335 — Algorithm Engineering — Project #3 Knights Max Flow

Project #3 – Knights Max Flow

Introduction

This is the Knights Max Flow problem. You try to find a good placement of "Knight's move" (kt-move) edges connecting pairs of grid cells into a flow graph. Each placed edge's flow capacity depends on the grid cells it connects. In effect, you construct the best graph you can on top of the grid.

The Grid

The whole cell grid is 10x10 (indexed from 0). Each grid cell needs to be big enough to show its 2-digit "flow capacity". (Maybe a 20x20 pixel cell size is good.) Edge capacities are computed from cell capacities.

The sole source cell is at (1,2). The sole sink cell is at (8,7). Each cell is given a random even number in the range 0..30 as its flow capacity. The number is even so that the derived edge capacities are integers.

The Edges

Each edge is a knight's move away from the edge's starting cell horizontally and then diagonally to the target cell. The distance between end-cells is sqrt(5) cell lengths, center to center. (Check Wikipedia, maybe.) When you place an edge, you should draw the edge from the center of the starting cell to the center of the end-cell.

You have a small pile of kt-move edges to use. The size of that pile is a random int in the range 15..30. After an edge is placed, you can compute its flow capacity as the average of the two flow capacities marked on those end-cells. This will always be an integer. Also, when you compute a flow through the graph, you should indicate the amount of flow across each edge along with its capacity. (A cell can display as a dot/circle, maybe.)

Edges can cross; that is, the graph need not be planar. You do not have to place all edges; sometimes it doesn't matter. Your algorithm can change the placement of an edge; it can "change its mind". If an edge is moved, you may feel the need to clean up the display after that edge is removed and before it is again placed.

The Solution

You are trying to create a graph with the maximum possible flow. As you know, given a graph you can use the Ford-Fulkerson (Edmonds-Karp) algorithm to find the max flow for a given graph. But you don't yet know the best graph for the given grid. So you are stuck trying graphs.

You don't have to try all possible graphs, brute force. (That may take awhile.) If the flow is not maximal, that isn't so bad, but your algorithm should try to do a good job. (We don't know whether this problem is NP-Hard or not.) At least, your algorithm should be as good as some kind of greedy algorithm. (It can be stupid as well as greedy.)

For example, from the source and sink cells, you could try to extend two unconnected sub-graphs, one from the source cell and one from the sink. Eventually, these would connect via some edge placement.

To extend each sub-graph, you could pick the biggest available edge flow by picking the best spot to connect to. If you run out of edges, you could consider moving a weak (weakly performing flow) edge to a new spot. Eventually, your two sub-graphs (from the source and from the sink) should get connected. From there, you can consider whether you can improve the flow by moving a weak edge.

As an alternative, you could start by creating a single arbitrary path from the source to the sink and then attempt to augment (fix-up) that path to get a better flow value.

To simplify your display, you could instead consider planar graphs only; but this may complicate your algorithm.

Display

You should display the grid, the cell capacities, a count of the unused/unplaced edges, and a current reasonable flow value from the source to the sink. If the source is not yet connected to the sink, this flow is 0. We should be able to tell what a cell's capacity is. You may want gaps (gutters) between the rows and columns of a grid to nicely show some of these values. You can show cell values and edge values “near” them.

Algorithm

You should prepare a 1-page (preferably, at most) paper describing your whatever algorithm(s) you use. You don't have to discuss the display; just how you choose edge placement, graph flow capacities, and how you compute reasonable flow values. All of this at a high level of abstraction (again, to simplify your life).

Complexity Order

You should prepare a 1-page (at most) paper describing your analysis of the Big-O running time of whatever algorithm you use. Address the usual issues such as main operations, input size, etc. (Assume we could use a bigger grid and more edges, hence a more complex graph.)

Team

The team size is the same as before, but you can change team members from the previous project if you wish.

Academic Rules

Correctly and properly attribute all third party material and references, if any, lest points be taken off.

Project Reports, Readme File, Submission, and **Grading**

As before.