Robust Maximum Network Flows

Daksh Aggarwal

Project Report for CSC-395, Fall 2020

1 Introduction

This project studies the problem of finding maximum network flows from the perspective of robustness and adaptivity. Understanding robust versions of classical network problems is an important task because most real-world networks−especially those for which we might desire to compute network flows−are susceptible to failure and uncertainty. One approach to coping with this reality would be to employ an optimal classical solution and hope that the solution retains its optimality (or, at least is not far from optimal) even after the network structure or parameters undergo small changes. However, as research on robust optimization has shown [3, 6], this hope is rarely realized and the classical solutions perform very poorly in comparison to robust ones. Moreover, a host of different applications can be formulated into network flow problems, which makes it all the more imperative to understand their robust solutions. We chose to specifically explore maximum flow since it is one of the most basic network flow problems and frequently arises in the real world in different shapes and forms.

We first briefly introduce the classical maximum flow problem.

1.1 Maximum Network Flow Problem

Let $G = (V, E)$ be a directed network with a *capacity* $c_{ij} \in \mathbb{R}_{\geq 0}$ associated with each edge $e = (i, j) \in E$. To each edge $(i, j) \in E$, we assign a *flow* $x_{ij} \in \mathbb{R}_{\geq 0}$ not exceeding its capacity c_{ij}. If we denote the in-neighbors and out-neighbors of node i by $N^-(i)$ and $N^+(i)$ respectively, we see that the net flow x_i^* into a vertex i is

$$x_i^* = \sum_{j \in N^-(i)} x_{ji} - \sum_{j \in N^+(i)} x_{ij}.$$

Now, we would like to fulfil a flow from some node s to another node t in V. The *maximum flow problem* seeks to find a flow in G such that $x_s^* = -x_t^*$ and x_s^* is maximized with the constraint that no flow be lost at any intermediate node:

$$x_i^* = 0 \text{ for all } i \in V \setminus \{s, t\}.$$

Apart from obvious applications to tasks such as transportation and supply scheduling, the maximum flow problem frequently arises in a form that does not seem conformant to the above structure, including the feasible flow problem, the matrix rounding problem, and the balancing of political representatives (see [1, Ch. 6] for more information).

Finding maximum network flows is strongly related to understanding minimum cuts in a network. A *cut* is a partition of the vertex set V into S and $\overline{S} = V \setminus S$, for any $S \subseteq V$. For nodes $s, t \in V$, an *s-t cut* is a cut specified by S such that $s \in S$ and $t \in \overline{S}$. If we denote by $N^+(S)$ the set of edges $(i, j) \in E$ such that $i \in S$ and $j \in \overline{S}$, then the *capacity* $c(S)$ of the *s-t* cut S is

$$c(S) = \sum_{(i,j) \in N^+(S)} c_{ij}.$$

The minimum *s-t* cut problem seeks to find an *s-t* cut S that has minimal capacity $c(S)$.

Without much difficulty it can be observed that any *s-t* flow has a value less than the capacity of any *s-t* cut (e.g., [1, pp. 179]. But, in fact, it can be shown that the maximum flow matches up to the minimum cut capacity:

Theorem 1 (Max-Flow Min-Cut Theorem [1]) *The maximum value of the flow from a node s to a node t in a capacitated network equals the minimum capacity among all s-t cuts.*

Theorem 1 is a special case of *strong duality*, a general result from optimization theory [1, Ch. 9]. Most algorithms that solve the maximum flow problem take advantage of the observation that the max flow must be the min cut capacity to prove their correctness. The basic approach to solving the max flow problem is the Ford-Fulkerson algorithm that iteratively increases the flow along *s-t* paths in an *s-t* flow till no more paths can be augmented, signalling the max flow has been achieved [1, Ch. 6, 7]. Refer to [13] for a historical overview of the origin and solution of the classical maximum flow problem.

2 A Robustness Approach to Max Flows

The need for a robust approach to the max flow problem is demonstrated forcefully by the experimental paper [3] of Ben-Tal and Nemirovski, that checks the reliability of classical solutions of 90 problems from the `Netlib`[1] collection of LPPs. It demonstrates that even

[1] `https://www.netlib.org/lp/`

for perturbations of the order of 0.1% in the LPP data, the original solution becoming infeasible for the perturbed LPP.

There are broadly two approaches in the literature to deal with uncertainty in network flow problems (and other optimization problems more generally). The first one is stochastic optimization and the second is robust optimization. *Stochastic optimisation* uses a probability distribution for the uncertain data and then attempts to optimize the *expected* value of the objective function; many papers have proven computational intractability results for this kind of an approach [5]. In this project, we were instead interested in *robust optimization*, which essentially attempts to find a solution that remains feasible and aspires to minimize losses for even worst-case realizations of the uncertain data [5].

As far as we know, the work of Bertimas et al. [6] is the only one in the literature that takes a general robustness perspective for the specific problem of maximum flow in networks and includes a number of interesting theoretical, algorithmic, and computability results about it. They study two main approaches to increasing the tolerance of failure in the network structure. In particular, they focus on uncertainty in the incoming edges into each node since this can be used to address the case when a node might fail too. For instance, the uncertainty in a node v can be converted into uncertainty of in-edges by splitting node v into two nodes v' and v'' and then finding a flow which is tolerant to the failure of the edges into v' and the edge (v', v''). To show how such robustness problems are formulated, we give an account of how [6] incorporates robustness into network flows in the most basic way; the resulting canonical problem is known as the *robust maximum flow problem*.

To moderate the degree of conservatism, a parameter Γ_v for each vertex v is introduced and we suppose that at most Γ_v in-edges can fail at each node v. For each edge e, we let $\mu_e \in \{0, 1\}$ denote whether edge e fails. Then a s-t flow x in a network $G = (V, E)$ is said to be *robust* if weak conservation

$$\sum_{\epsilon \in N^-(v)} (1 - \mu_e) x_e - \sum_{e \in N^+(v)} x_e \geq 0 \tag{1}$$

holds for each node v different from s. Here we have to consider all possible binary sequences $(\mu_e)_{e \in E}$ that respect the conservatism condition set by $(\Gamma_v)_{v \in V}$. Then, the *robust maximum flow problem* is to maximize the flow into $t \sum_{e \in N^-(t)} x_e$ subject to the constraints (1). Note that there are

$$\sum_{v \in V \setminus \{s\}} \sum_{k=0}^{\Gamma_v} \binom{|N^{-1}(v)|}{k}$$

possible choices for the vector (μ_e) and, thus as stated, the problem of maximizing the robust flow involves an exponential number of constraints. However, with a little work we can formulate an equivalent polynomially tractable Linear Programming Problem (LPP). This is the content of [6, Theorem 1] and we fill in a few details in their proof to illustrate more fully the powerful result of strong duality, which also plays an important role in the

formulation of other LPPs used in our project. An excellent introduction to strong duality can be found in [11].

For a vertex $v \in V \setminus \{s\}$, letting $\mathcal{F}(v) := \{(\mu_e) \in [0,1]^{N^-(v)} : \sum \mu_e \leq \Gamma_v\}$ denote the permissible sequences (μ_e), the constraint (1) holds if and only if

$$\min_{(\mu_e)\in\mathcal{F}(v)} \left(\sum_{e\in N(v)} (1-\mu_e)x_e \right) - \sum_{e\in N^+(v)} x_e \geq 0,$$

which, in turn, holds if and only if

$$\sum_{e\in N^-(v)} x_e - \sum_{e\in N^+(v)} x_e - \max_{(\mu_e)\in\mathcal{F}(v)} \left(\sum_{e\in N^-(v)} \mu_e x_e \right) \geq 0.$$

This approach to the problem is already emphasizing that the robust flow approach is concerned with ensuring the feasibility of the flow even in the *worst-case* scenario. To understand the last optimization term in the above condition, we study the following subproblem to establish a more tractable relationship between the flow x_e and uncertainty variables μ_e:

$$\begin{aligned} z_v(x) = \max & \sum_{e\in N^-(v)} x_e\mu_e \\ \text{s.t.} & \sum_{e\in N^-(v)} \mu_e \leq \Gamma_v, \\ \text{and} \quad & 0 \leq \mu_e \leq 1, \quad \text{for all } e \in N^-(v). \end{aligned} \tag{2}$$

The second constraint along with the $|N^-(v)|$ constraints $\mu_e \leq 1$ can be reformulated as

$$\underbrace{\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}}_{A\in\mathbb{R}^{(|N^-(v)|+1)\times|N^-(v)|}} \underbrace{\begin{bmatrix} \mu_{e_1} \\ \mu_{e_2} \\ \mu_{e_3} \\ \vdots \\ \mu_{e_{N^-(v)}} \end{bmatrix}}_{\mu_e\in\mathbb{R}^{|N^-(v)|}} \leq \underbrace{\begin{bmatrix} \Gamma_v \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}}_{\text{“cost” vector for dual}} . \tag{3}$$

To formulate the dual of (2), we need to introduce $|N^-(v)| + 1$ variables, one for each

constraint in (3). Letting the variables be δ_v and θ_e, $e \in N^-(v)$, the dual is

$$
\begin{aligned}
w_v(x) = \min \quad & \gamma_v \delta_v + \sum_{e \in N^-(v)} \theta_e \\
\text{where} \quad & \underbrace{\begin{bmatrix} 1 & 1 & 0 & \dots & 0 \\ 1 & 0 & 1 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \dots & 1 \end{bmatrix}}_{A^T} \begin{bmatrix} \delta_v \\ \theta_{e_1} \\ \theta_{e_2} \\ \vdots \\ \theta_{e_{N^-(v)}} \end{bmatrix} \geq \begin{bmatrix} x_{e_1} \\ x_{e_2} \\ x_{e_3} \\ \vdots \\ x_{N^{-1}(v)} \end{bmatrix}, \\
\text{and} \quad & 0 \leq \delta_v, 0 \leq \theta_e \quad \text{for all } e \in N^-(v).
\end{aligned} \tag{4}
$$

By strong duality [11], we must have $z_v(x) = w_v(x)$. Therefore, the robust max flow problem can be formulated as

$$
\begin{aligned}
f_{\text{robust}} = \max \quad & \sum_{e \in N^-(t)} x_e \\
\text{s.t.} \quad & \sum_{e \in N^-(v)} (x_e - \theta_e) - \sum_{e \in N^+(v)} x_e - \Gamma_v \delta_v \geq 0, \quad \text{for } v \in V \setminus \{s\}, \\
& 0 \leq x_e \leq c_e, \quad \text{for } e \in E, \\
\text{and} \quad & 0 \leq \delta_v, 0 \leq \theta_e, \quad \text{for } e \in E.
\end{aligned} \tag{5}
$$

Approaching robustness in this manner is rather conservative—and potentially loses much optimality compared to classical solutions—since it attempts to optimize the flow in the worst case realization of the data uncertainty without allowing for rerouting the flow after failure takes place. There is one special case, however, when the optimal value of the robust max flow problem matches that of the classical solution. Aneja et al. [2] study the case of this problem when at most one edge is allowed to fail in the entire network. In other words, $\Gamma_v = 1$ for all $v \in V$ and $\sum_{e \in E} \mu_e \leq 1$. It can be shown without much difficulty that there exists a max flow protected against this level of failure which matches the optimality of the classical solution [2, Lemma 2]. Unlike [6], which uses an LPP approach, [2] gives a graph-based algorithm based on Newton's method of optimization with polynomial time complexity to solve the problem.

The adaptive formulation of [6] takes a more practical approach than the robust solution; it uses the general framework of *adjustable* robust optimization presented in [4]. Here the conservatism is controlled by a global parameter Γ, which specifies the maximum number of edge failures. Further, the adaptive approach assumes that the network has been constructed with each edge e having maximum capacity x_e, so that the network is functioning at full capacity; this highlights the practicality of this version of the problem, since in practice it is uneconomical to augment network capacities once the network has

been built. The *maximum adaptive flow* x is then defined as the s-t flow that can be *rerouted* to attain a maximum flow in the worst-case scenario of edge failures that is maximal among all initial s-t flows. Formally, this is formulated as the problem

$$
\begin{aligned}
f_{\text{adaptive}} = \max_{x} \min_{\mu} \max_{y} & \sum_{e \in N^-(t)} y_e \\
\text{s.t.} \quad & \sum_{e \in N^-(v)} y_e - \sum_{e \in N^+(v)} y_e = 0, \quad \text{for } v \in V \setminus \{s,t\}, \\
& 0 \le y_e \le (1-\mu_e) x_e, \quad \text{for } e \in E, \\
& \sum_{e \in E} \mu_e \le \Gamma, \\
& 0 \le x_e \le c_e, \quad \text{for } e \in E, \\
\text{and} \quad & \sum_{e \in N^-(v)} x_e - \sum_{e \in N^+(v)} x_e = 0, \quad \text{for } v \in V \setminus \{s,t\}.
\end{aligned} \tag{6}
$$

The authors use connections with the interdiction problem to simplify this formulation. The *k-edge interdiction problem* for max flows involves determining the k most vital edges in a network from the perspective of the max flow problem. In other words, which k edges should be removed from a network so that the maximum flow in the modified network is minimized? Wollmer [15] proposes an algorithm to solve this classical problem using the concept of a topological dual of a capacitated network. Bertimas et al. [6] include an approximation algorithm to compute the adaptive flow since they show for even moderate values of Γ, the formulated LPP is NP-hard.

3 Methods

The central goal of this project was to make a computational comparison of different robust solutions and the classical solution of the maximum flow problem. To this end, we implemented the relevant algorithms or heuristics discussed in the papers described in Section 2. We begin with a description of the main algorithms we implemented, along with some high-level implementation details and challenges.

3.1 Algorithms

All of our implementation is in the Julia Language. To program LPPs, we used the `JuMP` package [9] with the open-source `GNU Linear Programming Kit Library (GLPK)`[2] as the backend solver.

[2] `https://github.com/jump-dev/GLPK.jl`

- (Classical max flow: `shortestpath_augment_maxflow`) We implemented the Shortest Augmenting Path algorithm [1, Ch. 7], which augments flows along shortest paths from s to t, determined through distance labels assigned using a backward BFS from t.
- (Robust Max Flow: `robust_maxflow`) We implemented the robust max flow LPP (5).
- (Robust max flow with global Γ: `robust_maxflow_globalapprox`) The LPP in (5) requires a paramater Γ_v specified for each vertex v. However, how do we solve the robust max flow problem for a global parameter Γ bounding the number of edge failures in the entire network? This would be useful from both a practical point of view and for drawing comparisons with the adaptive flow (which is defined using a global Γ). Having a global Γ is equivalent to solving (5) over all vectors $(\Gamma_v)_{v\in V}$ such that $\sum_v \Gamma_v = \Gamma$. This constraint results in a quadratic optimization problem, which cannot be handled by `GLPK`. Thus, finding an exact max robust flow for a global Γ seems to be intractable and the paper [6] does not mention how they handle this problem. For a global Γ, we approximate the max robust flow value as the minimum value of the max robust flow values $f_{\text{robust}}((\Gamma_v)_{v\in V})$ for each vector $(\Gamma_v)_{v\in V}$ summing to Γ. In other words, letting $\mathcal{G}(\Gamma) = \{(\Gamma_v)_{v\in V} \in \mathbb{Z}_{\geq 0}^{|V|} : \sum_v \Gamma_v = \Gamma\}$, we use

  $$\min_{S\in\mathcal{G}} f_{\text{robust}}(S).$$

  Now $\mathcal{G}(\Gamma)$ is simply all the length-$|V|$ weak compositions of Γ, which we generate in `weak_compositions`. Note, however, that since it can be shown that $|\mathcal{G}| = \binom{\Gamma+|V|-1}{\Gamma}$, this approximation is computationally very expensive and can only be used for small values of Γ (for example, with $|V| = 100, \Gamma = 2$, we would have to solve the LPP (5) a total of 5050 times). Therefore, we allow passing an optional parameter to randomly sample from all the weak compositions.
- (Robust max flow with $\Gamma = 1$: `newton_maxflow`) We implemented the algorithm from [2] which handles the special case of robust max flow problem with global $\Gamma = 1$. This uses Newton's method of optimization and Theorem 1.
- (Adaptive max flow: `adaptive_maxflow`) We implemented the optimization problem (6). However, it is a quadratic optimization, not supported by `GLPK`.
- (Approximate adaptive max flow: `adaptive_maxflow_approx`) Since the adaptive max flow problem cannot be solved with available resources, we implemented the approximation algorithm proposed in [6, LPP 39]. While the authors prove good theoretical properties for this algorithm (e.g., [6, Sec. 4.1, Cor. 3]), we are skeptical that it yields close approximations to the max adaptive flow (more about this later).
- (Approximate adaptive max flow: `adaptive_maxflow_sampling`) Since we were unconvinced that the previous approximation algorithm has practical utility, we implemented an LPP to approximate adaptive max flows using random sampling of the

weak-compositions of all non-negative integers $k \leq \Gamma$, thus allowing us to avoid the quadratic nature of the optimization problem while getting at least a rough idea of the actual value.

3.2 Generating Test Flow Networks

Since we hoped to draw empirical observations about different robust and the classical max flows, we required a good source of flow networks (i.e., directed graphs with a capacity matrix). Generating flow networks for the purposes of simulation and testing requires care, in order to obtain results that hold for real-world flow networks too; see Klingman et al. [10], which discusses some important points in this regard. Indeed, the flow network generator NETGEN authored by Klingman et al. has become the unofficial standard for testing in the network flow community. Unfortunately, the original code for NETGEN in Fortran is written for a different era. Fortunately, we were able to find a modern translation in C[3]. NETGEN can generate flow networks of specified size and capacity constraints.

The powerful pipelining capabilities of Julia have allowed us to write `generate_netgenNetworks`, which calls the compiled NETGEN executable with a list of network parameters and saves the generated networks into a single file. Since many of our computational experiments require iterating through the generated networks, to avoid considerable memory costs, we have implemented the `netgenNetworks` struct, which provides an iterator over a filestream to the data file (akin to a generator in Python).

3.3 Computational Experiments

We now detail the main questions we explored and the computational experiments we conducted to make progress on them. Using the methodology described in Section 3.2, we generated a total of 210 flow networks with network size (number of nodes) ranging from 100 to 500 in steps of 20, and 10 different networks for each size; we used a minimum capacity of 1000 and maximum capacity of 10,000 for all networks. We conduct all tests on these networks. We have chosen these parameters keeping in mind the computationally expensive nature of solving max flow problems, especially robust ones, and the computational resources available to us. Moreover, we believe the network generation capabilities of NETGEN combined with the sufficient range in network size will give us meaningful results. Of course, if more computational power were available, all our results could be made more accurate by using a larger dataset. Moreover, since we cannot hope to draw any exact numerical results through testing on 210 flow networks, our central concern has been to draw out qualitative comparisons between different flows in all our experiments. Thus, we prefer to use small values of Γ along with a relatively small number of samples to allow computation for 210 different networks in a reasonable amount of time.

[3] `http://elib.zib.de/pub/mp-testdata/generators/netgen/netgen.c`

3.3.1 Experiment 1: Optimality ratios

The first, perhaps most basic, question that arises is: how much optimality do we expect to lose in the nominal value of the max flow when we utilize robust solutions rather than the classical one? Clearly, such a question has critical importance from an economic point of view. We test this in the experiment `run_exper1`: given a Γ, for each test network, we compute flow values for (1) a classical solution, (2) an adaptive solution approximated via `adaptive_maxflow_approx`, (3) an adaptive solution approximated via `adaptive_maxflow_sampling`, and (4) an robust flow calculated using `robust_maxflow_globalapprox`; the ratios of the last three flows to the classical max flow value are then calculated.

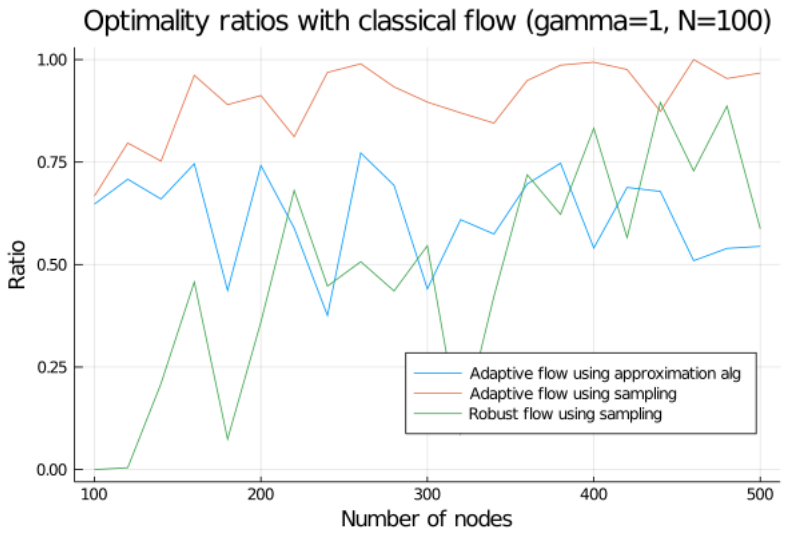

Figure 3.1

The result of one such experiment is shown in Figure 3.1, in which $\Gamma = 1$ and $N = 100$ samples were used to approximate the adaptive flow and robust flow. It is likely due to the low value of N that the resulting plot is not smooth, but a higher N would be more computationally expensive − the above computation already required an hour. Nevertheless, the plot still highlights important points about the three flows. First, note that the adaptive flow value derived using the approximation algorithm of Bertimas et al. [6] has similar optimality as the robust flow value, and at some points, is even dominated by it. This cannot truly be the case: the formulation of the problems (5) and (6) show that a solution to the robust max flow problem is also a solution to the adaptive version; this is clearly illustrated by the dominance of the optimality of the adaptive solution calculated via sampling. Thus, we are skeptical of the approximation algorithm and have used `adaptive_maxflow_sampling` as the sole method to approximate adaptive flows in later experiments. Next, note that the adaptive flow (derived via sampling) has an optimality quite close to 1 for all network sizes. In other words, for $\Gamma = 1$, an adaptive solution can

be found that is near-optimal. Similar results were observed for a few more small values of $\Gamma = 2, 3, 4$; the relation of the optimality ratio and Γ is explored in Experiment 2 3.3.2. Also, note the wide variation of the robust flow for different network sizes, while the other two flows are *relatively* stable. This is captured in standard deviations of the three flows: $0.109, 0.091, 0.357$, in the order they are plotted. From this we interpret that the relative optimality of a robust flow is highly dependent on the specific structure of the network while this might not be the case for adaptive flows.

3.3.2 Experiment 2: Vanishing robust flows

We can also approach the question of which solution is more conservative by asking at what level of pessimism (specified by Γ), do the different kinds of robust flows falls below a certain threshold in comparison to the classical flow? We carry out this comparison in the experiment `run_exper2`: for a specified parameter $\alpha \in [0, 1)$ and a network with classical max flow $f_{\text{classical}}$, we find the minimum positive integer Γ_{robust} such that the corresponding robust max flow $f_{\text{robust}}(\Gamma_{\text{robust}})$ satisfies

$$f_{\text{robust}}(\Gamma_{\text{robust}}) < \alpha \cdot f_{\text{classical}}.$$

Similarly, we find the minimum Γ_{adaptive} such that

$$f_{\text{adaptive}}(\Gamma_{\text{adaptive}}) < \alpha \cdot f_{\text{classical}}.$$

In Table 1, we have run this experiment with $\alpha = 0.1$ and using 100 samples to approximate flows. We chosen $\alpha = 0.1$ to take a very generous view towards what might be acceptable to the operator/manager of a network flow − operating a network flow while losing 90% optimality can only be conceived as overwhelming loss in any real-world network. The advantage of the flexible nature of an adaptive value is clearly visible. For most network sizes, the robust max flow drops below 10% of the classical value at around $\mathcal{G} = 2$ or 3. On the other hand, Γ_{adaptive} grows with the network size, allowing the failure of a proportionate number of edges before the adaptive max flow drops below 10%.

3.3.3 Experiment 3: Flow violation under edge perturbation

The handling of uncertainty in max flow problems by [6] leaves open an interesting question: how do we handle partial edge failures? In other words, can we find max flows that are robust against the fluctuation of an edge capacity $c[i, j]$ to some new capacity in the open interval $(0, c[i, j])$. Bertsimas and Sim [7] study how to handle such uncertainty in the case of a 0-1 LPP such as the shortest-path, min cost, maximal matching, and minimum spanning tree problems. They show that uncertainty in the cost coefficients can be handled by solving the original problem (with no data uncertainty) $n + 1$ times, thus retaining polynomial-time solvability. We investigated if their approach can be adopted to

# of nodes	Γ_{robust}	Γ_{adaptive}
100	1	9
120	4	5
140	1	8
160	2	10
180	1	12
200	2	17
220	1	21
240	2	16
260	2	25
280	3	29
300	1	26
320	2	22
340	3	31
360	5	33
380	3	32
400	2	36

Table 1: Avg. Γ at which robust and adaptive max flows drop below 10% of classical max flow (using 100 samples)

address uncertainty in edge capacities. We realized, however, that the max flow problem is fundamentally very different from problems that can be formulated as 0-1 LPPs. At the core, the issue is that there is no simple relationship between the partial failure of edges and the resulting effect on the max flow value. For instance, suppose we have two edges e_1 and e_2 whose capacities decrease by d_1 and d_2, respectively. Then, for the configuration in 3.2a, the total decrease in max flow will be $d_1 + d_2$, while for the configuration in 3.2b, it will be $\max(d_1, d_2)$.

Figure 3.2

So, we instead try comparing how the classical, robust, and adaptive flows perform under partial edge failures using perturbations of edge capacities; a perturbative analysis of general LPPs from the `Netlib` collection[4] was previously done in [3]. Taking a parameter

[4] `https://www.netlib.org/lp/`

$\alpha \in (0, 1)$, we perturb the capacity c_e of each edge $e \in E$ as

$$\widetilde{c_e} = (1 - \epsilon_e) \cdot c_e$$

for ϵ_e chosen randomly with uniform probability from $(0, \alpha)$. Then, for a max flow x in the original network, we measure its *perturbed capacity violation* ξ as

$$\xi(x) = \sum_{\substack{e \in E \\ x_e > \widetilde{c_e}}} (x_e - \widetilde{c_e}).$$

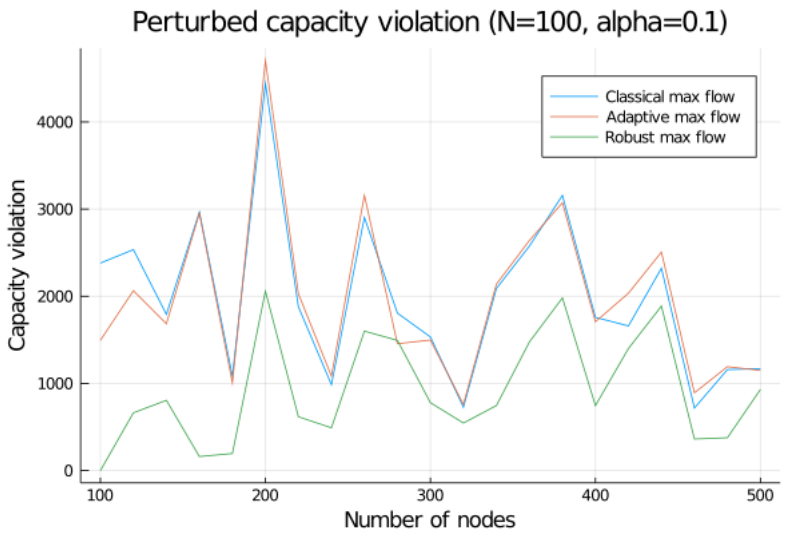

Figure 3.3: Perturbed capacity violation (using $N = 100$ samples, $\alpha = 0.1$, and $\Gamma = 1$)

Figure 3.3 shows the result of running the experiment `run_exper3` with a perturbation limit of 0.1 (so, edge capacities can decrease by as much as 10%), 100 samples, and $\Gamma = 1$. We see that the robust flow performs the best, with consistently the least capacity violation among all three flows. It is very striking that the adaptive flow so closely matches the classical flow in the violation of perturbed capacities. One reason for this result might be that the classical and adaptive flows $x_{\text{classical}}$ and x_{adaptive} are very similar to each other. This conjecture motivates the next experiment.

3.3.4 Experiment 4: Similarity of flows

We wish to measure the similarity between two $n \times n$ flow matrices x_A and x_B. We use the *Frobenius norm* Fr, defined as

$$\text{Fr}(x) = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}^2}.$$

Thus, we measure the *distance* of flows x_A and x_B as $\text{Fr}(x_A - x_B)$, and similarity as $\frac{1}{\text{Fr}(x_A - x_B)}$. We choose to use the Frobenius norm mostly because we require an aggregate measure of how much x_A and X_B differ. Moreover, this choice aligns with our definition of capacity violation ξ; the simplicity of Fr is a plus.

Figure 3.4 is obtained from running the experiment `run_exper4` with 100 samples and $\Gamma = 1$. It shows that the pair $(x_{\text{adaptive}}, x_{\text{calssical}})$ consistently has the lowest difference and thus highest similarity, as we had conjectured. Both the pairs $(x_{\text{classical}}, x_{\text{robust}})$ and $(x_{\text{robust}}, x_{\text{adaptive}})$ have matching similarities.

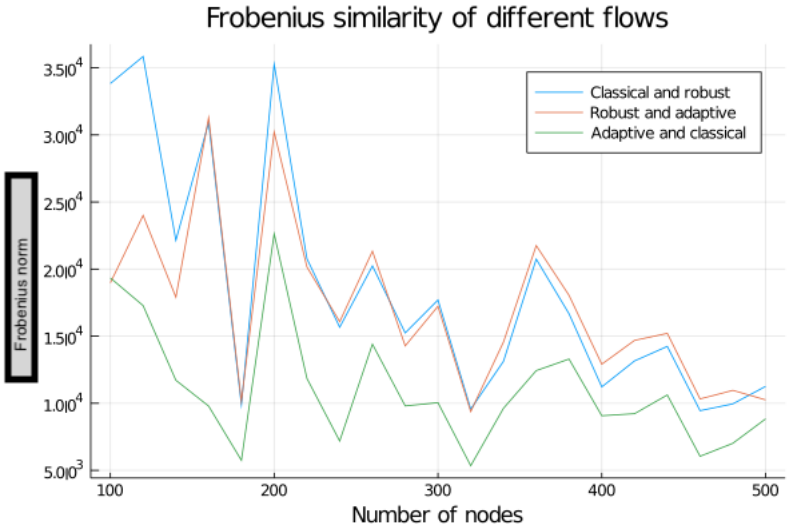

Figure 3.4: Avg. pairwise Frobenius similarity of classical, robust, and adaptive flows (using $N = 100$ samples and $\Gamma = 1$)

4 Discussion of Results

We have demonstrated many of the properties of robust and adaptive flows that one intuitively expects from their theoretical formulations. The rigid conservatism of robust flows is seen from Experiments 1 and 2. However, from Experiment 3, we see that the conservatism of robust flows might be beneficial when fluctuation in edge capacities or partial failure of edges is expected. From Experiment 1, we see that for small Γ, the adaptive max flow nearly matches the optimality of the classical solution. Moreover, Experiment 2 indicates that an adaptive flow with a high value (relative to classical solution) might exist for a wide range of moderate values of Γ.

What is, perhaps, most surprising is that in Experiment 3 the performance of an adaptive flow is about the same as that of a classical flow when we perturb edge capacities. In Experiment 4, we found an explanation for this phenomenon in the high similarity of

classical and adaptive flows; this also explains the observed near-optimality of adaptive solutions. Thus, an adaptive flow may be seen as a perturbation of the classical flow.

In each experiment, computational tractability has been a major hurdle for us. Thus, the small values of samples and Γ used in our experiments pose an obvious barrier in making any generalizations from our results. However, we are fairly confident that our qualitative observations will hold for bigger parameter values too.

5 Future Work

We briefly discuss some possible future directions for this project.

First, it would be interesting to further explain the reason for the high similarity of adaptive and classical flows. Next, since computing exact adaptive flows turned out to be intractable, we used approximations to it. Another way to gain an idea, and indeed an upper bound, of the adaptive flow value would be to solve the Γ-edge interdiction problem [15] and compute a classical maximum flow in the resulting graph. It would be interesting to see how the approximations compare with the max flow in the interdicted network.

One aspect of the maximum flow problem we have effectively ignored in our work is the concept of *flow decompositions*. For a flow x in G, it is a fundamental result of network flow theory (see e.g., [1, Ch. 6]) that x can be decomposed as $(x_P)_{P\in\mathcal{P}}$, with $\mathcal{P}$ being the set of all s-t paths, such that

$$x_e = \sum_{P\in\mathcal{P}:e\in P} x_P.$$

Bertimas et al. [6] formulate versions of robust and adaptive flows for such a flow decomposition. In particular, instead of failure of edges, they consider failure of s-t paths. Much of our experimental work could be attempted for these path-based robust flows. Further, we have some hope that considering the flow decomposition of a max flow can shed light on the problem of handling partial edges failures. The reason is that, in this formulation, the effect of failures of multiple edges along the same path P can be measured as a single decrease in x_P.

Finally, it would be useful to generalize the robust and adaptive flows to a network with multiple sources and sinks.

References

[1] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. New Jersey: Prentice Hall, 1993.

[2] Y. P. Aneja, R. Chandrasekaran, and K. P. K. Nair. "Maximizing Residual Flow under an Arc Destruction". In: *Networks* 38.4 (2001), pp. 194–198. ISSN: 1097-0037. DOI: 10.1002/net.10001.

[3] Aharon Ben-Tal and Arkadi Nemirovski. "Robust Solutions of Linear Programming Problems Contaminated with Uncertain Data". In: *Mathematical Programming* 88.3 (Sept. 2000), pp. 411–424. ISSN: 0025-5610. DOI: 10.1007/PL00011380.

[4] Aharon Ben-Tal et al. "Adjustable Robust Solutions of Uncertain Linear Programs". In: *Mathematical Programming; Heidelberg* 99.2 (Mar. 2004), pp. 351–376. ISSN: 00255610. DOI: 10.1007/s10107-003-0454-y.

[5] Dimitris Bertsimas and Vineet Goyal. "On the Power of Robust Solutions in Two-Stage Stochastic and Adaptive Optimization Problems". In: *Mathematics of Operations Research* 35.2 (May 2010), pp. 284–305. ISSN: 0364-765X, 1526-5471. DOI: 10.1287/moor.1090.0440.

[6] Dimitris Bertsimas, Ebrahim Nasrabadi, and Sebastian Stiller. "Robust and Adaptive Network Flows". In: *Operations Research* 61.5 (2013), pp. 1218–1242. ISSN: 0030-364X. URL: https://www.jstor.org/stable/24540495 (visited on 09/22/2020).

[7] Dimitris Bertsimas and Melvyn Sim. "Robust Discrete Optimization and Network Flows". In: *Mathematical Programming; Heidelberg* 98.1-3 (Sept. 2003), pp. 49–71. ISSN: 00255610. DOI: 10.1007/s10107-003-0396-4.

[8] Dimitris Bertsimas and Melvyn Sim. "The Price of Robustness". In: *Operations Research; Linthicum* 52.1 (Jan/Feb 2004), pp. 35–53. ISSN: 0030364X. URL: http://search.proquest.com/docview/219192770/abstract/B066B179AA974F38PQ/1 (visited on 09/22/2020).

[9] Iain Dunning, Joey Huchette, and Miles Lubin. "JuMP: A Modeling Language for Mathematical Optimization". In: *SIAM Review* 59.2 (Jan. 2017), pp. 295–320. ISSN: 0036-1445, 1095-7200. DOI: 10.1137/15M1020575.

[10] D. Klingman, A. Napier, and J. Stutz. "NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems". In: *Management Science* 20.5 (1974), pp. 814–821. ISSN: 0025-1909. URL: https://www.jstor.org/stable/2630092 (visited on 10/16/2020).

[11] James B. Orlin. "Duality in Linear Programming". In: *Lecture Notes for MIT Course 15.053.* URL: http://web.mit.edu/15.053/www/AMP-Chapter-04.pdf.

[12] H. Donald Ratliff, G. Thomas Sicilia, and S. H. Lubore. "Finding the n Most Vital Links in Flow Networks". In: *Management Science (pre-1986); Linthicum* 21.5 (Jan. 1975), pp. 531–539. ISSN: 00251909. URL: http://search.proquest.com/docview/205811166/abstract/3C5348121FA844CAPQ/1 (visited on 09/26/2020).

[13] Alexander Schrijver. "On the History of the Transportation and Maximum Flow Problems". In: *Mathematical Programming* 91.3 (Feb. 2002), pp. 437–445. ISSN: 0025-5610, 1436-4646. DOI: 10.1007/s101070100259.

[14] A. L. Soyster. "Convex Programming with Set-Inclusive Constraints and Applications to Inexact Linear Programming". In: *Operations Research* 21.5 (Oct. 1973), pp. 1154–1157. ISSN: 0030-364X, 1526-5463. DOI: 10.1287/opre.21.5.1154.

[15] Richard Wollmer. "Removing Arcs from a Network". In: *Operations Research* 12.6 (1964), pp. 934–940. ISSN: 0030-364X. URL: https://www.jstor.org/stable/168177 (visited on 09/24/2020).