Noticeable Network Delay Minimization via Node Upgrades

Sourav Medya
University of California, Santa Barbara
medya@cs.ucsb.edu

Jithin Vachery
Indian Institute of Technology, Madras
jithin@cse.iitm.ac.in

Sayan Ranu
Indian Institute of Technology, Delhi
sayanranu@iitd.ac.in

Ambuj Singh
University of California, Santa Barbara
ambuj@cs.ucsb.edu

ABSTRACT

In several domains, the flow of data is governed by an underlying network. Reduction of delays in end-to-end data flow is an important network optimization task. Reduced delays enable shorter travel times for vehicles in road networks, faster information flow in social networks, and increased rate of packets in communication networks. While techniques for network delay minimization have been proposed, they fail to provide any *noticeable* reduction in individual data flows. Furthermore, they treat all nodes as equally important, which is often not the case in real-world networks. In this paper, we incorporate these practical aspects and propose a network design problem where the goal is to perform k network upgrades such that it maximizes the number of flows in the network with a *noticeable* reduction in delay. We show that the problem is NP-hard, APX-hard, and non-submodular. We overcome these computational challenges by designing an *importance sampling* based algorithm with provable quality guarantees. Through extensive experiments on real and synthetic data sets, we establish that importance sampling imparts up to 1000 times speed-up over the greedy approach, and provides up to 70 times the improvement achieved by the state-of-the-art technique.

PVLDB Reference Format:
. . *PVLDB*, (): xxxx-yyyy, .
DOI:

1. INTRODUCTION

Many applications generate data that flow through a network. Examples include trajectories of vehicles in road networks [2], flow of packets in communication networks [6], and electricity distribution in power grids [18]. Naturally, the quality of flow is governed by the network properties. In this paper, we study the problem of network optimization to minimize delay in data flow.

To provide a concrete example, consider a road network and trajectories of vehicles that ply through this network. In a road network, each edge corresponds to a road segment and a node represents an intersection. Naturally, optimizing the network for smooth flow of vehicles is of critical importance. The quality of a traffic system is enhanced if the commuting time is as low as possible. Therefore, an important question in optimizing a road network is as follows. *Given the trajectories and the number of people navigating through these, how do we optimize the network to reduce the commuting time (delay) for the maximum number of people?* Optimizing the network in this example could mean widening a road (edge), installing better signalling procedure at an intersection (node) or deployment of traffic police at critical locations (edge or node) to better regulate vehicles.

An important consideration in this optimization problem is the budget. More specifically, we never have infinite resources to widen as many roads as needed. Rather, we can introduce only k changes where k is decided based on the resource constraints. Consequently, the goal is to intelligently perform k network optimizations such that the overall path delays are minimized. This problem of changing k network attributes (e.g. node or edge delays) to achieve or optimize a certain property (e.g. path delays) comes under *budget-constrained network design* problems.

Applications: A prominent application of the proposed problem is in transportation infrastructure management for traffic congestion minimization. Metropolitan cities worldwide are facing severe traffic congestion, which increases pollution, fuel usage, and unproductive travel time [12]. For example, it is estimated that drivers in the City of Chicago cumulatively suffered 302 million hours of travel delay with a total congestion cost of $\$7,222$ million in 2014[26]. The cause of traffic delays could arise from various factors such as overshooting road capacities, poor road conditions, and sub-optimal signalling infrastructure. It is therefore of practical importance to analyze road network data, identify bottleneck points, and propose systematic upgrades to these bottlenecks so that they work in cohesion to reduce congestion.

There are several applications beyond traffic networks as well. For instance, given a communication network, end-to-end delays of data packet flow can be improved via upgrading network devices [6]. The delays of sending packets (data) arise in the device level. Improving or upgrading the devices, such as upgrading a switch, enable a better and faster communication system. As another application scenario, consider an airport network [15] generated by a particular carrier. In the network, airports represent nodes and the edges correspond to flights offered by the carrier between endpoint cities. Based on information of the past flights one can associate airports with airline-caused delays such as security check delay, luggage handling, etc. An important question for a carrier is how to minimize overall travel time by improving the available infrastructure (e.g. luggage handling).

The importance of this problem has been recognized in the data mining literature [13, 17]. The problem is modeled as follows. The delay in each node or edge of the network is quantified using a weight. The length of a path is the summation of its constituent node and edge weights. A *network upgrade* involves reducing the weight of a node or an edge to a small value $\alpha \geq 0$. The total de-

Proceedings of the VLDB Endowment, Vol. , No.

DOI:

Improvement (%)	$k = 8$	$k = 10$
$= 0$	92.22	92.16
> 0	7.78	7.84
$>= 2$	6.73	6.78
$>= 5$	2.44	2.45
$>= 10$	0.43	0.42

Table 1

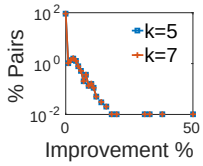


Figure 1

Table 2: **Table shows the improvement (%) of pairs (%) for budget 8 and 10. Figure shows how the percentage of improvement is skewed over the number of pairs.**

lay in the *flow* from node u to v is therefore the length of shortest path from u to v. Given a budget k, the goal is to perform k *network upgrades* such that the sum (or average) of all pairs shortest path distances in the network is minimized. Existing techniques, however, ignore two key practical aspects.

1. Noticeable impact: Existing techniques [15, 6] focus on minimizing the sum of the shortest path distances across all pairs of nodes in a network. This formulation leads to negligible reduction in the delay between most of the individual pairs. Consequently, none of the stakeholders (such as vehicles in road networks) witness any noticeable difference in their experiences and may not be satisfied even in the improved network.

To substantiate our claim that existing techniques fail to provide noticeable reduction on individual pairwise delays, we execute the state-of-the-art algorithm [15] on the Los Angeles road network and present the results in Table 1 and Fig. 1. Each node represents a traffic intersection and node weights represent the delay faced by commuters in the corresponding intersection. Further details on the dataset are available at [15]. Fig. 1 presents the distribution of improvement (%) over all node pairs and Table 1 presents the cumulative distribution of improvement (%). As shown in Table 1, more than 92% of the total node pairs do not improve at all. Furthermore, less than 3% of the total pairs have improvement more than 5%.

To mitigate the above outlined issue: in this paper, we ask a more practical question:*How should we perform k node upgrades such that the maximum number of shortest paths have a* significant reduction *in their lengths?* We call a reduction in length *significant* if the reduced length is at least β% shorter than its original length, where β is a user-provided input parameter.

2. Node pair importance: Existing techniques assume that the shortest path between any pair of nodes is equally important. In reality, this is often not the case. In road networks for example, arterial roads connecting prominent places such as the downtown, office and residential districts, airport, etc., have far higher traffic than other roads. Consequently, having a noticeable reduction in the delay in these arterial roads is of higher importance than other less frequently travelled roads.

In this study, we incorporate these practical aspects. Our main contributions are as follows:

- We propose and formalize a novel and practical *Path Optimization Problem (POP)* to minimize delays in a network. The proposed problem incorporates practical aspects such as *importance* of node pairs and *noticeable* improvement in quality.
- We show that POP is NP-hard as well as APX-hard. We formulate an optimal mixed integer programming (MIP) formulation.
- To overcome the hardness of POP, we propose an *importance sampling* algorithm with probabilistic approximation guarantees.
- We perform extensive benchmarking on real-world road networks and establish that importance sampling obtains a speed-up of three orders of magnitude over greedy and 70 times better than the state-of-the-art algorithm. In addition, our experiments on synthetic datasets establish that ISS is robust to variation in network structure, delay distributions and node-pair importances.

2. PROBLEM DEFINITION

In this section, we first define the concepts central to our problem and then analyze the problem complexity.

2.1 Preliminaries

The path optimization problem ingests two sources of data as inputs: a network and a collection of *network flows*.

DEFINITION 1 (NETWORK). *A network is modeled as a graph (directed or undirected) $G(V, E, L)$, where V and E are sets of nodes and edges respectively and L is function $L : V \rightarrow \mathbb{R}_{>0}$ over V. This function specify the delays (weights) $l_v := L(v)$ of individual nodes.*

In several domains, objects flow through a network. For example, in transportation networks, vehicles move through a road network. Similarly, in a communication network, data packets move through a sequence of connected routers. We capture the flow of such objects through the idea of network flows.

DEFINITION 2 (NETWORK FLOW). *A network flow represents the flow of an object through a network. Mathematically, each flow corresponds to some path $P_{v_1,v_r} = (v_1, v_2, ..., v_r)$ from the source node v_1 to destination v_r.*

In a transportation network, the set of network flows would be a set of vehicular trajectories, and in a communication network, the network flows would constitute the routes taken by data packets

DEFINITION 3 (PATH DELAY). *The* delay *of a path is defined as the cumulative delays (weights) of the nodes along the path, excluding that of the destination node. More formally, if $P_{v_1,v_r} = (v_1, v_2, ..., v_r)$ is a path from node v_1 to v_r, its delay is defined as $\Sigma_{i=1}^{r-1} l_{v_i}$.*

Weight at the destination node in a path is excluded since the destination node typically does not add any delays in case of the targeted applications (e.g., commuting time in the traffic network). Nonetheless, if this assumption is not true for a particular domain, the delay in the last node can easily be incorporated in the path delay without causing any difference to the properties of the proposed algorithms. The *shortest path* between two nodes s and t is the one with the minimum delay among all paths connecting these two nodes and its delay is denoted as $d(s, t)$. By convention, $d(s, s) = 0$ for all $s \in V$. Furthermore, if there does not exist a path between two nodes u and v, then $d(u, v) = \infty$. We next define the concept of *improving* a node.

DEFINITION 4 (NODE IMPROVEMENT). *A node is improved by reducing the node delay l_v to a small value $\alpha \geq 0$. The nodes (a fixed/budget k number of nodes) to be improved are chosen from a given candidate set of nodes Γ.*

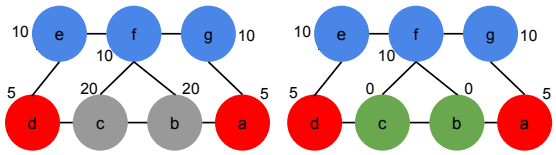
For simplicity, in the rest of the paper, we assume $\alpha = 0$. The proposed techniques and proofs hold for any value of α.

The reduction of delays in few nodes (call it *solution set* S) may *significantly* reduce the delay of the shortest paths. We quantify significant improvement in path delay through the definition of β-*improvement*.

DEFINITION 5 (β-IMPROVEMENT). *A node pair (s, t) is β-improved if $\frac{d(s,t)-d(s,t;S)}{d(s,t)} \geq \beta$, where $d(s, t)$ is the original shortest path and $d(s, t; S)$ is the updated shortest path after improving nodes in solution set S.*

We use Λ_S to denote the set of β-improved node pairs.

Not all node pairs are equally important. For instance, if the traffic flow between two prominent regions is much higher than the flow between two less-visited regions, then it is more important to improve the path between the prominent regions. This aspect is modeled through *node pair flow*.



(a) Initial graph (b) Modified graph

Figure 2: **Example of Path Optimization Problem. We want to optimize the set of pairs $\{(a,d),(d,f)\}$ with a budget of two nodes from the candidates $\Gamma = \{b,c,e,g\}$.**

DEFINITION 6 (NODE PAIR FLOW:). *Let $\mathbb{F}$ be the collection of all network flows in the network. The flow associated with a node pair (u,v), denoted by $\xi_{u,v}$, is the proportion of flows originating at u and terminating at v. Mathematically,*

$$\xi_{u,v} = \frac{\big|\{f \in \mathbb{F} \mid f \text{ starts at } u \text{ and ends at } v\}\big|}{|\mathbb{F}|} \tag{1}$$

The quality of a solution set S is quantified as the total flow $f(S)$ among β-improved node pairs. Mathematically,

$$f(S) = \sum_{(u,v)\in\Lambda_S} \xi_{u,v} \tag{2}$$

The path optimization problem is defined as follows.

PROBLEM 1. *(Path Optimization Problem (POP)) Given a network $G = (V,E,L)$, the set of flows $\mathbb{F}$, the improvement parameter β, a candidate set of nodes Γ that can be improved, and a budget k, find a solution set $S \subset \Gamma$ such that $|S| = k$ and $f(S)$ is maximized.*

EXAMPLE 1. *Fig. 2 shows a possible solution of size $k = 2$. Initially, the delays of the green, red, blue and grey nodes are $0, 5, 10, 20$ respectively. Let the candidate set $\Gamma = \{b,c,e,g\}$ and $\beta = 60\%$. Furthermore, let the collection of network flows $\mathbb{F}$ be such that $|\mathbb{F}| = 150$ and $\xi_{a,d} = \frac{100}{150}, \xi_{d,f} = \frac{50}{150}$, and the flow between all other pairs is 0. Fig. 2b shows an optimal solution, where the modified graph has the delays of b and c are reduced to 0. Improving b and c results in $f(\{b,c\}) = \frac{150}{150} = 1$ since it produces at least 60% improved shortest path between (a,d) (35 becomes 5) and between (d,f) (15 becomes 5). normalized*

Edge Upgrades: In our formulation, we only allow node upgrades. Therefore, a valid question arises: *What happens if delays are on edges and one needs to upgrade edges to improve the network?* We limit ourselves to node upgrades since this is a more generic formulation and any network with delays on edges can easily be mapped to an equivalent network with delays on nodes. More specifically, an edge (u,v) with delay l_e is partitioned into two edges (u,e) and (e,v), where e is a new node inserted into the network and the delay on node e is l_e. By setting Γ to include only the newly added nodes corresponding to the edges, we solve the problem for edge upgrades. If delays are both on edges and nodes, Γ may include all candidate nodes and edge-converted-nodes. This mapping can be performed in polynomial time holds for both undirected and directed networks.

Practical Implications: In a practical scenario, the cause of a delay can be multiple. For example, in a road network, it could arise due to reaching the capacity of a road segment, poor road quality such as potholes, or sub-optimal signalling infrastructure. Reducing delays in such cases could therefore mean widening a road (edge), improving the quality of a road (edge), or upgrading signalling infrastructure at an intersection (node). In our problem, however, we do not deal with the exact cause of the delay or the upgrade mechanism to remove this delay. These domain-specific aspects are abstracted out and the problem only identifies the delay causing entities, and the impact of removing these delays. Due to this layer of abstraction, the proposed algorithm could potentially be used in networks from multiple domains.

2.2 Hardness and Approximability

In this section, we analyze the hardness and approximability of the proposed problem. First, we prove that POP is NP-hard.

THEOREM 1. *POP (decision version) is NP-hard.*

PROOF. Consider an instance of the NP-complete Set Cover problem, defined by a collection of subsets $S = \{S_1, S_2, ..., S_m\}$ for a universal set of items $U = \{u_1, u_2, ..., u_n\}$. The problem is to decide whether there exist k subsets whose union is U. To define a corresponding POP instance, we construct an undirected graph with $m + n + 1$ nodes in V: there are nodes i and j corresponding to each set S_i and each element u_j respectively, and an undirected edge (i,j) whenever $u_j \in S_i$. Node a is added to the graph. Node a is connected to i for all $S_i \in S$. The reduction clearly takes polynomial time. The candidate set $\Gamma = \{i | S_i \in S\}$, $\xi_{a,j} = 1/n$ for each node j corresponding to $u_j \in U$, and all remaining node pairs have 0 flow between them. All weights are equal (let us say d) and $\beta = \frac{d}{2d} = \frac{1}{2}$.

A set $S' \subset \Gamma = S$, with $|S'| \leq k$ is a set cover iff $f(S')$ becomes 1. Assume that S' is a set cover and weights are reduced to $\alpha = 0$ for every node in S'. Then $f(S')$ becomes 1 as the nodes in U are of distance d from a. Note, in the initial graph, the distances between a and nodes in U are $2d$.

On the other hand, assume that the $f(S')$ becomes 1 after reducing the weights of nodes in any set $S' \subset S$ with $|S'| \leq k$. The only way to have the distance between (a,u) $(u \in U)$, improved by $\beta = \frac{d}{2d}$ or $\frac{1}{2}$ is by making S' a set cover. □

THEOREM 2. *POP is APX-hard. More specifically, it is NP-hard to approximate POP within a factor of $(1 - \frac{1}{e})$.*

PROOF. For reduction, we use the Maximum Coverage (MSC) problem. Given a collection of subsets $S_1, S_2, ..., S_m$ for a universal set of items $U = \{u_1, u_2, ..., u_n\}$, the problem is to choose at most k sets to cover as many elements as possible. We give an L-reduction [29] from the MSC problem with parameters x and y. Our reduction is such that following two equations are satisfied:

$$OPT(I_{POP}) \leq xOPT(I_{MSC}) \tag{3}$$

$$OPT(I_{MSC}) - s(T^M) \leq y(OPT(I_{POP}) - s(T^P)) \tag{4}$$

where I_{MSC} and I_{POP} are problem instances, and $OPT(Y)$ is the optimal value for instance Y. $s(T^M)$ and $s(T^P)$ denote any solution of the MSC and POP instances respectively. If the conditions hold and POP has an ϵ approximation, then MSC has an $(1 - xy(1-\epsilon))$ approximation. However, MSC is NP-hard to approximate within a factor greater than $(1 - \frac{1}{e})$. It follows that $(1 - xy(1-\epsilon)) < (1 - \frac{1}{e})$, or, $\epsilon < (1 - \frac{1}{xye})$ [3]. So, if the conditions are satisfied, POP is NP-hard to approximate within a factor greater than $(1 - \frac{1}{xye})$.

We apply the same construction as in Theorem 1. However, we use MSC problem for reduction. Let the solution of I_{POP} be $s(T^P)$. That implies that the number of β-improved pairs increases by $s(T^P) \cdot n$. Note that, by construction, $s(T^P) \cdot n = s(T^M)$. So, it follows that both the conditions are satisfied when $x = 1/n$ and $y = n$. Thus, POP is NP-hard to approximate within a factor greater than $(1 - \frac{1}{e})$. □

We next investigate the existence of submodularity property. A function $f(.)$ is submodular if the marginal gain from adding an element to a set S is at least as high as the marginal gain from adding it to a superset of S. Mathematically, it satisfies:

$$f(S \cup \{o\}) - f(S) \geq f(T \cup \{o\}) - f(T) \tag{5}$$

for all elements o and all pairs of sets $S \subseteq T$. For submodular and monotone functions, the greedy algorithm of iteratively adding the

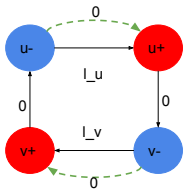


Figure 3: **The figures shows the transformed graph G' as a representation of nodes u and v and the edge (u, v) in G. The values show the delays of the edges.**

element with the maximum marginal gain approximates the optimal solution within a factor of $(1 - \frac{1}{e})$[19]. The next theorem shows that the optimization function related to POP is monotone but does not have submodular property.

THEOREM 3. *The objective function $f(.)$ is monotone but not submodular.*

PROOF. Monotone: Follows from the definition of a shortest path. Let the reduction of delays in S result in the set of β-improved node pairs, Λ_S. Reducing the delay of one more node $v \in \Gamma$ cannot decrease $\sum_{(u,v)\in\Lambda_S} \xi_{u,v}$, i.e., $f(S)$. Thus, $f(.)$ is monotone.

Non-submodular: To prove non-submodularity, we consider the simple example of a chain graph G of four nodes with unit delays: node x_1 is connected to x_2, x_2 to x_3 and x_3 to x_4 by edges. The intuition is the following: a super-set of nodes as solution might force the shortest paths improved by β along with the newly added vertex, whereas, a sub-set of nodes is not sufficient to improve by β. Let us set $A = \phi, B = \{x_2\}, \beta = \frac{2}{3}, \alpha = 0, \Gamma = V$. Only the node pair $\{(x_1, x_4)\}$ has a flow of 1. In our example, $f(B \cup \{x_3\}) = 1$, $f(B) = 0$ as $\beta = \frac{2}{3}$. $f(A \cup \{x_3\}) = 0$, $f(A) = 0$. So, $f(B \cup \{x_3\}) - f(B) > f(A \cup \{x_3\}) - f(A)$. So, $f(\cdot)$ is not submodular. □

3. ALGORITHMS

POP is not only NP-hard, but also hard to approximate. Furthermore, due to lack of submodularity, it is hard to even decide on an optimization strategy. We therefore start with the optimal solution to POP through *mixed integer programming (MIP)*. Next, we improve efficiency through a greedy approach. Finally, we further expedite greedy through importance sampling with probabilistic guarantees on the approximation error, which allows us to scale on real networks containing more than half a million nodes.

3.1 Optimal Solution

We formulate the POP problem as a mixed integer program (MIP), in order to obtain the optimal solution. We use a multi-commodity flow formulation[6] to compute the shortest path delay between a node pair $p = (u, v) \in \mathscr{P}$ (for simplicity, $\mathscr{P} = V \times V$). To apply MIP on a given graph G, we first convert it to a directed graph G' as follows: a node v is replaced by two nodes v^- and v^+ with two additional parallel edges from v^- to v^+ with delays l_v (original node edge, e_v) and 0 (upgraded node edge e'_v) respectively. If an edge (u, v) is present in the original graph, there are two edges (u^+, v^-) and (v^+, u^-) with delays 0 in G'. Figure 3 shows an example of this transformation procedure for an edge $e = (u, v)$. The variables used in the MIP formulation are as follows:

- x_v: a flag for whether node v is to be upgraded.
- x_p: a variable that indicates a pair p is β-improved
- $d'(p)$:the effective shortest path delay between nodes in pair p
- Δ_p: difference between effective improvement and β in pair p.
- $budget$: the total number of upgraded nodes
- g_{pv}: a continuous variable that indicates the flow of the commodity p on edge e_v.
- g'_{pv}: continuous variable that indicates the flow of the commodity p on edge e'_v.
- h_{pe}: continuous variable that indicates whether edge e is chosen to be on the shortest path for the pair p.

In an integral solution, g_{pv} and g'_{pv} denote whether the original node and upgraded node respectively are chosen to be on the shortest path between the pair p in an integral solution. We use $\delta^-(v^-)$ and $\delta^+(v^+)$ to denote the set of incoming and outgoing edges respectively. $d(p)$ denotes the original shortest path distance (we assume this as constant as it is given) in the initial graph for pair p. M is a large positive constant. The full MIP formulation is as follows:

$$\max \left\{ \sum_{p\in\mathscr{P}} \xi_p \right\} \text{ such that,}$$

$$g_{ps} + g'_{ps} = 1, g_{pt} + g'_{pt} = 1 \qquad \forall p = (s,t) \in \mathscr{P} \tag{6}$$

$$\sum_{e\in\delta^-(s^-)} h_{pe} = 0 \qquad \forall p = (s,t) \in \mathscr{P} \tag{7}$$

$$\sum_{e\in\delta^+(s^+)} h_{pe} = g_{ps} + g'_{ps} \qquad \forall p = (s,t) \in \mathscr{P} \tag{8}$$

$$\sum_{e\in\delta^-(t^-)} h_{pe} = g_{pt} + g'_{pt} \qquad \forall p = (s,t) \in \mathscr{P} \tag{9}$$

$$\sum_{e\in\delta^+(t^+)} h_{pe} = 0 \qquad \forall p = (s,t) \in \mathscr{P} \tag{10}$$

$$\sum_{e\in\delta^+(v^+)} h_{pe} = g_{pv} + g'_{pv} \qquad \forall p = (s,t) \in \mathscr{P}, \forall v \neq s,t \in V \tag{11}$$

$$\sum_{e\in\delta^-(v^-)} h_{pe} = g_{pv} + g'_{pv} \qquad \forall p = (s,t) \in \mathscr{P}, \forall v \neq s,t \in V \tag{12}$$

$$g'_{pv} \leq x_v, g_{pv} \leq 1 - x_v \qquad \forall p = (s,t) \in \mathscr{P}, \forall v \neq s,t \in V \tag{13}$$

$$d'(p) = \sum_{v\in V} l_v \cdot g_{pv} \qquad \forall p \in \mathscr{P} \tag{14}$$

$$\Delta_p = \frac{d(p) - d'(p)}{d(p)} - \beta \qquad \forall p \in \mathscr{P} \tag{15}$$

$$budget = \sum_{v\in V} x_v, \qquad budget \leq k, x_v \in \{0,1\} \forall v \in V \tag{16}$$

$$\Delta_p \geq -M(1 - x_p), \qquad \Delta_p < Mx_p, \forall p \in \mathscr{P} \tag{17}$$

$$x_p \in \{0,1\} \qquad \forall p \in \mathscr{P} \tag{18}$$

$$h_{pe}, g_{pv}, g'_{pv} \geq 0 \qquad \forall p \in \mathscr{P}, e \in E, v \in V \tag{19}$$

The constraints in MIP formulation for POP are shown in Eqs. (6-19). The constraints as Eqs. (6-12) are used to model the shortest path delay of each terminal pair as multi-commodity flow. Constraints (6-10) enforce the nodes s and t to be the source and sink respectively with one unit of flow in each terminal pair (s, t). The next two constraints (11,12) ensure the flow conservation through the rest of the nodes. Constraint 13 enforces that the upgraded node edge e'_v will carry the flow instead of the original node edge e_v, when the node v is upgraded. The original node edge e_v carries the flow when the node v is not upgraded. Constraint 14 computes the total effective delay. Constraint 15 computes the difference between effective fractional improvement of each pair and β. Constraint 16 computes the total budget and sets the maximum as k. It also ensures that the upgrade decision variables for nodes are binary. Constraints 18 and 19 ensure that improvement decision variables for

pairs and flow variables are binary and non-negative respectively. Constraint 17 ensures that x_p is 1 when Δ_p is non-negative.

While the above MIP formulation allows us to compute the optimal solution, it is not scalable to large networks. This motivates us to design approximation algorithms for POP. For any approximation algorithm, it is desirable to provide theoretical guarantees on the approximation error. However, since POP is APX-hard, it is NP-hard to even approximate within a factor of $1 - \frac{1}{e}$. Owing to this property, we first consider a restricted version of POP and develop a greedy algorithm. Next, we generalize this greedy algorithm for POP and provide probabilistic error bounds.

3.2 Restricted Path Optimization Problem (RPOP)

RPOP introduces the restriction that *each node pair can be β-improved by only one node*. In other words, there cannot be two β-improved nodes in the shortest path between a node pair. We next show that RPOP is a lower bound of POP.

THEOREM 4. *Let $f^r(S)$ be the total flow from all pairs that satisfy the RPOP constraint. For any solution set S, $f^r(S) \leq f(S)$.*

PROOF. Let P^r and P be the set of all β-improved node pairs under $RPOP$ and POP for a solution set S respectively. Since $P^r \subseteq P$,

$$f^r(S) = \sum_{\forall (u,v) \in P^r} \xi_{u,v} \leq f(S) = \sum_{\forall (u,v) \in P} \xi_{u,v} \quad \square$$

Since RPOP is a lower bound of POP, intuitively, a good solution to RPOP is likely to yield a good solution to POP as well if the objective functions yield values that are close to each other. With this intuition, we next show that RPOP is monotone and submodular.

THEOREM 5. *The objective function $f^r(.)$ is monotone and submodular.*

PROOF. Monotone: The proof is similar as in Theorem 3.
Submodular: We consider improvement (delay reduction) of two sets of nodes, V_a and V_b where $V_a \subset V_b$, and show that $f^r(V_a \cup \{v\}) - f^r(V_a) \geq f^r(V_b \cup \{v\}) - f^r(V_b)$ for any node $v \in \Gamma$ such that $v \notin V_a$ and $v \notin V_b$. Let $F(A)$ be the set of node pairs (s,t) which are β-improved by a vertex $v \in A$. Then $f^r(.)$ is submodular if $F(V_b \cup \{v\}) \setminus F(V_b) \subseteq F(V_a \cup \{v\}) \setminus F(V_a)$. To prove this claim, we use the described constraint. Therefore, each pair $(s,t) \in F(V_b)$ is β-improved by only one node in V_b. As $V_a \subset V_b$, adding v to V_a will β-improve some of the pairs which are already β-improved by $V_b \setminus V_a$. Then, for any newly β-improved pair $(s,t) \in F(V_b \cup \{v\}) \setminus F(V_b)$, it must hold that $(s,t) \in F(V_a \cup \{v\}) \setminus F(V_a)$. $\square$

THEOREM 6. *RPOP is APX-hard, i.e., RPOP cannot be approximated within a factor greater than $(1 - \frac{1}{e})$.*

PROOF. The proof directly follows from Theorem 2 as the construction respects the described constraint. $\square$

Since the objective function $f^r(.)$ is submodular and monotone, the greedy algorithm of iteratively adding the node with the maximum marginal gain on Eq. 2 approximates RPOP within a factor of $(1 - \frac{1}{e})$[19]. This result, in conjunction with Theorem 6, allows us to conclude that greedy is the best possible polynomial-time algorithm for RPOP.

The above conclusion motivates us to propose a greedy heuristic (GSN, Algorithm 1) for optimizing POP as well. More specifically, we know from Theorem 4 that RPOP is a lower bound for POP. Furthermore, from an intuitive point of view, in most real life networks the number of node pairs is much larger than the budget k.

Algorithm 1: Greedy Selection of Nodes (GSN)

Require: Network $G = (V, E, L)$ with vertex delays $l(v)$, network flows $\mathbb{F}$, Budget k, candidate set Γ, β.
Ensure: A subset of k nodes
1: $S \leftarrow \emptyset$
2: Compute the shortest path between all nodes that is part of some node pair with non-zero flow to all other nodes in the network. Store the corresponding shortest path delays in distance matrices A, B
3: **while** $|S| < k$ **do**
4: **for** $v \in \Gamma \setminus S$ **do**
5: $\lambda_v \leftarrow \sum_{(x,y) \in \Lambda_s} \xi_{x,y}$ where Λ_s is the set of new β-improved pairs when $l_v = 0$ (Use A to compute β-improvement and B to compute marginal gain)
6: $v' \leftarrow \arg\max_{v \in \Gamma \setminus S} \{\lambda_v\}$ and then set $l(v')$ as 0
7: $S \leftarrow S \cup \{v'\}$
8: Update $d(s,t)$ in B as $l(v')$ becomes 0
9: Return S

Thus, the likelihood of having more than one improved node in the shortest path between a randomly selected node pair is low. Consequently, it is likely that the lower bound is tight.

However, an important question remains. *Can we provide bounds on the quality of the iterative greedy solution for POP?* We answer this question through the *sandwich theorem*. The idea of the sandwich theorem is as follows: First, run the greedy algorithm on the actual function ($f(.)$) and its lower bound ($f^r(.)$). Let us say S' and S^r are the produced solution sets respectively and $S = \arg\max_{S \in \{S', S^r\}} f(S)$. Now $f(S)$ has the following lower bound:

THEOREM 7. ***Sandwich Theorem:*** *If f^r is a lower bound of f and f^r is monotone and submodular, then*

$$f(S) \geq \mathscr{C} \cdot (1 - \frac{1}{e}).f(S^*) \tag{20}$$

where $\mathscr{C} = \frac{f^r(S^*)}{f(S^*)}$ and S^* is optimal solution under cardinality constraint for function $f(.)$.

PROOF. $f(S) \geq f^r(S) \geq (1 - \frac{1}{e}) f^r(S^*)$

$\geq \mathscr{C} \cdot (1 - \frac{1}{e}).f(S^*)$ Since $f^r(S^*) = \mathscr{C} f(S^*) \square$

Theorem 7 says that the performance of greedy on POP is directly proportional to the tightness of RPOP. More specifically, $\mathscr{C} = \frac{f^r(S^*)}{f(S^*)}$ quantifies the tightness of RPOP. The closer the ratio is to 1, the better is the approximation quality. Our empirical evaluation in Sec. 4 reveals that $\mathscr{C}$ typically lies in the range $[0.5, 1]$.

3.3 Greedy Selection of Nodes

Algorithm 1, called *GSN*, outlines the pseudocode of the greedy algorithm for POP. In each iteration, it selects the node that produces the maximum marginal gain on the total flow from β-improved pairs given the current solution set, S (step 7). To enable this operation, first, GSN pre-computes the shortest path delays between any node that is part of at least one node pair with a positive flow to all other nodes in the network and stores them in matrices A and B (step 2). Note, we ignore shortest paths between nodes u and v, if neither u nor v is part of some node pair with a positive flow, since such paths do not contribute to the flow improvement. If the network is undirected, computing only the upper half of the distance matrix is enough since distances are symmetric. Stored path delays in A remain unchanged through out the algorithm. On the other hand, B stores the updated shortest path delays following the node upgrades made in S. A is used to determine if a pair has been β-improved and B is used to compute the marginal gain of a node upgrade. S is populated iteratively (3-9) and B is updated in each of these

iterations (line 9). Finally, after k iterations, the solution set S is returned (line 11).

EXAMPLE 2. *Figure 2 shows a possible solution of size $k = 2$ for a small network. The settings are already described before in the main paper. In the first step, GSN can choose either node c or e as both of them individually β-improve (the initial shortest path delay, 15 becomes 5) the pair (d, f) and thus improves flow of $\frac{50}{150}$. If node c is chosen, then in second iteration, GSN will choose node b as it β-improves (the initial distance, 35 becomes 5) the pair (a, d) and thus flow of $\frac{100}{150}$.*

3.3.1 Cost Analysis

Computation Cost: The most important steps in GSN are lines $2, 5$ and 9. In the worst case, line 2 computes all-pairs-shortest-paths in time $O(n^2 \log n)$, where n is the number of vertices in the network. Next, it chooses, among the candidate nodes Γ, the one that maximizes the number of β-improved pairs (line 5), which takes $O(|\Gamma|n^2)$ time. As the shortest path distances are stored, the computation of β-improved pairs takes $O(n^2)$ for each candidate node. After choosing the best node and improving its delay, the shortest path distances are updated in B (line 9) consuming $O(n^2)$ time. Therefore, the total running time of GSN is $O(n^2 \log n + k|\Gamma|n^2)$.

Memory footprint: Since in the worst case we need to store the distance matrix for all pairs shortest path delays, the memory footprint is $O(n^2)$.

Both the computation cost and the memory footprint are prohibitively large for large networks. Consider road networks of large metropolitan cities. As we show in Sec. 4, these networks may contain more than half a million nodes and edges. Repeated shortest path computations on such large networks is not computationally scalable. Although techniques exists to index shortest path computations[10], in our problem the shortest paths are dynamic; they change with each node upgrade. Hence, existing index structures for shortest paths are not applicable.

The scalability bottleneck also extends to the memory footprint. More specifically, storing the all pairs distance matrix incurs a large amount of storage. To provide a concrete example, a network with 500k nodes would require 500GB of RAM to store all pairs shortest paths if each distance value is stored as an integer.

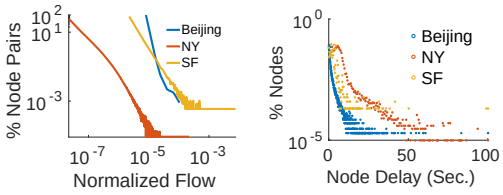
Clearly, a more efficient approach than greedy (Alg. 1) is required. Towards that goal, we expedite greedy through sampling and provide theoretical guarantees on the sampling quality.

3.4 Sampling

As in any sampling algorithm, the goal is to carefully select a subset of the data and compute the answer set by only analyzing this subset. In our particular case, the goal is to sample a subset of node pairs, and execute the greedy algorithm only on the sampled subset. The sampling algorithm is effective if the computed answer set is as good as the answer set that would be computed if the entire dataset is processed. The key step towards ensuring this quality control is to choose a subset that is representative of the entire data for the task in hand. Towards that end, we first consider the naïve approach of uniform sampling.

3.4.1 Uniform Sampling

As the name suggests, in this procedure, we sample uniformly with replacement a set of ordered node pairs $\mathcal{U}$ from the set of all node pairs in $V \times V$. Although the approach is simple, this sampling algorithm under-utilizes the information hidden in node pair flows. More specifically, not all node pairs are of equal importance. In our optimization function (Eq. 2), improving the shortest paths



(a) Flow (b) Delay

Figure 4: **The distribution of (a) node-pair flows and (b) node delays in three real road network datasets.**

on important node pairs has a much more profound impact than improving pairs that have negligible flow between them.

To better understand the impact of ignoring node pair importance, in Figure 4a, we plot their distribution on three real road networks of Beijing, New York, and San Francisco. The flow of a node pair (a, b) is the proportion of taxi trips from location a to b. As can be seen, the distribution follows a power law, which means a small set of node pairs are highly more popular than the remaining majority. Since a small minority of the node pairs contribute majority of the node flows, uniform sampling is unlikely to have a large enough sample of these important node pairs. Consequently, the estimates computed from the sampled sets may suffer. To overcome this drawback, we propose *Importance Sampling*.

3.4.2 Importance sampling

Importance sampling [1, 27] samples elements proportional to their importance. When applied to POP, importance sampling samples node pairs proportional to their flow value. Consequently, the sampling is biased towards node pairs that are more relevant to compute the given estimate. The formal definition is as follows.

DEFINITION 7 (IMPORTANCE SAMPLING). *In importance sampling, a node pair (s, t) is selected (with replacement) in the sampled set $\mathcal{I}$ with probability $\hat{p}_{s,t} = \xi_{s,t}$.*

Next, we show that importance sampling is an unbiased estimator of the entire set. A sampling procedure is *unbiased* if it is possible to estimate the mean of the entire set from the sampled set. More formally, $\mathbb{E}[\hat{\mu}(I)] = \mu_\xi = \frac{\sum \xi_{s,t}}{n(n-1)} = \frac{1}{n(n-1)}$, where $\hat{\mu}(\mathcal{I})$ is the mean estimator. We define $\hat{\mu}(\mathcal{I})$ as the *weighted average* over the samples in $\mathcal{I}$: $\hat{\mu}(\mathcal{I}) = \frac{1}{\sum_{(s,t)\in\mathcal{I}} \hat{w}_{s,t}} \sum_{(s,t)\in\mathcal{I}} \hat{w}_{s,t} \cdot \xi_{s,t}$, where $\hat{w}_{s,t} = \frac{1}{\hat{p}_{s,t}}$. Next, we show $\hat{\mu}(\mathcal{I})$ is an unbiased estimator of μ_ξ.

LEMMA 3.1. *$\hat{\mu}(\mathcal{I})$ is an unbiased estimate of μ_ξ when $\hat{w}_{s,t} = \frac{1}{\hat{p}_{s,t}}$, i.e., $\mathbb{E}[\hat{\mu}(\mathcal{I})] = \mu_\xi$.*

PROOF. $\mathbb{E}[\hat{\mu}(\mathcal{I})] = \frac{1}{\mathbb{E}[\sum_{(s,t)\in\mathcal{I}} \hat{w}_{s,t}]} \cdot \mathbb{E}\left[\sum_{(s,t)\in\mathcal{I}} \hat{w}_{s,t} \cdot \xi_{s,t}\right]$
If we simplify the first term, we obtain

$$\mathbb{E}\left[\sum_{(s,t)\in\mathcal{I}} \hat{w}_{s,t}\right] = |\mathcal{I}|.\mathbb{E}[\hat{w}_{s,t}]$$
$$= |\mathcal{I}|. \sum_{\forall(s,t)\in V\times V} \hat{w}_{s,t} \cdot \xi_{s,t} = n(n-1)|\mathcal{I}|$$

where, $n = |V|$. From the second term, we get,

$$\mathbb{E}\left[\sum_{(s,t)\in\mathcal{I}} \hat{w}_{s,t} \cdot \xi_{s,t}\right] = |\mathcal{I}|\mathbb{E}[\hat{w}_{s,t} \cdot \xi_{s,t}]$$
$$= |\mathcal{I}| \sum_{\forall(s,t)\in V\times V} \hat{p}_{s,t}(\hat{w}_{s,t} \cdot \xi_{s,t}) = |\mathcal{I}|$$

Combining these two, $\mathbb{E}[\hat{\mu}(\mathcal{I})] = \frac{|\mathcal{I}|}{n(n-1)|\mathcal{I}|} = \mu_\xi$. □

Armed with an unbiased estimator, we show that by carefully choosing the size of the sample set, importance sampling provides an accurate estimation of the marginal gain of adding a node to the solution set (line 5 in Alg. 1). More formally, we prove the following.

THEOREM 8. *Let S be the solution set till the current iteration. Furthermore, let $\mathcal{I}$ be the set of sampled node pairs. In this setting, let v_g be the node providing the highest marginal gain on the entire set of node pairs and v_s be the highest one when only considering the sampled set $\mathcal{I}$. The difference in the flow from the β-improved paths by these choices is bounded as follows:*

$$Pr\left[|f(S\cup\{v_g\}) - f(S\cup\{v_s\})| < \epsilon\right] > 1 - \frac{1}{n^2}, \qquad (21)$$

where $|\mathcal{I}|$ is $O(\frac{c\cdot\log n}{\epsilon^2})$, $c = (\frac{\xi_m}{\mu_\xi})^2$, ϵ is the error bound, and ξ_m and μ_ξ are the maximum and average flow of all pairs of nodes respectively.

PROOF. Let $\mu_g = \frac{f(S\cup\{v_g\})}{n(n-1)}$ and $\mu_s = \frac{f(S\cup\{v_s\})}{n(n-1)}$ denote the corresponding means and Y_g and Y_s be the corresponding *expected* means from the samples.

The samples can be viewed as random variables associated with the selection of a pair of nodes. More specifically, the random variable, X_i, is the flow associated with the i-th pair of vertices in the importance sample $\mathcal{I}$. Since the samples provide an unbiased estimate (Lemma 3.1) and are i.i.d., we can apply *Hoeffding's inequality* [8] to bound the error of the mean estimates:

$$Pr[|Y_g - \mu_g| \geq \theta] \leq \delta \qquad (22)$$

where $\delta = 2\exp\left(-\frac{2|\mathcal{I}|^2\theta^2}{\mathcal{T}}\right)$, $\mathcal{T} = \sum_{i=1}^{|\mathcal{I}|}(b_i - a_i)^2$, where each X_i is strictly bounded by the intervals $[a_i, b_i]$. Similarly,

$$Pr[|Y_s - \mu_s| \geq \theta] \leq \delta \qquad (23)$$

Applying union bound,

$$Pr[(|Y_g - \mu_g| \geq \theta) \cup (|Y_s - \mu_s| \geq \theta)] \leq 2\delta \qquad (24)$$

By construction, $\mu_g \geq \mu_s$ as GSN selects the best next node at each step. On the other hand, if importance sampling selects v_s, it must be that $Y_s \geq Y_g$. As, the sampled best node is probabilistic, we need to apply *union bound* over n possible nodes. As a consequence, $Pr[|\mu_g - \mu_s| \geq 2\theta] \leq 2n\delta$ and $Pr[|\mu_g - \mu_s| < 2\theta] > 1 - 2n\delta$. Now, $Pr\left[|f(S\cup\{v_g\}) - f(S\cup\{v_s\})| < \epsilon\right]$

$$= Pr[|\mu_a - \mu_g| < \frac{\epsilon}{n(n-1)}]$$

$$> 1 - 4n\exp\left(-\frac{2|\mathcal{I}|^2\left(\frac{\epsilon}{2n(n-1)}\right)^2}{\mathcal{T}}\right) \qquad (25)$$

Since the average flow $\mu_\xi = \frac{1}{n(n-1)}$ and $\xi_m \geq b_i, a_i \geq 0$, we get $\mathcal{T} \leq |\mathcal{I}|\cdot\xi_m^2$. Combining these factors, we get,

$$1 - 4n\exp\left(-\frac{2|\mathcal{I}|^2\left(\frac{\epsilon}{2n(n-1)}\right)^2}{\mathcal{T}}\right)$$

$$> 1 - 4n\exp\left(-\frac{2|\mathcal{I}|^2\left(\frac{\epsilon\cdot\mu_\xi}{2}\right)^2}{|\mathcal{I}|\cdot\xi_m^2}\right) \qquad (26)$$

Algorithm 2: Importance Sampling's Selection (ISS)

Require: Network $G = (V, E, L)$, Approximation error ϵ, Sampling factor c, Budget k, candidate node set Γ, network flows $\mathbb{F}$, β.
Ensure: A subset of k nodes
1: Choose $O(c(\log n)/\epsilon^2)$ sample pairs of vertices in $\mathcal{I}$ via importance sampling
2: $A \leftarrow$ A distance matrix containing the distance of all nodes appearing in the sampled set $|\mathcal{I}|$ to all other nodes in the network.
3: $S \leftarrow \Phi$
4: **while** $|S| < k$ **do**
5: **for** $(s,t) \in \mathcal{I}$ **do**
6: $B \leftarrow$ Re-compute the distances in matrix A after considering the upgraded nodes in S.
7: **for** $v \in \Gamma$ **do**
8: $\lambda_v \leftarrow \sum_{(x,y)\in\Lambda_s} \xi_{x,y}$ where Λ_s is the set of new β-improved pairs in $\mathcal{I}$ when $l(v) = 0$ (Use A to compute β-improvement and B to compute marginal gain)
9: $v' \leftarrow \arg\max_{v\in\Gamma\setminus S}\{\lambda_v\}, l(v') \leftarrow 0$
10: $S \leftarrow S \cup \{v'\}$
11: Return S

Combining Eq. 25 and Eq. 26, we get

$$Pr\left[|f(S\cup\{v_g\}) - f(S\cup\{v_s\})| < \epsilon\right]$$

$$> 1 - 4n\exp\left(-\frac{|\mathcal{I}|\left(\epsilon.\mu_\xi\right)^2}{2\xi_m^2}\right) \qquad (27)$$

By setting the number of samples $|\mathcal{I}| = \frac{2\xi_m^2\cdot\log(4n^3)}{(\epsilon\cdot\mu_\xi)^2}$, we have

$$Pr\left[|f(S\cup\{v_g\}) - f(S\cup\{v_s\})| < \epsilon\right] > 1 - \frac{1}{n^2}$$

Remarks: Theorem 8 tells us the number of samples required to keep the approximation error, with respect to the greedy approach in Alg. 1, sufficiently low. The key results from this theorem are as follows.

- $|\mathcal{I}|$, which is the sample size, is a function of the error ϵ and inversely proportional to ϵ. Thus, with higher number of samples, our estimates get more accurate.
- The sample size grows logarithmically with the network size.

3.4.3 *Efficient greedy through Importance Sampling*

Armed with Theorem 8, we next describe how Importance Sampling is used to speed up greedy. We call this algorithm *Importance Sampling's Selection (ISS)*. Alg. 2 presents the pseudocode. In simple terms, Theorem 8 is used to first decide the sample size (line 1), and the Greedy selection of node upgrades is performed by analyzing the impact of an upgrade only on the sampled set of node pairs. As in greedy(Alg. 1), the algorithm runs for k iterations and in each iteration, the best node from the candidate set is selected based on the sampled node pairs (lines 4-13). Instead of computing the entire distance matrix, we compute the distance from all nodes in the network to only those nodes that appears in at least one sampled pair (lines 2 and 6). These distances provide the β-improved pairs in the sampled set $\mathcal{I}$ of pairs of nodes. The remaining operations remain same as in greedy (Alg. 1).

Computation Cost: First, we sample $|\mathcal{I}|$ node pairs based on importance sampling. Recall from Def. 6, that the importance of a node pair (u, v) is the proportion of flows from u to v. Let d be the total number of network flows (Def. 2) in the dataset and each flow is assigned an ID from 1 to d. To sample $|\mathcal{I}|$ node pairs proportional to their importance, we generate $|\mathcal{I}|$ random IDs in the range $[1, d]$ and extract the corresponding flows. The originating and destination nodes of each of these sampled flows form a node pair in our sample set. The sampling procedure consumes $O(|\mathcal{I}|)$ time. As defined in Def. 7, the sampling is performed *with replacement*. Hence,

Name	#Trajectories	Type	$\|V\|$	$\|E\|$
Bejing (BJ)	123K	Directed	623.9K	672.2K
San Fransisco (SF)	442K	Undirected	5.08K	41.7K
New York (NY)	49247K	Directed	72.7K	169.8K

Table 3: **Dataset description and statistics.**

the number of times a node pair appears in $\mathcal{I}$ is proportional to its importance.

The costliest steps of our algorithm are lines 2, 5-7 and 8-12. In line 2, we compute a distance matrix A, which stores the initial shortest path distances between all nodes that appear in some pair in $|\mathcal{I}|$ to all other nodes in the network. This computation consumes $O(|\mathcal{I}|n \log n)$ time since the shortest path is computed between each node of a sampled pair to all other nodes in the network. In steps 5-7, ISS recomputes the shortest paths based on the node upgrades made so far. This operation again consumes $O(|\mathcal{I}|n \log n)$ time. Next, the algorithm estimates the additional number of β-improved pairs after removing the delay of each of the nodes in the candidate set Γ. (steps 8-10). As the shortest path distances of all sampled nodes are stored in A, the computation of β-improved pairs takes $O(|\mathcal{I}|)$ for each candidate node upgrade. Therefore, the best node to be upgraded is selected in $O(|\Gamma||\mathcal{I}|)$ time (step 11-12). This entire process from line 5-12 is then repeated k times to select the k nodes upgrade to be performed. Combining all these factors, the total running time of ISS is $O(k|\mathcal{I}|n(\log n) + k|\Gamma||\mathcal{I}|)$.

Since $|\mathcal{I}|$ is a logarithmic function of n (i.e., $|V|$), the complexity of ISS reduces to $O(kn(\log^2 n) + k|\Gamma| \log n)$. Recall that the computation cost of greedy (Alg. 1) is $O(n^2 \log n + k|\Gamma|n^2)$. Consequently, we get a dramatic reduction in running time.

Storage Cost: In addition to the network ($O(|V| + |E|)$ space), ISS stores $O(|\mathcal{I}|)$ node pairs and their importances. Note that the network flows are not maintained in memory. They are kept in disk and only the sampled flows are extracted and loaded into memory to compute the node pair importance values. The matrices A and B consume $O(|\mathcal{I}||V|)$ space. Since $|\mathcal{I}|$ is a logarithmic function of n (i.e., $|V|$), the space complexity is bounded by $O(|\mathcal{I}| + |\mathcal{I}||V| + |V| + |E|) = O(n \log n + m)$.

4. EXPERIMENTS

In this section, we benchmark the proposed algorithms and evaluate their approximation quality and scalability.

4.1 Experimental Setup

All experiments are performed using codes written in Java on an Intel(R) Xeon(R) E5-2609 8-core machine with 2.5 GHz CPU and 512 GB RAM running Linux Ubuntu 14.04.

4.1.1 Baselines

We denote our main algorithm based on importance sampling as *ISS*. The optimal algorithm based on mixed integer programming is denoted as *MIP-OPT* and the greedy algorithm as *GSN*. The other baselines with which the proposed algorithms have been compared with are as follows:

(1) **USS (Uniform Sampling's Selection):** We adapt the state-of-the-art method [15] of uniform sampling and apply towards our problem. The algorithm samples node pairs with equal probability from the set of all pairs ignoring the importance (flow) associated with each pair.

(2) **High-Cen [30]:** We choose the top-k most central nodes to improve. The central nodes are present on the maximum number of distinct shortest paths and therefore could potentially β-improve a large number of node pairs. However, this method does not capture the dependency among node upgrades.

(3) **High-Delay:** This baseline selects the top-k nodes with the highest delays (weights).

4.1.2 Datasets:

Table 3 summarizes the real-world datasets used in our experiments. Each dataset contains the road network of a city. The road network of each city is extracted from *OpenStreetMap*[1]. Each node corresponds to a region and an edge denotes a street connecting these regions.

For network flows, we use the cab trajectory data from each of the cities listed in Table 3.

(1) **Beijing (BJ)[31]:** We have trajectories of cabs, over a period of 1 week. Each trajectory contains the sequence of nodes visited in a trip and the timestamps at which the nodes were visited.

(3) **San Francisco (SF)[24]:** The dataset has been collected over a duration of one month and contains taxi pick-up and drop-off information from taxis in San Francisco.

(4) **New York (NY) [7]:** This is the largest publicly available taxi dataset. It was collected over a period of four years ranging from 2009 to 2013. It contains records of yellow and green taxis in the city of New York. We use the trajectories from January, 2013 to March, 2013 to find the importance of the node pairs.

To compute the delay in a node, we first partition the cab trajectories based on their starting timestamp into four windows of six hours each viz. 00:00 to 06:00, 06:00 to 12:00, and so on. Next, for each edge, we compute the average time taken to cross it in each of the four time slots. The delay in an edge is quantified as the difference between the maximum and the minimum average times across the four windows. Finally, the delay in a node is set to the sum of the delays in all of its incoming edges.

4.1.3 Performance metric and parameters

Performance Metric: The quality of a solution set S in a network N is defined in Eq. 2. We call this metric the *flow improvement* due to S and is denoted as $FI(N)$. From our formulation of node-pair importance in Def. 6, the total flow in a network is 1.

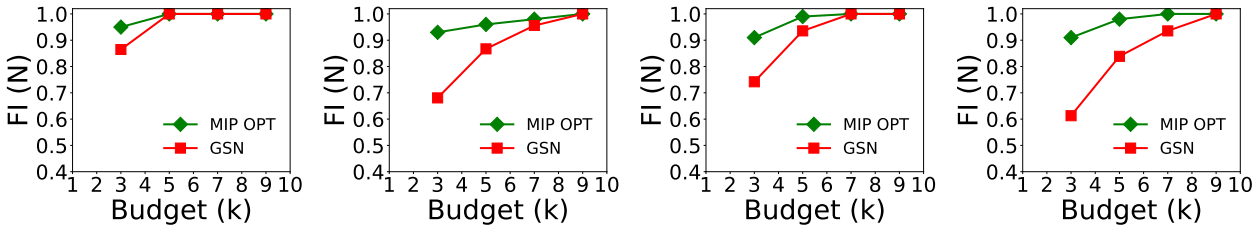
The solution set S produced by any algorithm is evaluated based on $f(S) = \sum_{(u,v)\in\Lambda_S} \xi_{u,v}$ where Λ_S denotes the set of β-improved node pairs. For large graphs, this evaluation is time consuming as it involves all-pair-shortest-paths computation. To mitigate this scalability bottleneck, we evaluate the algorithms based on randomly chosen sampled pairs Q. More specifically, the metric is $f(S) = \sum_{(u,v)\in\Lambda^*_S} \xi_{u,v}$, where Λ^*_S denotes the set of β-improved node pairs in Q. The size of the sample set Q is set to $50,000$.

Default Parameters: We set Γ, which is the candidate set of nodes that can be improved, to V, i.e., the set of all nodes. Unless specifically mentioned, the default value for β is 0.1 or 10%, and the default size of sample set used in ISS (and USS) is $15\log(n)$. Note that, we use the number samples in the form of $c\log(n)$ where c controls the error ϵ (mentioned in Theorem 8).

4.2 GSN and optimal MIP

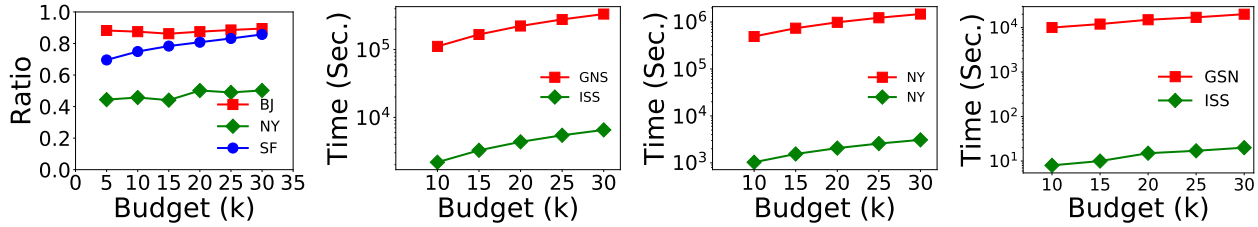
First, we compare the performance of GSN with the optimal solution. As described in Sec. 3.1, the optimal solution is computed using mixed integer programming (MIP). We implement MIP using CPLEX and validate on the SF and BJ datasets. However, we extract a sub-network containing only 1000 nodes from these datasets since MIP consumes exorbitantly high running times on larger networks. Figure 5 presents the results as we vary β and the budget k. Across both datasets, GNS produces results that are close to optimal. More specifically, beyond $k = 5$, GSN is at most 10% away from the flow improvement achieved by the optimal algorithm. This result validates our intuition that greedy is an effective heuristic for the proposed problem. Moreover, GSN takes only a few seconds (< 100 seconds) to produce the nearly optimal results whereas MIP

[1] `http://openstreetmap.org/`



(a) SF ($\beta = 5\%$) (b) SF ($\beta = 10\%$) (c) BJ ($\beta = 5\%$) (d) BJ ($\beta = 10\%$)

Figure 5: **Comparison between MIP vs GSN: Normalized Flow Improvement for (a-b) SF, (c-d) BJ.**



(a) Ratio (b) Running Time in BJ (c) Running Time in NY (d) Running Time in SF

Figure 6: **(a) Approximation quality of ISS against GSN. (b-d)Running times of ISS and GSN.**

takes more than a day's time to terminate. A pattern consistent across both datasets is that as β increases, the gap between GSN and optimal increases as well. This trend is a direct consequence of the fact that when β increases, only a group of node upgrades can β-improve a path and hence higher is the need to fully search the combinatorial space and identify the best group. Since GSN does not explore the entire combinatorial space, the performance is worse than optimal. Nonetheless, this gap is small.

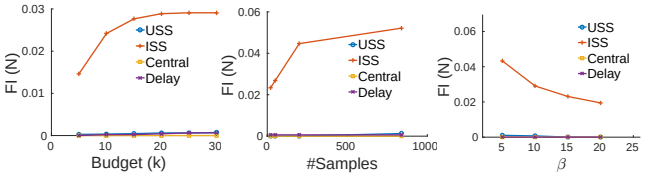
4.3 ISS vs GSN

We next compare the performance of GSN with ISS. In Fig. 6a, we plot the approximation quality of ISS with respect to GSN. The approximation quality is the ratio between the flow improvements produced by ISS and GSN. A high ratio indicates that ISS produces similar quality results as that of GSN. As can be seen, the ratio is the highest in BJ, followed by SF and finally NY. This result follows directly from the distribution of node pair importances. Specifically, it is evident from Fig. 4a, that the node-pair importances are most skewed in BJ, followed by SF, and finally NY. When the distribution is skewed towards a small number of important node pairs, importance sampling is better able to estimate the marginal gain of a node upgrade from just the sampled collection of node pairs. This results in the trend visible in Fig. 6a.

Next, we analyze the running times of GSN and ISS. Figs. 6b-6d present the results. As can be seen, ISS is up to three orders of magnitudes faster than GSN. GSN finds it most difficult to scale in the NY dataset, where it consumes around 17 days (410 hours) to terminate. Although, NY is a smaller network than BJ, GSN consumes more time since the number of node pairs with non-zero importance is much higher in NY. To estimate the marginal gain of upgrading a node, GSN needs to recompute the shortest path distances between all node pairs with non-zero importance. Consequently, GSN finds it most difficult to scale in NY.

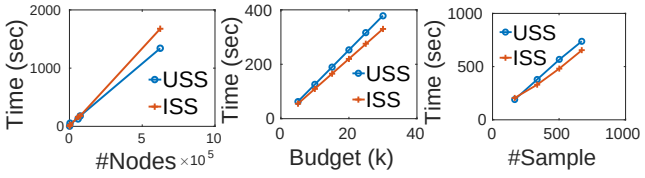
4.4 Comparison with scalable baselines

Next, we compare the performance of ISS with the baseline algorithms listed in Sec. 4.1.1. We omit GSN from further experiments since it fails to scale. Tables 4 and 5 present the results for NY and BJ respectively. To highlight the efficacy of ISS more prominently, we report the *performance improvement* of ISS in terms of ratio. More specifically, each cell reports the ratio



(a) Varying Budget (b) Varying #Sample (c) Varying β (%)

Figure 7: **(NY) The flow improvement (FI) by varying (a) budget, (b) the number of samples, and (c) β (%).**



(a) Scalability (b) Vs Budget (c) Vs #Sample

Figure 8: **(a-c) The running times of ISS and USS by varying (a) (All) the size of graphs, (b) (NY) the budget, (c) (NY) the number of samples.**

$$\text{Performance improvement}(X) = \frac{FI_{ISS}}{FI_X} \tag{28}$$

where X is the method corresponding to the cell's column and FI_X is the flow improvement achieved by that method.

Across both NY and BJ, ISS drastically outperforms all baselines. Some baselines fail to provide any flow improvement at all and therefore the improvement ratio in those cases is ∞. This impressive performance of ISS stems from two factors. First, unlike ISS, both High-Cen and High-Delay are oblivious to the dependencies between the nodes selected for reduction. Second, although this problem does not exist in USS, USS samples node pairs uniformly and therefore fails to capture an adequate representation of the important node pairs. Consequently the performance suffers.

A consistent trend we see across both NY and BJ is that as β increases the performance gap between ISS and other baselines increases. This is another natural consequence of being oblivious to the node dependencies. More specifically, at a higher β, it is even more important to know the other nodes in the solution set since such high improvement is typically possible only when all nodes in

	$\beta = 5\%$			$\beta = 10\%$		
k	**High-Cen**	**USS**	**High-Delay**	**High-Cen**	**USS**	**High-Delay**
5	∞	27.5	350	∞	50.3	2085
10	36000	37.5	110	∞	69.6	115
15	40000	40	80	∞	46	120
20	42000	42	56.5	20624	47.2	72
25	43000	43	54.8	20772	47.5	46
30	21500	43	54	21000	40.9	46

Table 4: **(NY) Comparison of *Flow Improvement* between ISS and other baselines on NY against varying budget (k) and β. Each cell reports the relative Improved Flow w.r.t. ISS, i.e. $\frac{FI_{ISS}}{FI_X}$, where X is the method used in that particular cell. When X does not produce any FI ($FI_X = 0$), $\frac{FI_{ISS}}{FI_X} = \infty$.**

	$\beta = 5\%$			$\beta = 10\%$		
k	**High-Cen**	**USS**	**High-Delay**	**High-Cen**	**USS**	**High-Delay**
5	4.0	25.1	∞	∞	∞	∞
10	9.1	10.3	∞	∞	∞	∞
15	5.1	12.2	∞	9.1	∞	∞
20	2.5	14.8	∞	6.2	∞	∞
25	1.8	16.8	∞	3.1	∞	∞
30	1.7	17.7	∞	2.4	∞	∞

Table 5: **(BJ) Comparison of *Flow Improvement* between ISS and other baselines on BJ against varying budget (k) and β.**

the solution set collectively to bring down the shortest path delay by $\beta\%$. Finally, we point out that the performance improvement is higher in BJ than in NY. Since BJ road network is significantly larger than NY, attaining β improvement within the given budget is a harder task in BJ. While ISS is able to cope with this task, other techniques fail to provide much improvement. Hence the ratio is ∞ in BJ for most comparisons.

4.5 Impact of Parameters on Performance

The key parameters impacting the performance are the budget k, the number of samples, and β. Note that the number of samples affects the error ϵ (Theorem 8). The running time is affected by the budget, the number of samples and the size of the graph. We study the impact of these parameters on the quality and running time.

4.5.1 Quality

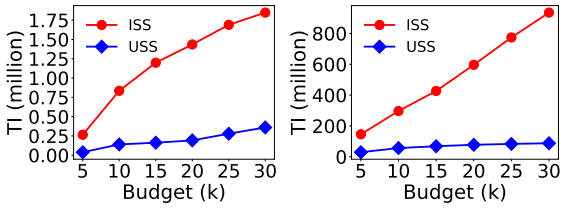
First, we evaluate the quality in terms of Flow Improvement (FI) against varying budget. Figure 7a shows the results for NY. As the budget increases ISS shows growth in improving the flows and outperforms all other baselines convincingly.

Next, we vary the number of samples and observe its effects in Figure 7b. The budget (k) is set to 50. Though ISS is able to show a steady growth in quality, others struggle to make a noticeable impact. This behavior is a direct consequence of not being sensitive to the importance of node pairs. More specifically, all techniques except ISS randomly sample node pairs. As shown in Fig. 4a, the node pair importance follows a power-law distribution and thus a small minority of the pairs contribute majority of the network flows. The chances of these important pairs to get randomly sampled is extremely low and thus the performance of the other baselines suffer.

Finally, we show the quality of the algorithms varying β. With the increase of β, the possibility of improving a node pair reduces and thus the number of improved flows should reduce. Figure 7c shows the results that validates this intuition. Consistent with all previous results, ISS performs significantly better than the baselines.

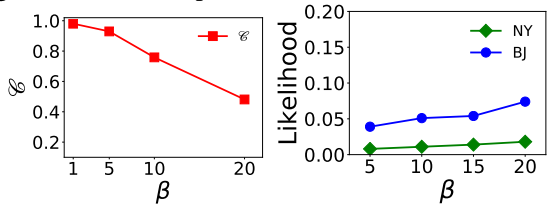
4.5.2 Scalability

First, we study the growth rate of running time against the network size. In this experiment, we compute the running time for each of the datasets listed in Table 3 and verify how the running time grows with respect to the network size. Figure 8a presents the results. The running time grows at a rate that is slightly higher



(a) BJ (b) NY

Figure 9: **The total improvement (TI) on (a) BJ and (c) NY.**



(a) $\mathscr{C}$ in BJ (b) Likelihood

Figure 10: **(a) Tightness of RPOP and (b) The likelihood to have more than one improved node in the shortest path between a randomly selected node pair against β (in percentage).**

than linear with increase in network size. This result is consistent with the theoretical analysis of the computation cost (Section 3.4.3), where we show that the computation cost of ISS grows at $O(n \log^2 n)$ with respect to the network size. On the largest network of Beijing, ISS finishes with 30 minutes.

Next, we evaluate the growth rate of running time against budget on the NY dataset. Figure 8b presents the results. As expected from the theoretical analysis of computation cost in Section 3.4.3, the running time grows linearly and consumes less than 7 minutes across all values of k.

Finally, we analyze scalability against the number of samples on NY dataset and present the results in Fig. 8c. Here, we fix the budget as 30 and vary the number of samples. As expected, the running time grows linearly with the number of samples. Overall, these results firmly establish that ISS is able to overcome the scalability challenges without compromising significantly on quality.

4.6 Total Improvement

In our analysis, we have argued that noticeable improvement for each individual pair of nodes is important. Does this focus on individual improvement compromise on the total improvement across the entire network? In the next experiment, we analyze this question by measuring the performance of ISS on the *total improvement (TI)* metric. TI is the total reduction in delay (in minutes) across all trajectories of the datasets. Mathematically, TI is defined as follows.

$$TI = \sum_{\forall P_{u,v} \in \mathbb{F}} \big(d(u,v) - d(u,v;S)\big) \tag{29}$$

Here, $\mathbb{F}$ is the set of all trajectories, $P_{u,v}$ denotes a trajectory that starts at node u and terminates at v. S is the set of upgraded nodes and $d(u,v)$ and $d(u,v;S)$ are the delays of the shortest paths from u to v before node upgrades and after node upgrades respectively. We compare the performance of ISS with the state-of-the-art method USS [15], which is designed specifically for total improvement. For ISS, we set $\beta = 0$ since the goal is to optimize total improvement. Fig. 9 presents the results in NY and BJ. As visible, ISS is up to 8 times better than USS. USS is blind to the idea of node pair importance and hence its performance suffers.

4.7 Tightness of RPOP

Recall that our motivation to use greedy as the optimization strategy for POP emerged from the observation that RPOP is a lower

	Gaussian		Uniform	
k	High-Cen	USS	High-Cen	USS
5	∞	∞	∞	∞
10	∞	829	1700	∞
15	∞	1400	5800	1600
20	∞	1600	6900	1900
25	∞	1700	8300	2200
30	∞	1900	9200	2500

Table 6: **(Synthetic Delay:) Comparison of *Flow Improvement* between ISS and other baselines on NY against varying budget (k) and type of synthetic delays.**

	Gaussian		Uniform	
k	High-Cen	USS	High-Cen	USS
5	∞	2.2	∞	2.2
10	∞	2.7	∞	2.6
15	∞	2.0	∞	2.0
20	∞	2.3	∞	2.3
25	∞	2.4	∞	2.3
30	∞	2.5	∞	2.3

Table 7: **(Synthetic Flows/Importance:) Comparison of *Flow Improvement* between ISS and other baselines on NY against varying budget (k) and type of synthetic flows/importance.**

bound of POP (Section 3.2) and greedy is the optimal polynomial-time algorithm for RPOP. Theorem 7 establishes an error bound on the performance of greedy on POP as a function of the tightness factor $\mathscr{C} = \frac{f^r(S)}{f(S)}$, between RPOP and POP. In the next, experiment, we analyze how the tightness varies with increase in β.

Fig. 10a presents the results in the BJ dataset at $k = 30$. As can be seen, $\mathscr{C}$ decreases with increase in β. This behavior is not surprising. When β is high, it is extremely hard for a single node upgrade to β-improve a path. Consequently, multiple node upgrades in a path are necessary to achieve β-improvement. Since RPOP allows only one node upgrade per path, $f^r(S)$ (RPOP) stays much lower than $f(S)$ (POP). As a result $\mathscr{C}$ decreases. Overall, $\mathscr{C}$ lies in the range $[0.5, 1]$. It is important to note that Theorem 7 provides the worst-case approximation error of greedy. In practice, as discussed in Section 4.2, greedy (GSN) is typically within 90% of the optimal.

We next analyze another assumption regarding RPOP. We claim in Section 3.2 that intuitively, in most real life networks the number of node pairs is much larger than the budget k. Thus, the likelihood of having more than one improved node in the shortest path between a randomly selected node pair is low. Consequently, POP should behave similarly to RPOP. We empirically evaluate this likelihood in real large datasets of BJ and NY. Fig. 10b shows that our assumption is indeed true and the likelihood is less than 0.1 (i.e., 1%) in BJ and 0.08 in NY. k is set to 30 in this experiment.

4.8 Experiments on Synthetic data

The three data properties that have a profound impact on the quality of the approximation algorithms are: 1) the delay distribution of nodes, 2) the flow/importance distribution of the node pairs, and 3) the graph structure itself. In this section, we systematically pick each of these properties and study its impact on the performance. Towards that end, let X be the property under study. To isolate the impact of property X, we pick a real dataset and synthetically alter its property X while keeping the other two properties intact. Next, we evaluate the performance of each of the algorithms on this new dataset. Note that the running time of ISS (and USS) only depends on the sample set size, which is a function of the network size. Hence, the efficiency of ISS is shielded from variation in the above three properties.

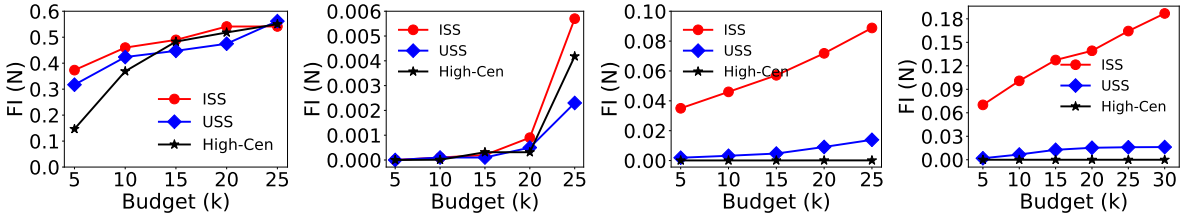
1) Synthetic Delay: As shown in Fig. 4b, the delay distributions in real transportation networks are highly skewed. Specifically, only a small minority of nodes face high delays. In this experiment, we benchmark the performance of ISS on uniform and standard normal distributions. Towards that end, we pick the NY dataset, and assign node delays synthetically from $\mathcal{U}(0, 1)$ and $\mathcal{N}(0, 1)$, while retaining the original node pair flows and network structure. In the case of $\mathcal{N}(0, 1)$, it may generate negative numbers. Since negative delay is not feasible, we add the value of the minimum delay to all the delays to make them non-negative. Table 6 presents the Performance Improvement ratio (Eq. 28). Clearly, regardless of the distribution, ISS is significantly better than the competing baselines. This superior performance of ISS is a direct consequence of the other baselines being ignorant of node pair flows.

2) Synthetic Node Pair Flows: In this experiment, we alter the NY dataset by assigning the node pair flows synthetically. First, we assign flow values to node pairs from uniform distribution $\mathcal{U}(0, 1)$ and standard normal distribution $\mathcal{N}(0, 1)$. As in the case of synthetic node delays, we shift the normal distribution to exclusively non-negative values by adding the minimum of all flow values. Finally, for both Uniform and Normal, the flow values are normalized by dividing them with the sum of all node pair flows. Note that the NY dataset contains more than 5 billion node pairs and assigning flow values to all of them is not computationally feasible. In addition, to normalize the flow values, we also need to store them, and this memory requirement is prohibitively large. To mitigate this issue, we randomly pick 100 million unique node pairs and assign flow values to only these. The rest of the node pairs are assumed to have 0 importance. As a comparison, the real NY dataset has ≈ 11 million node pairs with positive flow.

Table 7 shows the results in terms of performance improvement ratio (Eq. 28). Consistent with previous trends, ISS outperforms both baselines. However, the gap between ISS and USS is much smaller compared to their performance on the real unaltered datasets. This result is expected since in the real dataset the node-pair flows follow a power-law distribution (Fig. 4a). In a power law distribution, only a small set of node pairs contribute significantly to the flow improvement. These highly important pairs are also likely to be present in the importance sample set. Consequently, the estimate of a node upgrade computed from this sample set is accurate. In uniform or normal distributions, since a large number of node pairs contribute to the flow improvement, a small sample set is unable to capture the entire picture. As a result, the gap between ISS and USS reduces. We also observe that the performance of High-Cen is worse than both ISS and USS. Since the flows of node pairs are synthetic, the likelihood that a central node will lie on the shortest path between an important pair of nodes is low. Hence, we see the effect visible in Table 7.

3) Synthetic Network Structure: Finally, we investigate the impact of the network structure. We generate synthetic network structures from three well-studied models: (a) Barabasi-Albert (BA), (b) Watts-Strogatz (WS) and (c) Erdos-Renyi (ER). We choose these three models since collectively they cover most of the network structure properties found in the real world. BA has *scale-free* property where node degree distribution follows a *power law* distribution. WS produces graphs with *small-world* properties, where average path delays (path lengths) are short and the graphs have a high clustering coefficient. Different from BA and WS, ER generates random graphs where any possible edge is created with equal probability.

As in the previous two experiments, we alter the NY dataset by synthetically constructing the network structure, while retaining the original node-pair and delay distributions. Specifically, we first construct a network through one of the graph generation models where the size of the network in terms of number of nodes is the same as the original NY network. Since the number of nodes is the same, we create a mapping from each node in the original structure to the synthetically generated one. Based on this mapping, we assign delays and node-pair importances in the synthetic dataset. Note that



(a) BA (b) WS (c) ER (d) Edge-NY

Figure 11: **(a-c) Synthetic Network Structure: Comparison of *Flow Improvement* FI(N) between ISS and other baselines against varying budget (k) on (a) BA, (b) WS, and (c) ER graph generation models. (d) Comparison of *Flow Improvement* FI(N) between ISS and other baselines against varying budget (k) when the delays are on the edges.**

although the number of nodes is the same, the number of edges would be different.

Figs. 11a-11c present the results. Several key insights emerge from this experiment. First, ISS is the best performing technique across all models of graph structure. Second, all techniques achieve high flow improvement in BA (Fig. 11a). Since BA has scale-free property, there are few nodes of extreme high degree. These nodes typically also have high centrality and thus, reducing the delays in these central nodes bring a reduction in the shortest path delays among a large number of node pairs. Due to this same reason, all three techniques provide similar flow improvements in the BA model, particularly at a high value of k. In contrast, the flow improvement problem is most difficult in WS. WS generates small-world networks where the average shortest path delays are small (in terms of number of hops and not delays). The flow improvement problem is easier when many shortest paths go through a central node, since in such a case improving the central node improves many node pairs. When shortest paths are small, the likelihood of two shortest paths going through a common node is also small. Consequently, achieving a high flow improvement through a small number of node upgrades is difficult. Nonetheless, as the budget grows, ISS begins to outperform other baselines. For example, at $k = 30$, ISS is 3 times better than USS and 1.5 times better than High-Cen.

The gap between ISS and the baselines is most pronounced in ER. The structure of ER is random by construction. In this scenario, the flows of the node pairs and the delays of the nodes play a crucial role. Unlike USS and High-Cen, ISS takes into account these aspects and provides up to 5 times more flow improvement.

4.9 Edge Delays

Recall from Sec. 2, that the proposed algorithm can incorporate delays in either nodes, edges or a mixture of both. To showcase this ability, we next measure the performance of ISS with delays on edges. The delay on an edge is computed as discussed in Sec. 4.1.2. Fig. 11d presents the results. Consistent with previous results, ISS outperforms USS and High-Cen significantly. ISS is up to 36 times better than the next best baseline USS. High-Cen does not provide any substantial improvement as the notion of *edge centrality*, i.e.,the number of node pairs with at least one shortest path going through the edge, is an even weaker indicator of a good upgrade than node centrality. Consequently, the performance suffers.

5. RELATED WORK

The majority of work on network design target various objectives via modifying the network structure and attributes. The problems differ in the upgrade models and the objective functions.

Paik et al. [20] first introduced a set of design problems in which vertex upgrades improve the delays of adjacent edges. Later, Krumke et al. [11] generalized this model assuming varying costs for vertex/edge upgrades and proposed algorithms to minimize the cost of the minimum spanning trees. Lin et al. [13] also proposed the shortest path improvement problem where the weights are associated with undirected edges. In all of the above problems, the upgrade models are different and cannot be used to solve out problem.

The closest works to our problem are [6] and [15]. Though they consider a variation of POP, our formulation is different in the sense that we target *significant* improvement of the *important* paths. Both these techniques do not capture the intricacies of significant improvement and node pair importance and thus, as shown in our comparative evaluation, the performance suffers (Section 4).

Network design problems to improve several global objectives (vertex eccentricity, diameter, all-pairs shortest paths etc.) by *addition of edges* have been addressed in the past literature [17, 21, 22, 5, 23]. Meyerson et al. [17] designed approximation algorithms for single source and all pairs shortest path minimization. Demaine et al. [5] proposed a constant factor approximation algorithm to minimize the diameter of a network by adding shortcut edges. In addition, their algorithm solves the problem of minimizing eccentricity of a particular node. Prior work has also studied the eccentricity minimization problem in a composite network [23]. All of the above problems, however, consider structural modification (addition of new edges), and hence are complementary to our setting.

Other related problems involve efficient computation of network centrality. In [25], the authors compute top-k nodes based on betweenness centrality via sampling. The group betweenness problem has been solved in almost linear time [30, 14] by a high quality probabilistic approximation algorithm. The design problems to improve the shortest path based centralities of nodes had been studied in recent past [16, 4, 9, 28]. In these works, the shortest path based centralities has been improved via edge addition. These words differ from the proposed problem in both the upgrade model as well as the objective function.

6. CONCLUSION

In this paper, we studied and proposed solutions for a novel network design problem of delay minimization. Different from existing techniques, our formulation incorporated the practical considerations that the impact of delay minimization should be noticeable and favor important paths in a network. The proposed problem has diverse applications in a variety of domains including road, airline, power and communication networks. We showed that the problem is NP-hard as well as APX-hard and cannot be approximated within a factor greater than $(1 - \frac{1}{e})$. To overcome the exponential cost of the optimal solution, we proposed an importance sampling based algorithm with provable quality guarantees. Through extensive evaluation on multiple real-word traffic networks, we established that importance sampling is accurate and up to three orders of magnitude faster than the greedy approach. In addition, importance sampling produces flow improvement that is up to 70 times better than the state-of-the-art technique. Finally, our experiments on synthetic datasets established that ISS is robust to variation in network structure, delay distributions and node-pair importances.

7. REFERENCES

[1] S. Asmussen and P. W. Glynn. *Stochastic simulation: algorithms and analysis*, volume 57. Springer Science & Business Media, 2007.

[2] P. Banerjee, S. Ranu, and S. Raghavan. Inferring uncertain trajectories from partial observations. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 30–39. IEEE, 2014.

[3] P. Crescenzi, G. D'Angelo, L. Severini, and Y. Velaj. Greedily improving our own centrality in a network. In *SEA*, pages 43–55. Springer International Publishing, 2015.

[4] P. Crescenzi, G. D'angelo, L. Severini, and Y. Velaj. Greedily improving our own closeness centrality in a network. *ACM Trans. Knowl. Discov. Data*, 11(1), 2016.

[5] E. D. Demaine and M. Zadimoghaddam. Minimizing the diameter of a network using shortcut edges. *in SWAT, ser.Lecture Notes in Computer Science, H. Kaplan,Ed.*, pages 420–431, 2010.

[6] B. Dilkina, K. J. Lai, and C. P. Gomes. Upgrading shortest paths in networks. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 76–91. Springer, 2011.

[7] D. Donovan, Brian; Work. New york city taxi trip data (2010-2013), 2016.

[8] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[9] V. Ishakian, D. Erdos, E. Terzi, and A. Bestavros. A framework for the evaluation and management of network centrality. In *SDM*, pages 427–438. SIAM, 2012.

[10] M. Jiang, A. W.-C. Fu, R. C.-W. Wong, and Y. Xu. Hop doubling label indexing for point-to-point distance querying on scale-free networks. *Proc. VLDB Endow.*, 7(12):1203–1214, Aug. 2014.

[11] S. Krumke, M. Marathe, H. Noltemeier, R. Ravi, and S. Ravi. Approximation algorithms for certain network improvement problems. *Journal of Combinatorial Optimization*, 2:257–288, 1998.

[12] Z. Li, R. A. Hassan, M. Shahidehpour, S. Bahramirad, and A. Khodaei. A hierarchical framework for intelligent traffic management in smart cities. *IEEE Transactions on Smart Grid*, PP(99):1–1, 2017.

[13] Y. Lin and K. Mouratidis. Best upgrade plans for single and multiple source-destination pairs. *GeoInformatica*, 19(2):365–404, 2015.

[14] A. Mahmoody, E. Charalampos, and E. Upfal. Scalable betweenness centrality maximization via sampling. In *KDD*. ACM, 2016.

[15] S. Medya, P. Bogdanov, and A. K. Singh. Towards scalable network delay minimization. In *ICDM*, pages 1083–1088, 2016.

[16] S. Medya, A. Silva, A. K. Singh, P. Basu, and A. Swami. Group centrality maximization via network design. 2018.

[17] A. Meyerson and B. Tagiku. Minimizing average shortest path distances via shortcut edge addition. In *APPROX-RANDOM, I. Dinur, K.Janson, J.Noar and J. D. P. Rolim Eds, Vol. 5687. Springer*, pages 272–285, 2009.

[18] A. E. Motter, S. A. Myers, M. Anghel, and T. Nishikawa. Spontaneous synchrony in power-grid networks. *Nature Physics*, 9(3):191–197, 2013.

[19] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. Best algorithms for approximatingthe maximum of a submodular set function. *Math. Oper. Res.*, pages 177–188, 1978.

[20] D. Paik and S. Sahni. Network upgrading problems. *Networks*, pages 45–58, 1995.

[21] M. Papagelis, F. Bonchi, and A. Gionis. Suggesting ghost edges for a smaller world. In *CIKM*, pages 2305–2308, 2011.

[22] N. Parotisidis, E. Pitoura, and P. Tsaparas. Selecting shortcuts for a smaller world. In *SDM*, pages 28–36. SIAM, 2015.

[23] S. Perumal, P. Basu, and Z. Guan. Minimizing eccentricity in composite networks via constrained edge additions. In *MILCOM*, 2013.

[24] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. CRAWDAD dataset epfl/mobility (v. 2009-02-24). Downloaded from `http://crawdad.org/epfl/mobility/20090224`, Feb. 2009.

[25] M. Riondato and E. M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. In *WSDM*, pages 413–422, 2014.

[26] D. Schrank, T. Lomax, and B. Eisele. 2015 urban mobility scorecard and appendices. *Texas AM Transportation Institute*, 39.

[27] A. Silva, P. Bogdanov, and A. Singh. Hierarchical in-network attribute compression via importance sampling. In *ICDE*, pages 951–962. IEEE, 2015.

[28] M. Waniek, T. P. Michalak, T. Rahwan, and M. Wooldridge. On the construction of covert networks. In *AAMAS*, pages 1341–1349, 2017.

[29] D. P. Williamson and D. B. Shmoys. *The design of approximation algorithms*. Cambridge university press, 2011.

[30] Y. Yoshida. Almost linear-time algorithms for adaptive betweenness centrality using hypergraph sketches. In *KDD*, pages 1416–1425, 2014.

[31] J. Yuan, Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL International conference on advances in geographic information systems*, pages 99–108. ACM, 2010.