University *of Ljubljana*
Faculty *of Mathematics and Physics*

Department of Physics

Distribution of Data Transmission in a Randomized Computer Network

Final Assignment for Model Analysis 1, 2023/24

Author: Marko Urbanč
Professor: Prof. Dr. Simon Širca
Advisor: doc. dr. Miha Mihovilovič

Ljubljana, August 2024

Contents

1 Introduction

The scope of computer networks has been expanding rapidly in the past few decades. What once were simple networks of interconnected computers have now evolved into complex systems that are used for a wide range of applications. It makes sense then for one to plan and analyze the flow of data in such networks to ensure that they are efficient and reliable and thus cheaper to maintain.

As my final assignment for Model Analysis 1, I opted to simulate the flow of data in a randomized computer network. The network consists of a $N \times N$ grid, where each unit of the grid can be a server, a user or a wire used to connect the two. Each of these units is connected to its four nearest neighbors and has a value associated with it that specifies it's bandwidth. For wires this is the maximum data throughput, for servers it is the maximum data processing rate and for users it is the maximum data consumption rate. We'd like to find the distribution of server-loads for a grid of wires with random bandwidths. It is possible to study many different types of server and user placements but the most interesting one to us will be where we have the users and servers placed on the top and bottom rows of the grid, respectively.

We can solve for the flow rates through the network by solving a set of constrained linear equations. This field of study is known as linear programming and is a powerful tool for solving optimization problems. We've already seen some linear programming use in the second task of this course `mod102` where we created a dietary plan from a set of given foods based on their costs and nutritional values. For the sake of completeness it makes sense to quickly go over the basics of linear programming before we proceed with the simulation. Linear programming uses linear optimization to find the maximum or minimum of a linear function subject to a set of linear constraints. This function is known as the objective function and is quite analogous to the cost function we've seen mentioned in the world of Machine Learning. The constraints are linear inequalities or equations that define the feasible region of the problem. Mathematically formulated we can consider a cost function $f(x)$ and a set of constraints defined as:

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) &= c_1 x_1 + c_2 x_2 + \cdots + c_n x_n \,, \\ a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n &\leq b_1 \,, \\ a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n &\leq b_2 \,, \\ &\vdots \\ a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n &\leq b_m \,. \end{aligned}$$

The goal is then to find the values of $x_1, x_2, \ldots, x_n$ that maximize or minimize the cost function $f(x)$ while satisfying the constraints. The feasible region is the set of all points that satisfy the constraints and the optimal solution is the point in the feasible region that maximizes or minimizes the cost function. That is all that is necessary from a mathematical perspective. Of course the actual implementation of solvers for such problems are much more complex and involve a lot of optimization techniques but that is not the focus of this assignment.

2 Task

The original text of the assignment reads as follows:

> **Razporejanje prenosa podatkov po omrežju:** Za model omrežja vzemi $N \times N$ kvadratno mrežo, vsak rob pa ima naključno maksimalno hitrost povezave med 0 in 1. Vozlišča na zgornjem robu so internetni odjemalci, spodnji rob pa so strežniki. V notranjih vozliščih velja 1. Kirchhoffov zakon. S pomočjo linearnega programiranja določi, kolikšne hitrosti prenosa imajo strežniki in odjemalci, ko je skupna hitrost prenosa največja. Ker gre za naključna omrežja, si oglej tudi statistično porazdelitev zanimivih količin.

Translated to English this reads:

> **Distribution of Data Transmission in a Network:** Take a $N \times N$ square grid as the network model, where each edge has a random maximum connection speed between 0 and 1. The nodes on the top edge are internet clients, while the bottom edge are servers. In the internal nodes, Kirchhoff's first law holds. Using linear programming, determine the transmission speeds of the servers and clients when the total transmission speed is maximized. Since these are random networks, also examine the statistical distribution of interesting quantities.

3 Solution Overview

I took upon the task of simulating the flow of data in Python using `PuLP` [1] as my choice of linear programming library among others such as `numpy`, `scipy` and `matplotlib`. I'm aware that more complex network simulation packages exist for Python, such as `networkx`, but I wanted to keep the simulation as simple as possible and home-brewed. To start I created the base classes, `User`, `Server` and `Wire` that represent the nodes of the network. Among other things each of these classes has a `bandwidth` attribute that specifies the maximum data throughput, processing rate and consumption rate, respectively. The other attributes determine the position of the node in the grid, its ID etc. All these classes are slotted into a `Grid` class that represents the network as a whole. The `Grid` class contains the necessary methods to setup the network, solve the linear problem and extract or plot the results.

I think it makes sense to go over the various constraints I've imposed on the network. The most challenging part of setting the constraints was my want to use the absolute values of the flow rates in the linear equations. This becomes a problem because the absolute value function is not linear and thus not directly compatible with linear programming. To get around this I've used the following trick. I've introduced two new variables ρ^+ and ρ^- that represent the positive and negative parts of the flow rates, respectively. Both of these quantities are non-negative and their sum is equal to the absolute value of the flow rate, such that $\rho = \rho^+ - \rho^-$. This way I can rewrite the absolute value of the flow rate as a linear combination of ρ^+ and ρ^-.

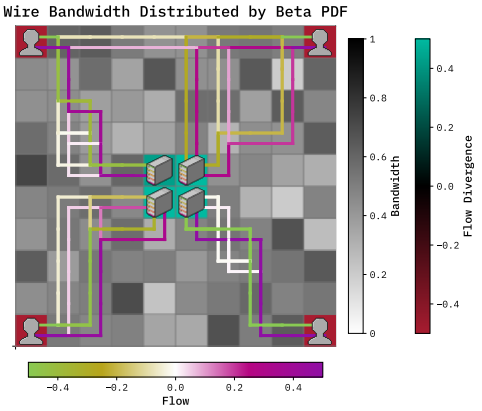
The next step was to impose constraints for the flow rates through the network. The flow rates are determined by the bandwidths of the wires, servers and users. Wires are allowed to transmit data in both directions thus the absolute value of the flow rate cannot exceed two times the bandwidth of the wire. The sum of the flow rates into a Wire node must be equal to the sum of the flow rates out of the Wire. Servers and Users are special cases since they cause a divergence in what we could call the flow field, if we had a continuous field. The flow rate into a Server is strictly zero (thus $\rho^- = 0$). Likewise the flow rate out of a User is also strictly zero. The maximum flow rate into a User is determined by its bandwidth. The way I've set up the constraints is that the users demand the maximum amount of data they can consume unconditionally while the servers are allowed to process data at a rate that is less than or equal to their bandwidth.

The objective function of the linear problem is to satisfy the demands of the users while minimizing the flow rates through the network and minimizing server loads. The flow rates are minimized by minimizing the sum of the positive and negative parts of the flow rates. The server loads are minimized by minimizing the sum of the flow rates into the servers. The objective function is thus a linear combination of the flow rates and server loads.

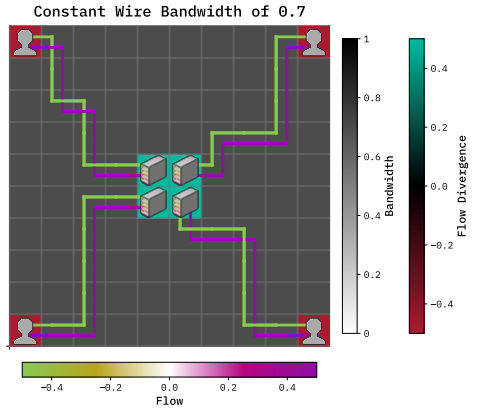
The way the problem has been set up results in two sets of data for flow rates, one for the positive part and one for the negative part. The two are equal in magnitude but opposite in sign and we can

consider the real solution to be one or the other. The flow rate divergence is thus given by the sum of the two. The term flow rate divergence is used here to describe data that is either consumed or produced by the network. The flow rate divergence must thus be zero at all nodes except for the Users and Servers.

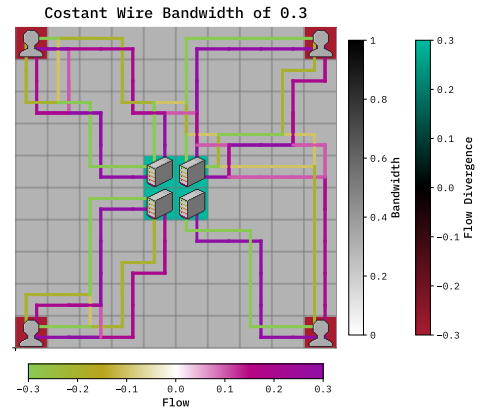
To stick with the purpose of having a home-brewed simulation I also drew some icons for the Users and Servers in a pixel art style. From what we've specified and implemented so far we can solve for the flow rates through an arbitrary network. I've drawn two examples for arbitrary grid configurations on a 10×10 grid. I've also tried to show the effect of changing the way the bandwidths of wires are sampled. The results can be seen Figures 1a, 1b, 2a and 2b and Figures 3a, 3b, 4a and 4b. In all examples each of the users demanded a flow rate of 0.5 while each server could supply a flow rate of 1.



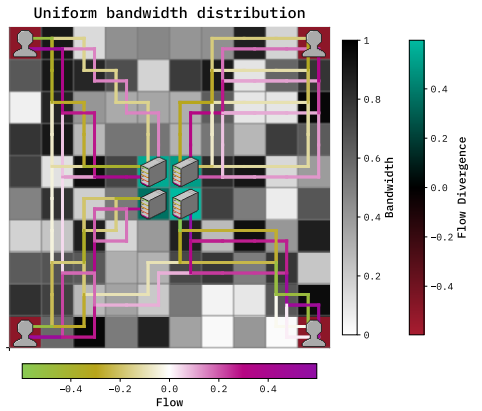
(a) Flow rate in a 10×10 grid with wires having bandwidths sampled from a beta distribution with $\alpha = 10$ and $\beta = 10$.



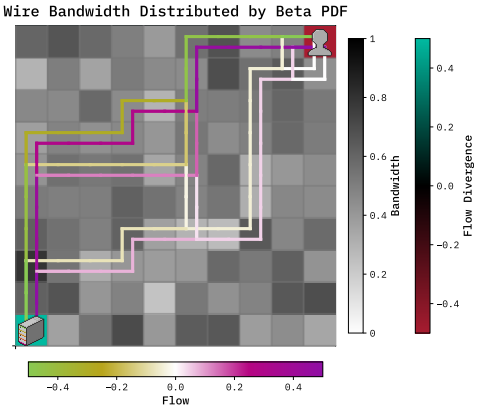
(b) Flow rate in a 10×10 grid with wires having constant bandwidths of 0.7.



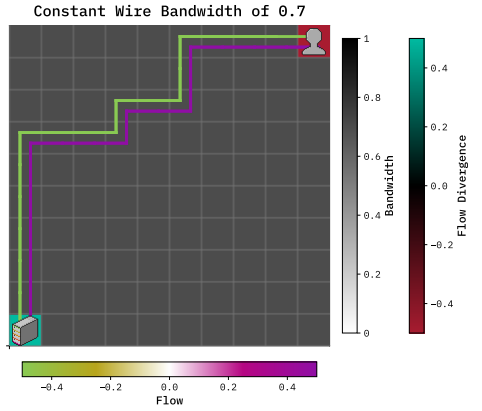
(a) Flow rate in a 10×10 grid with wires having constant bandwidths of 0.3.



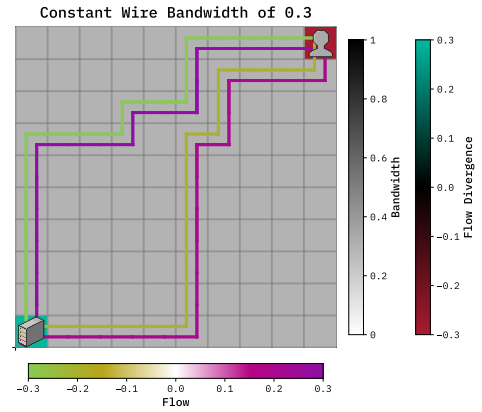
(b) Flow rate in a 10×10 grid with wires having bandwidths sampled from a uniform distribution between 0 and 1.



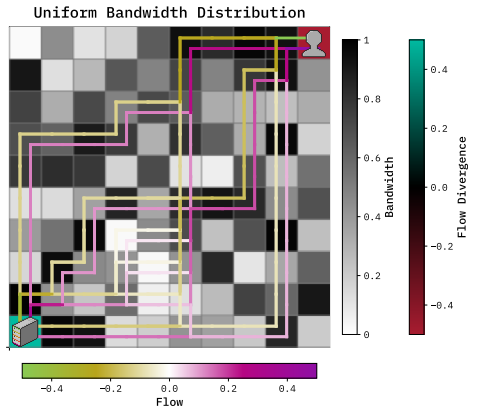
(a) Flow rate in a 10×10 grid with wires having bandwidths sampled from a beta distribution with $\alpha = 10$ and $\beta = 10$.



(b) Flow rate in a 10×10 grid with wires having constant bandwidths of 0.7.



(a) Flow rate in a 10×10 grid with wires having constant bandwidths of 0.3.



(b) Flow rate in a 10×10 grid with wires having bandwidths sampled from a uniform distribution between 0 and 1.

I think that the effect of changing the distribution of the bandwidths is extremely clear as it is well pronounced in the results. It has a very large impact on the flow rates and path choice through the network. The beta distribution with $\alpha = 10$ and $\beta = 10$ is very skewed towards the higher bandwidths and thus the flow rates are very uniform. The uniformly distributed bandwidths on the other hand yield a very chaotic network with a lot of wires needed to carry the data. The constant bandwidths are there to illustrate the effect of a clean grid with no variation in bandwidths and how additional paths are added if the individual wires cannot carry the data. Personally I think the patterns are very pretty and I'm happy with the results given by my implementation. I will not claim that it was effortless to set up properly.

4 Results

First I ran the simulation for a 30×30 grid with wires having bandwidths between 0 and 1. It is important to note that the bandwidths were not sampled uniformly but rather from a beta distribution. The beta distribution is a continuous probability distribution that is defined on the interval $[0, 1]$. It is a good choice for this problem since we can easily skew the distribution towards the lower or higher bandwidths. Having a uniform distribution yielded grids that were infeasible to solve. The Figure 5 shows the probability density function of the beta distribution for different values of the shape parameters α and β. The shape parameters determine the skewness of the distribution.

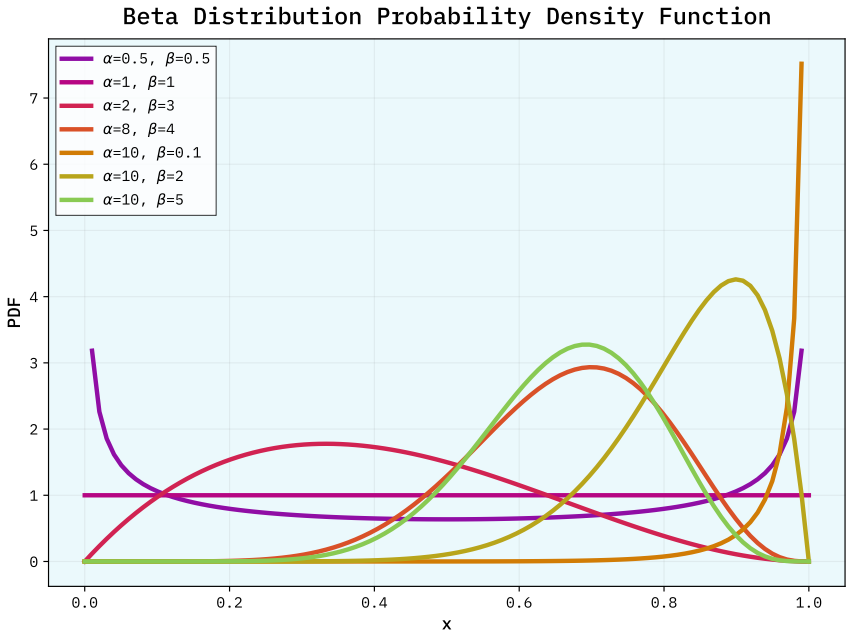


Figure 5: Probability density function of the beta distribution for different values of the shape parameters α and β.

We can see the results of a basic simulation for a 10×10 grid in Figure 6. The wires in this grid have bandwidths sampled from a beta distribution with $\alpha = 10$ and $\beta = 2$. For illustrative purposes I've colored the nodes based on their flow rate divergence and plotted both the positive and negative parts of the flow rates.

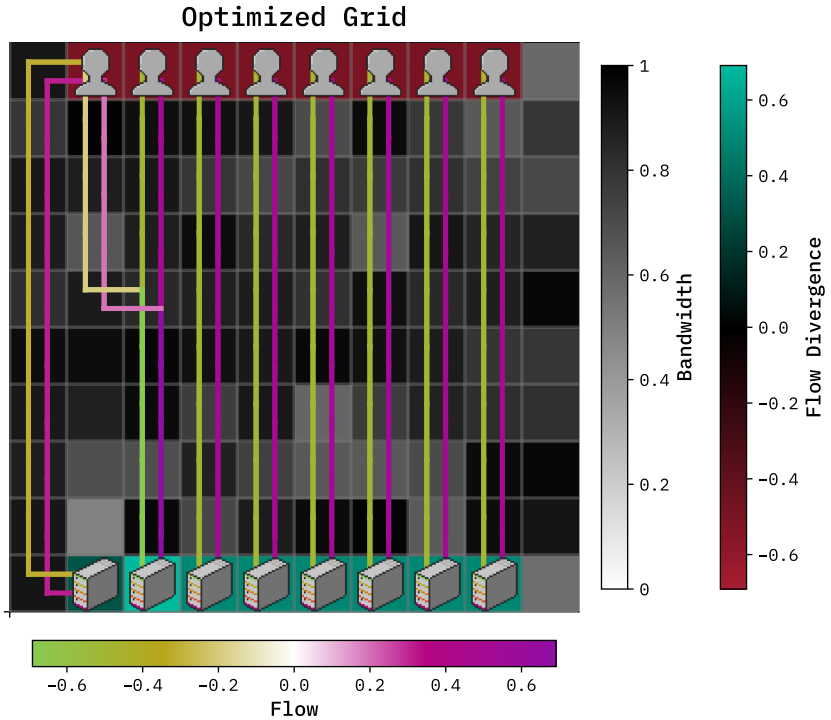


Figure 6: Flow rate in a 10×10 grid with wires having bandwidths sampled from a beta distribution with $\alpha = 10$ and $\beta = 2$.

We can see that the flow rate divergence is zero at all nodes except for the Users and Servers. The flow of data goes almost as linearly as possible from the Users to the Servers, however due to the objective functions demand to minimize server loads the flow rates are not uniform. We can see on the left-hand side of the grid that the flow is higher for the second server and lower for the first. The first user receives

data from both these servers as a portion of the data is carried over from the second server. This is likely due to wires in the direct path to the first server having lower bandwidths, thus making it cheaper to carry the data over from the second server and to place the original connection to the first user slightly to the side. The same results can be seen for grids of larger sizes. I've plotted the results for a 30×30 grid and a 100×100 grid in Figure 7. The bandwidths of the wires here were also sampled from a beta distribution with $\alpha = 10$ and $\beta = 2$.

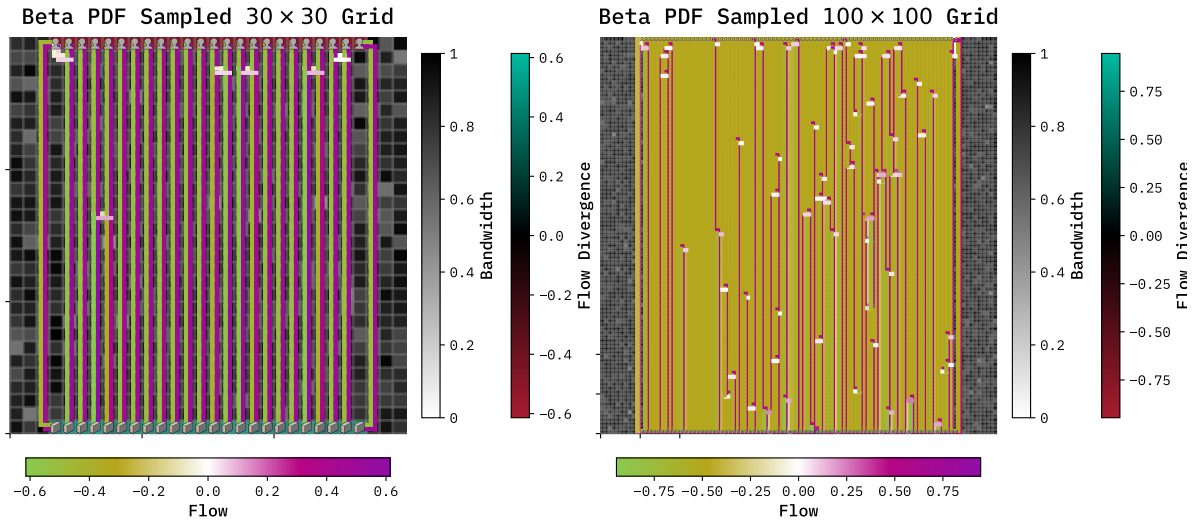


Figure 7: Flow rate in a 30×30 and 100×100 grid with wires having bandwidths sampled from a beta distribution with $\alpha = 10$ and $\beta = 2$.

No lie, that the individual paths here are much harder to see but the general flow of data is still very clear. I'd like to point the readers attention to the small white bridges that form in both cases but are more prevalent on the larger grid. These are very small almost zero flows that probably happen as an artifact of the method. I've added an ε tolerance to the flow rates to avoid numerical instability but these were just slightly above the threshold. I kept them regardless to show that the model does also make some nonsensical connections with extremely small flow rates.

The main result I'd like to present is the distribution of the server loads for a 30×30 grid with wires having bandwidths sampled from the beta distribution for different shape parameters. I've also added a special case where the user demand is lower at 0.1. The results can be seen in Figure 8. The plot was created by solving a 30×30 grid 100 times for each shape parameter pair.

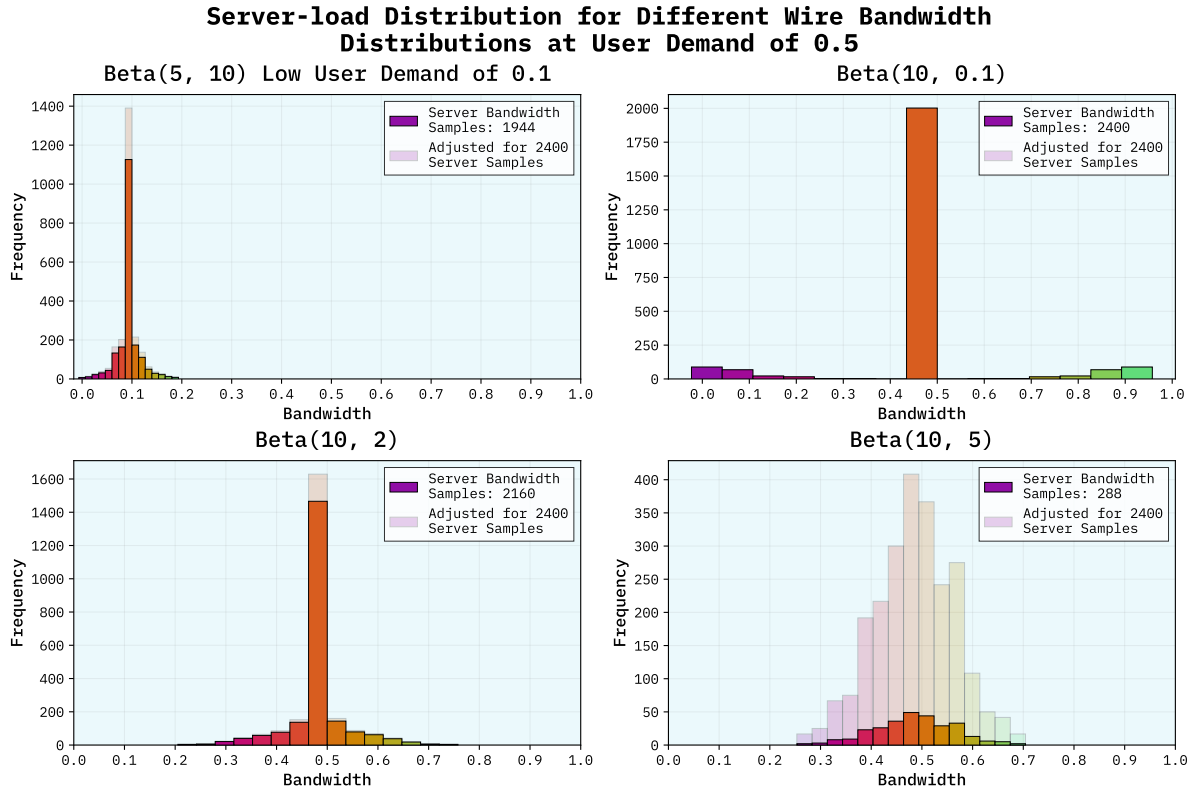
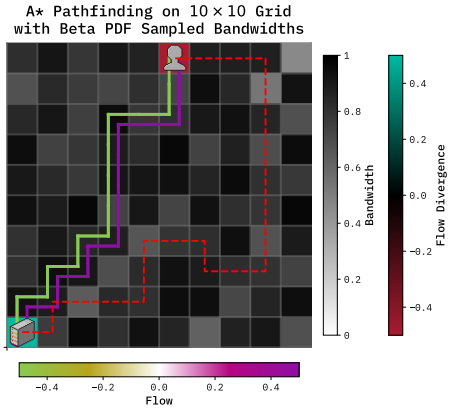


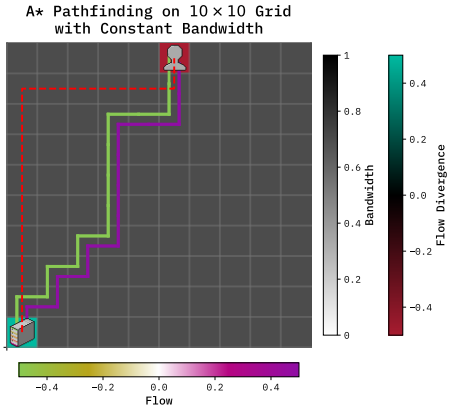
Figure 8: Distribution of server loads for a 30×30 grid with wires having bandwidths sampled from a beta distribution with different shape parameters.

Findings from this plot can be somewhat inconclusive however as there is really not much of a trend to compare. In addition, sampling from a less skewed distribution has a large impact on the feasibility of the grid. This infeasibility only grows with the size of the grid. I've tried to do some rudimentary correction to the histograms by rescaling them to the proper number of samples however this can definitely lead to false results especially in the case of the lower rightmost plot. The shape of the histogram is only for those configurations that were feasible. There's no telling how the shape could change were the other configurations also feasible. I wonder if I have perhaps misunderstood the instructions since the majority of large grids are infeasible to solve with random bandwidth sampling for the wires. What we can see is that the more skewed the distribution of bandwidths is to 1 the more we see a *delta* function like distribution of server-loads centered around the bandwidth the users demand. More uniform wire bandwidth distributions lead to a more uniformly distributed server-load with a peak at the user demand. I find it interesting that in the case of a very very skewed beta distribution, as seen in the upper rightmost plot, server-loads slightly rise on the extreme ends of the bandwidth constraints. This is likely due to the fact that the objective function also tries to minimize the server-loads and it appears that having some servers work harder and others rest is cheaper than having all servers work at the same rate.

I wish I had more time to do further analysis on the model but I'm critically out of time. I find the report somewhat disappointing as I feel that I could have done more with the model and everything seems somewhat uninspired. I tried to calculate network relevant quantities such as the *ping* time between the Users and Servers and *jitter*, which is the variation in the ping time but I had big problems with implementing proper pathfinding based on the results of the LP problem. Looking back I think it could have been easier to just calculate some of these quantities directly by hand. Figures 9 and 10 show the results of the pathfinding algorithms $A*$ [2] and *Dijkstra's* Algorithm [3] for the 10×10 grid again with wires having bandwidths sampled from a beta distribution with $\alpha = 10$ and $\beta = 2$ or set to a constant value of 0.7.

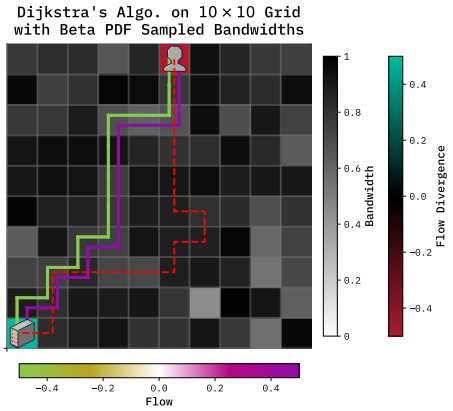


(a) Pathfinding with the A* algorithm for a 10×10 grid with wires having bandwidths sampled from a beta distribution with $\alpha = 10$ and $\beta = 2$.

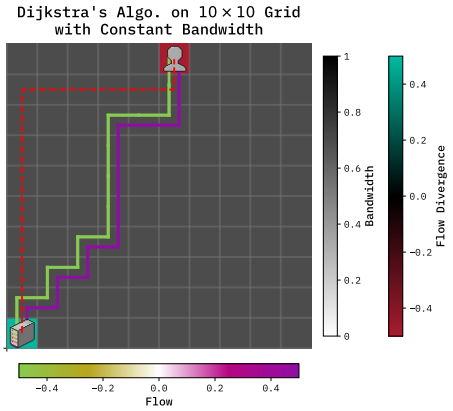


(b) Pathfinding with the A* algorithm for a 10×10 grid with wires having bandwidths set to a constant value of 0.7.

Figure 9: Demonstration of the A* pathfinding algorithm.



(a) Pathfinding with Dijkstra's algorithm for a 10×10 grid with wires having bandwidths sampled from a beta distribution with $\alpha = 10$ and $\beta = 2$.



(b) Pathfinding with Dijkstra's algorithm for a 10×10 grid with wires having bandwidths set to a constant value of 0.7.

Figure 10: Demonstration of Dijkstra's pathfinding algorithm.

The `A*` algorithm used the wire bandwidth as the path cost function and the `Manhattan distance` as the heuristic. `Dijkstra's algorithm` used the wire bandwidth as the path cost function and has no heuristic. I'd love to go into more detail of these pathfinding algorithms but I hardly see any point in doing so as I've not been able to use them for any useful results. The paths they find are not the same as the ones the LP problem finds and I'm not sure how to reconcile the two.

As a sort of consolation prize I've calculated the LP problem total and solve time for different grid sizes. Each grid was attempted to be solved 10 times. The total time is the time needed to solve the full problem including it's setup while the solve time is only the time needed to run the LP problem solver once the problem has been setup. The results can be seen in Figure 11.

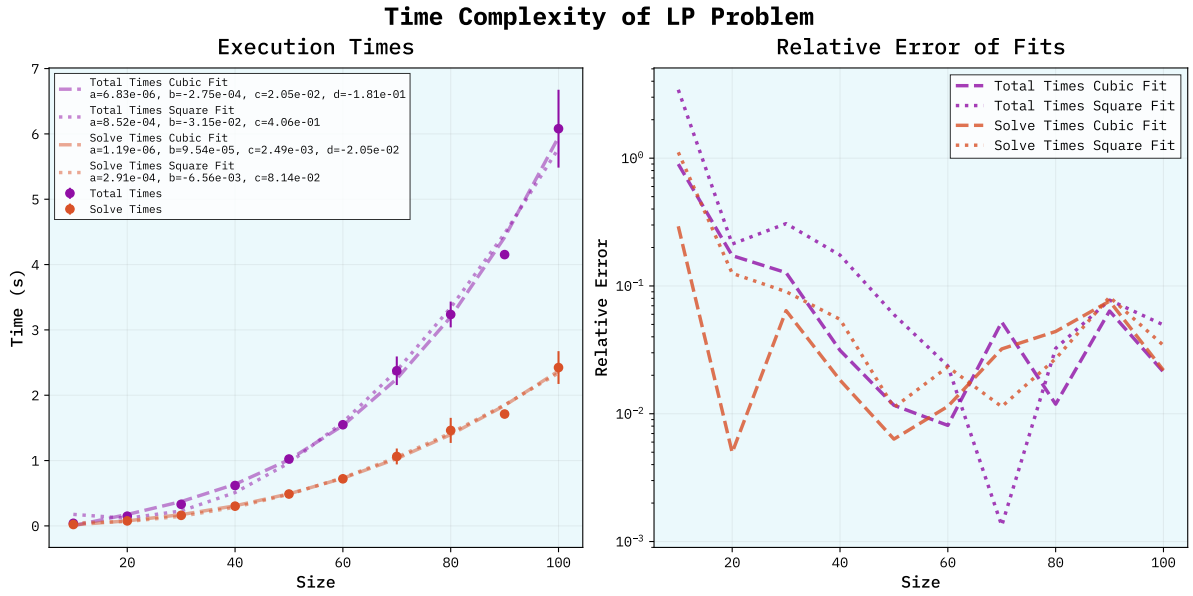


Figure 11: Total and solve time for different grid sizes.

I've tried to fit a square and cubic polynomial to both times and I think it's safe to say that the dependence of the solve/total time on the grid size is at least quadratic. To close off the report I tried to improvise a bit and calculate the *ping* and *jitter* by simply taking the sum of all the flows on the grid and dividing it by the number of nodes. The jitter is then the standard deviation of the flows, while the ping could be shown to be related to the average flow. The results can be seen in Figure 12.

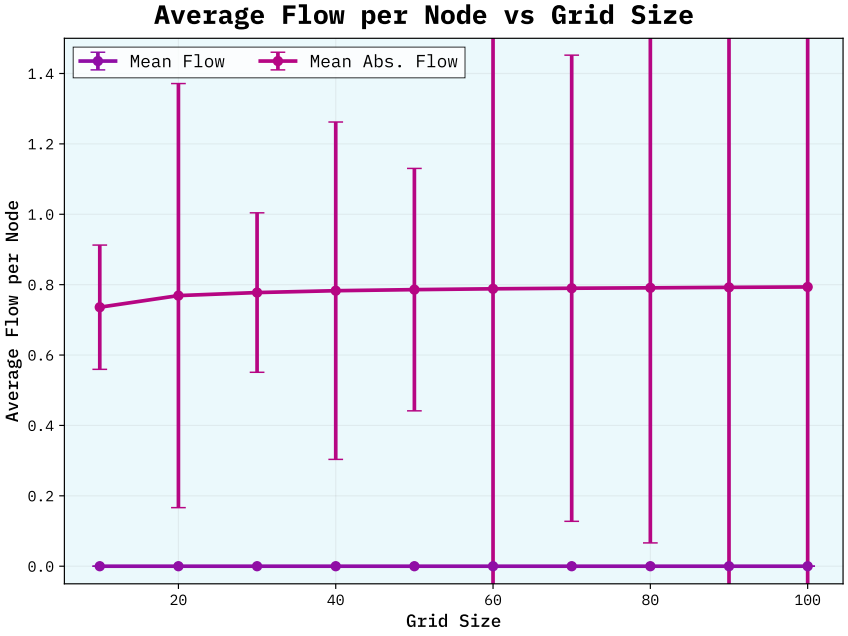


Figure 12: Flow sum (ping) and standard deviation of the flow (jitter) for different grid sizes.

We can see that the mean flow is constantly 0 which is exactly as expected since it is the sum of both the negative and positive parts we've defined earlier. The absolute value of the flow is then the sum of the two and it seems to tend to a constant value of 0.8. That would mean that the average flow rate for either the positive or negative part tends to 0.4. This seems to be a good sign, since it means that the network seems to be balanced regardless of it's size. The jitter however is quite horrible and seems to grow with the size of the grid. To analize this better I'd need more results for better statistics since I have a feeling that out of the 10 attempts to solve for each size, the larger grids did not have as many feasible solutions.

5 Conclusion and Comments

References

[1] Iain Dunning and et al. Pulp: A linear programming toolkit for python. *PuLP: A Linear Programming Toolkit for Python*, Sep 2011.

[2] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[3] Donald E. Knuth. A generalization of dijkstra's algorithm. *Information Processing Letters*, 6(1):1–5, 1977.