As I started with the implementation I was not to comfortable with C and did not prioritise time on optimising it after getting more grasp on it.

4.2.1 Self evaluation marks

67/100	**Total mark of self evaluation**
9/10	Following formatting guide
7/10	Introduction
14/20	Modeling
6/10	Implementation
9/10	References
10/10	Correctly handles known files
5/10	Correctly handles additional files
6/10	Readability of code (useful comments)

In my opinion I have in general done a good job with this assignment, there is of course many areas that could be improved with more time and understanding. But overall the modeling calculates adequate results, except for some special instances, for the given test data, and I would expect it to at least be able to solve 50% of additional inputs. As discussed earlier I know that my code could be improved in structure and comments to be more readable. I believe that I have followed the structure expected in the report and somewhat been able to explain my models, ideas, processes in an understandable way. But of course I know most areas could be improved.

References

[1] Big Duck Games LLC. *Flow Free (version 4.8)*. [Mobile app]. Accessed: 24/11/2020. 2012.

[2] Andrew Makhorin. *: GNU Linear Programming Kit Reference Manual for GLPK Version 4.65.* URL: `https://www.gnu.org/software/glpk/`. Accessed: 24/11/2020.

[3] Andrew Makhorin. "Modeling language gnu mathprog". In: *Relatório Técnico, Moscow Aviation Institute* 63 (2000).

[4] Christos H Papadimitriou. "On the complexity of integer programming". In: *Journal of the ACM (JACM)* 28.4 (1981), pp. 765–768.