Public Transport Planning: When Transit Network Connectivity Meets Commuting Demand

Sheng Wang$^\diamond$, Yuan Sun*, Christopher Musco$^\diamond$, Zhifeng Bao*
$^\diamond$New York University, *RMIT University
[swang,cmusco]@nyu.edu,[yuan.sun,zhifeng.bao]@rmit.edu.au

ABSTRACT

In this paper, we make a first attempt to incorporate both commuting demand and transit network connectivity in bus route planning (CT-Bus), and formulate it as a constrained optimization problem: *planning a new bus route with k edges over an existing transit network without building new bus stops to maximize a linear aggregation of commuting demand and connectivity of transit network.* We prove the NP-hardness of CT-Bus and propose an expansion-based greedy algorithm that iteratively scans potential candidate paths in the network. To boost the efficiency of computing the connectivity of new networks with candidate paths, we propose to use a Lanczos method to estimate the natural connectivity of the transit network with a guaranteed error bound. Moreover, we derive upper bounds on the objective values and use them to greedily select candidates for expansion. Our experiments conducted on real-world transit networks in New York City and Chicago verify the efficiency, effectiveness, and scalability of our algorithms.

ACM Reference Format:
Sheng Wang$^\diamond$, Yuan Sun*, Christopher Musco$^\diamond$, Zhifeng Bao*. 2021. Public Transport Planning: When Transit Network Connectivity Meets Commuting Demand. In *SIGMOD '21: ACM International Conference on Management of Data, June 20–June 25, 2021, Xi'an, Shaanxi, China.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the population density increasing over time [1], the gap between the demand and the supply in public transport system is becoming larger [26, 37]. Upgrading it with new transit routes can reduce this gap [51, 66], hence having the potential to bring people from private cars and taxis to public transport [19, 40, 61, 64]. To achieve this mission, much attention has been paid to planning new routes based on emerging demand discovered from commuting records, which is also known as *demand-aware route planning* [40, 54, 57, 58]. Unfortunately, most of these studies require constructing new bus stops. For well-covered cities like New York City, however, constructing new bus stops is often unnecessary and costly.[1] In contrast, it does not incur any cost to create new edges by linking two unconnected existing stops, due to the connectivity of a road network. Existing studies try to meet demands without creating new stops in various ways, such as formulating it as maximal reverse k nearest neighbor trajectory queries [58], or a budgeted optimization problem [40], or optimizing the time schedule of existing bus routes [42].

Apart from meeting passengers' commuting demands alone, we argue that an ideal transit network should also be more connected and convenient for passengers to transfer, i.e., the new bus route should well connect existing routes such that more transfer options can be provided. As reported in [14, 26, 62, 70, 71], network connectivity is an important indicator; our empirical study also shows that a connectivity-aware route planning can help the commuters along the new route avoid up to **4.7** transfers on average in the Bronx of NYC, while a normal demand-aware planning can only avoid around **1.6** transfers (see the bold numbers in Table 6). Unfortunately, there has not been any transit route planning work that aims to optimize the connectivity of a transit network yet.

Motivated by the above observations, we make a first attempt to define the objective to be optimized as a weighted sum of *transit network connectivity* and *commuting demand.* It provides a flexible way to specify configurations that can meet different planning requirements [18, 30, 33]. Consequently, our optimal bus route planning problem CT-Bus can be formulated as: *given a trajectory dataset of users' commuting records and a transit network over the road network in a city, CT-Bus plans a new route with at most k edges (new and existing), to maximize the objective value.* After a careful study of existing connectivity measures and an evaluation over real-life transit networks (in Section 2), we adopt *natural connectivity* [21, 23, 28, 67] to measure the transit network connectivity.

Optimizing the objective of CT-Bus is challenging because it is a combination of two complex constrained optimization problems over graph [21, 59]. A straightforward solution is to generate a large number of candidate paths from the graph and choose the one with the highest objective value. Then, for every candidate, we need to compute the connectivity of the enhanced network. Unfortunately, it is computationally expensive to evaluate the objective function of CT-Bus since the calculation of connectivity requires the computation of eigenvalues of the adjacency matrix [28, 67].

To overcome this challenge, we convert the connectivity computation as a *matrix trace estimation problem*, and employ a Lanczos method [45, 55] to estimate the natural connectivity with a bounded error. Subsequently, we derive two upper bounds on the objective values to greedily construct candidate bus routes. Furthermore, we propose a pre-computation based method that can significantly reduce the running time and meanwhile generate bus routes with competitive objective values. Finally, when evaluating the proposed methods, we not only monitor the convergence efficiency of the

[1] *A NYC bus network redesign* [5] is being conducted, and will eliminate 400 stops and add new routes in the Bronx [2].

SIGMOD '21, June 20–June 25, 2021, Xi'an, Shaanxi, China
© 2021 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

objective values over real-world transit networks, but also propose multiple metrics to measure the transfer convenience of the new transit network.

To summarize, this paper makes the following contributions:

- We propose and formally define a novel route planning problem-CT-Bus, aiming to plan a new route without constructing new bus stops, such that commuting demands are met and meanwhile the transit connectivity is improved (Section 3).
- We prove the NP-hardness of CT-Bus and propose a general algorithm by expanding, ranking, and pruning candidate paths by traversal in the network (Section 4).
- We identify that network connectivity computation is the major efficiency bottleneck of the above algorithm, and propose to convert it into a matrix trace estimation problem and solve it approximately with a Lanczos-based method via several iterations of simple matrix multiplications. As a result, the efficiency is boosted by up to three order of magnitudes (Section 5).
- We employ pre-computations on edges' connectivity increment to devise a fast connectivity estimation method, to further accelerate the calculation of each candidate path (Section 6).
- We conduct experiments on real-world datasets to verify that our algorithm can efficiently generate a new bus route that not only brings high connectivity increment but also helps connect multiple existing routes for more convenient transfers (Section 7).

2 RELATED WORK

Demand-Aware Route Planing. Compared with traditional route planning via conducting passenger surveys or estimating the demand from demographic data, demand-aware route planning aims to effectively discover timely demands from commuters, whose methodologies can be divided into two groups: 1) network (re-)design and 2) specific route optimization or adding a new route. Specifically, Chen et al. [24] detected the dense areas to build candidate stops and a transit network by utilizing overnight taxi trajectories, and chose popular routes to build the night bus routes from scratch. Pinelli et al. [49] redesigned the whole transit network based on mobile phone trajectories from cell towers, by deriving frequent movement patterns and planning new routes in existing stops, without the consideration of transfer and connectivity.

Instead of knocking down existing transit systems, Liu et al. [40] proposed to discover the routes that are not well operated, and optimized them based on popular origin-destination pairs extracted from taxi and bus trip records, by optimizing a route in the existing transit network. Reverse k Nearest Neighbors over Trajectories (RkNNT) [58] is a tool for estimating the demand for a bus route based on trajectory data in the existing transit network, and it is used to plan a route with a maximum capacity between two given stops. Recently, a trajectory clustering method [59] is proposed to find k representative paths (i.e., traffic trend) in the road network rather than the transit network, hence new stops need to be constructed. To summarize, none of them considered whether new edges should be linked to form a route and make the network more connected like CT-Bus.

Transit Network Connectivity. As one of the key metrics in measuring the transfer convenience of a transit network [14, 34, 71], the concept of *connectivity* [38] has been proposed and studied

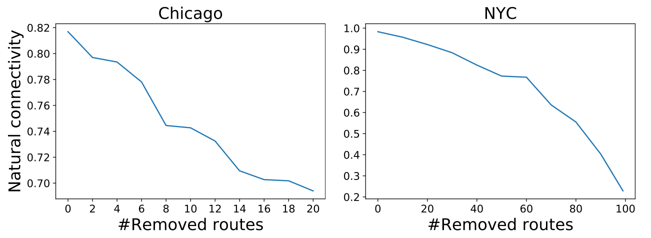


Figure 1: Evaluation of natural connectivity on two real-world datasets.

extensively in the transportation area. By modeling the transport network as an undirected graph composed of vertexes and edges, the connectivity can be measured in numerous ways, such as the vertex and edge connectivity in graph theory [22, 65, 68], and algebraic connectivity [31, 62] and natural connectivity [67] (also known as an extension from *Estrada index* [28]) that are more proper for complex networks. Among all these measures, natural connectivity (see the detailed definition in Equation 1) is arguably the most proper one for transit network, since it will not show drastic changes by small graph alterations (algebraic connectivity) or no change by big graph alteration (edge connectivity); instead, it can monotonically evolve w.r.t. more modifications [21]. To verify its monotonicity in real transit networks, we randomly remove existing routes from Chicago and New York City transit networks gradually, and we observe a nearly linear decrease of natural connectivity (see Figure 1).

To our best knowledge, there has not been any study that can plan a new route to optimize the connectivity of public transport networks yet, despite the choice of connectivity adopted. One loosely related work is Wei et al. [62], aiming to optimize air transport network's connectivity by adding k new discrete edges, which is also a classical graph augmentation problem [21]. However, such an edge is not a route and hence cannot be used to solve our problem; also, it does not consider the demand from commuters. On the other side, in the graph mining field, there have been techniques aiming to improve the connectivity of a network via edge augmentation [21, 23, 47], but the resulted edges are typically discrete and hence cannot be directly applied to plan a connected route.

Comparable Approaches. We call the aforementioned work [21, 62] as *connectivity-first* approaches, and will use it as a baseline in our experiments. As expected, our results (see Figure 6) show that the discrete edges are hard to be connected as a smooth bus route. Similarly, meeting the commuting demand maximally with a single bus route can be another baseline, and we call it as *demand-first* approaches; essentially, it is equivalent to the refinement step in the trajectory clustering [59] and can be implemented with proper parameter setting on our objective function formally defined in Definition 6, which will also be compared.

3 PROBLEM FORMULATION

3.1 Preliminaries

Definition 1. ***(Road Network)*** *A road network is an undirected graph* $G = (V, E)$*:* V *is a set of vertices representing the intersections and terminal points of the road segments;* E *is a set of edges representing road segments. Vertices are indexed from* 1 *to* $|V|$*:* $\{v_1, v_2, \cdots, v_{|V|}\}$.

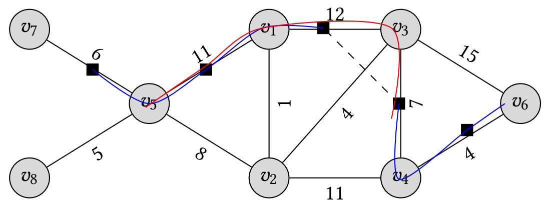


Figure 2: An example of the road network (gray circles represent vertexes, the number denotes edge demand), transit network (black squares indicate stops, two lines with blue color denotes bus routes), and a trajectory (the red line).

Table 1: Summary of major notations.

Symbol	Description
D	the dataset composed of trajectories T
$G_r, G_r^{'}$	the old and new transit networks enhanced by μ
$\boldsymbol{A}, V_r, E_r$	the adjacency matrix, vertex, and edge set of G_r
$\lambda(G_r)$	the connectivity of transit network G_r
μ	the new route composed of edges e in $G_r^{'}$
$O(\mu), O^{\uparrow}(\mu)$	the objective value of μ and its upper bound
$O_d(\mu), O_\lambda(\mu)$	the demand and connectivity increment
τ, $tn(\mu)$	μ's edge length threshold and number of turns
L_d, L_λ	lists of edges e descending ranked by their demand and connectivity increment

Definition 2. ***(Transit Network)*** *A transit network is an undirected graph $G_r = (V_r, E_r)$: V_r is a set of vertices representing bus stops; E_r is a set of edges connecting two vertices. Each vertex in V_r is affiliated with an edge in G. Each edge e corresponds to a path composed of connected edges in G and $|e|$ is the travel length of edge e. A bus route is composed of a set of connected edges in G_r.*

Definition 3. ***(Trajectory Data*** *[59, 60]) A trajectory T in the road network is a set of connected vertices with timestamps (the time entering each vertex) in G, such that $T : (v_1, t_1) \rightarrow (v_2, t_2) \rightarrow \ldots \rightarrow (v_l, t_l)$. Each trajectory can be converted to a path in G and G_r.*

It is worth mentioning that a raw GPS-sampled trajectory composed of points can be projected to the road network effectively via *map-matching* [41] and can have higher analytic precision [60]. We define both networks as undirected since bus routes are usually round trips and can be modeled into one graph. Figure 2 is designed to illustrate all the above definitions. The dashed line shows that a new edge is necessary for transfer between two bus routes. We plot the trajectory and bus routes with a little shift to roads for better visual distinction.

3.2 Problem Definition

3.2.1 Transit Connectivity Measure. Connecting nodes and generating new edges to make the network more connected can offer more transfer choices to passengers. As discussed in Section 2, we choose to use *natural connectivity* [21, 23, 28, 67] (Equation 1).

Definition 4. ***(Transit Network Connectivity)*** *Given a transit network graph G_r, the connectivity of G_r is defined to be the natural connectivity of G_r:*

$$\lambda(G_r) = \ln\left(\frac{1}{n}\sum_{j=1}^{n} e^{\lambda_j}\right) \tag{1}$$

Accordingly, $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ denote a non-increasing ordering of the eigenvalues of G_r's adjacency matrix $\boldsymbol{A}$,[2] *and $n = |V_r|$ is the number of vertexes in G_r.*

3.2.2 Commuting Demand Measure. We choose to extend an edge-based trajectory similarity measure from a trajectory clustering model [59] which can present human mobility well, and the measure is also the state-of-the-art for network-constrained trajectories. We define the *commuting demand* for a bus route with this measure as:

Definition 5. ***(Commuting Demand)*** *Given a set of trajectories $D = \{T_1, T_2, \cdots, T_m\}$ in a road network, and a new bus route μ, we denote the commuting demand that can be met by μ as:*

$$O_d(\mu) = \sum_{T_i \in D} |T_i \cap \mu| \tag{2}$$

where $T_i \cap \mu$ denotes the common edges that T_i and μ share.

3.2.3 Planning New Route. With the definitions of transit connectivity and commuting demand measures, the objective of CT-Bus is to find a new path to optimize both, which is formally defined as:

Definition 6. ***(CT-Bus)*** *Given a set of trajectories $D = \{T_1, T_2, \cdots, T_m\}$ in a transit network G_r, CT-Bus aims to find a path μ as a new bus route with at most k edges, to maximize the following weighted objective value:*

$$O = \underset{\mu \in G_r^{'}}{\arg\max} \left(w \cdot \frac{O_d(\mu)}{d_{max}} + (1 - w) \cdot \frac{O_\lambda(\mu)}{\lambda_{max}} \right) \tag{3}$$

where μ should be a circle-free path both in $G_r^{'}$ and G,[3] *and $G_r^{'} = \{V_r, E_r^{'}\}$ is the new transit network enriched by new edges in μ, and $O_\lambda(\mu) = \lambda(G_r^{'}) - \lambda(G_r)$ is the connectivity increment with μ.*

In real bus route planning, two bus stops should not be too far [6] and turn-around should not be frequent [13]. Based on the statistics on NYC, two neighbor stops usually have a threshold τ on the straight line distance and number of turns $tn(\mu)$ [58] (see Figure 17 of [58]). We select a fixed constant $\tau = 0.5km$ for CT-Bus in this paper, and we also set a threshold Tn on the number of turns of μ, i.e., $tn(\mu) \leq Tn$.

Combining multiple objectives into a single weighted objective value is the most straightforward way and has been widely applied in transit route planning [17, 30, 52, 63].[4] Here, two constants λ_{max} and d_{max} are used to normalize two dimensions into the same scale (choices will be discussed in Equation 13 for experimental setting). There are two parameters that are independent of the data and can be set by the users: 1) $w \in [0, 1]$ is a constant value to balance two objectives, and we set $w = 0.5$ by default; 2) k is the number of edges.

[2] After exponentiation and re-normalization, natural connectivity can be seen as an extension of the *Estrada index* [28] ($EE = \sum_{j=1}^{n} e^{\lambda_j}$) which is also widely used in chemistry for measuring the structure of protein.

[3] Here, we mean that all stops except the departure station should be crossed by the new planned path only once, so one-way loop is allowed.

[4] We adopt a linear combination to weigh these two objectives, where a configurable parameter w can meet various planning requirements [18, 33]. The capacity of bus is not considered here as it is usually considered in a subsequent task after the planned route is determined, i.e. optimal timetable scheduling [43].

4 OUR METHODS

4.1 NP-Hardness of CT-Bus

Lemma 1. ***CT-Bus is NP-hard.***

Proof. Considering the extreme case where the transit network G_r is a fully connected graph (complete graph), the natural connectivity of G_r will not change when we add a new bus route into the network. Hence, the objective of CT-Bus is degenerated to the case where only the commuting demand is considered in bus route planning. To simplify the calculation, we convert the demand objective function (Equation 2) into a function of f_e, where f_e denotes the number of trajectories that include edge e. We then have:

$$\begin{aligned} O_d(\mu) &= \sum_{e \in \mu} \sum_{T_i \in D} b \cdot |e|, \ where\ b = \begin{cases} 1, & if\ e \in T_i \\ 0, & otherwise \end{cases} \\ &= \sum_{e \in \mu} f_e \cdot |e| \end{aligned} \tag{4}$$

After such a conversion, we can reduce CT-Bus to a routing problem which aims to maximize $\sum_{e \in \mu} f_e \cdot |e|$ when constructing a route μ with k edges. This is equivalent to the *k minimum travel salesman problem (k-TSP)*, that has been proved to be NP-hard [15, 32, 59], where renormalization can be conducted to convert minimum to maximum. More generally, when both commuting demand and network connectivity have to be considered, the problem is at least as hard as the k-TSP problem, and thus is also NP-hard. □

Solving the CT-Bus problem is challenging even in the extreme case where only the commuting demand needs to be considered (Equation 4). An approximation algorithm with bounded error can be achieved only if the edge weight satisfies the triangle inequality [32]. However, in our case, the edge weight (i.e., $f_e \cdot |e|$) does not satisfy the triangle inequality, making it impossible to derive an error bound for an approximation algorithm. Hence, we propose an expansion-based greedy algorithm to solve CT-Bus.

4.2 Expansion-based Traversal Algorithm

Main Idea. We employ a classical expansion-based graph traversal method [35, 59] to solve the CT-Bus problem, with new accelerating optimizations proposed, as shown in Algorithm 1. We first select edges with high demand in the graph for expansion in the *initialization* stage. In the *expansion* phase, we scan candidate paths by initializing seeding candidates with all edges (including existing edges in G_r and potential edges with a length less than τ) first, and then incrementally add neighbor edges to the ends as new candidates. In the *verification* phase, we compute the connectivity of each candidate, and update the result if the candidate has a higher objective value, after passing the *feasibility* and *domination* checking. The above two phases will be conducted iteratively until the termination criterion is satisfied, i.e., the number of iterations exceeds a predefined threshold or there is no more candidate in the queue to be processed (Line 5).

4.2.1 Initialization. In the Function `Initialization` (details in Line 19 to 27), we select all the potential edges with a distance within τ as the seeding paths for expansion, and insert them to a priority queue Q for further expansion. To generate the candidate edges, we find all the neighboring stops that are within a distance τ of a stop and add the pair into the candidate edge list L_d. The edges in L_d are sorted in descending order based on their demand (i.e., $f_e \cdot |e|$), and we use $L_d(i)$ to denote the demand of the i-th edge in L_d. To compute the demand of each new edge, we conduct a shortest path search to connect its two bus stops and aggregate the demands of all the road network edges it crosses.

Algorithm 1: ETA(G, G_r, L_d)

```
Output: $\mu$: new path
1  $Q \leftarrow \emptyset, DT \leftarrow \emptyset, O_{max} \leftarrow 0, it \leftarrow 0$;
   /* Initilize candidate edges */
2  Initialization($G, G_r, Q, \tau$);
3  while $Q \neq \emptyset$ do
       /* Scan every candidate path $cp$ from $Q$ */
4      $(O^{\uparrow}(cp), cp, O(cp), tn(cp), cur) \leftarrow Q.poll()$;
5      if $O^{\uparrow}(cp) \leq O_{max}$ or $it \geq it_{\max}$ then
6          break;
       /* Expand candidate $cp$ with best neighbors */
7      $it \leftarrow it + 1, max_c \leftarrow 0$;
8      for each neighbor edge $e \in L_d$ of $cp$'s two ends do
9          if $e \notin cp$ then
10             $p \leftarrow cp + e$, compute $O(p)$ by Lanczos method;
11             if $O(p) > max_c$ then
12                 $max_c \leftarrow O(p)$, update $be$ ($ee$) with $e$;
       /* Update the best path $\mu$ with new $cp$ */
13     $cp \leftarrow be + cp + ee$, compute $O(cp)$ by Lanczos method;
14     if $O(cp) > O_{max}$ then
15         $O_{max} \leftarrow O(cp), \mu \leftarrow cp$;
       /* Insert $cp$ into $Q$ for further expansion */
16     FurtherExpansion($cp, O(cp), Q$);
17 return $\mu$;
18 Function Initialization($G, G_r, Q, \tau$):
19     $L_d \leftarrow$ CandidateEdges($G_r, \tau, G$);
20     for each edge $e_i$ in $L_d$ do
21         Update $\mu$ and $O_{max}$ by $e_i$ and $O(e_i)$ ;
22         $cur \leftarrow k, O^{\uparrow}_d(e_i) \leftarrow \sum_{i=1}^{k} L_d(i)$;
23         if $i > k$ then
24             $cur \leftarrow k - 1$;
25             $O^{\uparrow}_d(e_i) \leftarrow O^{\uparrow}_d(e_i) - (L_d(k) - L_d[e_i])$;
26         $O^{\uparrow}(e_i) \leftarrow w \cdot \frac{O^{\uparrow}_d(e_i)}{d_{max}} + (1-w) \cdot \frac{O^{\uparrow}_\lambda(e_i)}{\lambda_{max}}$;
27         $Q.push(O^{\uparrow}(e_i), e_i, O(e_i), 0, cur)$;
28 Function FurtherExpansion($cp, O(cp), Q$):
29     if $tn(cp) < \mathrm{Tn}$ & $O^{\uparrow}(cp) > O_{max}$ & $len(cp) < k$ then
30         Update $(O^{\uparrow}_d(cp), tn(cp), cur)$ by Algorithm 2;
31         $O^{\uparrow}(cp) \leftarrow w \cdot \frac{O^{\uparrow}_d(cp)}{d_{max}} + (1-w) \cdot \frac{O^{\uparrow}_\lambda(cp)}{\lambda_{max}}$;
32         if $O(cp) > DT(cp.be, cp.ee)$ then
33             $DT(cp.be, cp.ee) \leftarrow O(cp)$;
34             $Q.push(O^{\uparrow}(cp), cp, O(cp), tn(cp), cur)$;
```

4.2.2 Expansion from Two Ends. In Line 8, the expansion is conducted by adding new edges, where we have two options:

1) All Neighbors. Enqueueing the candidate path appended with each new neighbor edge enables us to scan potential candidates fully. Both depth-first and breadth-first can be used to fully scan all the candidates, and here we employ a breadth-first search based on a priority queue. Note that appending with each new neighbor will result in a large queue and thereby the algorithm is hard to terminate. We call the method with this option as **ETA-AN**, which will be compared in experiments.

2) Best Neighbor. To solve the convergence problem, we propose to choose only the best neighbor edge as shown in Algorithm 1, i.e., after selecting the optimal neighbors be and ee with the highest increment, we let $cp \leftarrow be + cp + ee$ (Line 13). The queue size will then remain a proper size without growing, and even become smaller after *feasibility checking*.

Before inserting candidate cp into Q for further expansion in the function at Line 16 (details in Line 29 to 34), we compute the objective and update the optimal path μ in Line 13, then further estimate its upper bound $O^{\uparrow}(cp)$ with the current best result's score O_{max}. If $O^{\uparrow}(cp) > O_{max}$, this candidate is inserted; otherwise, it is discarded. To distinguish all candidates in the queue, each will be attached with an upper bound on its objective value (Line 34).

4.2.3 Verification. From Line 13, we compute the objective value of each candidate path cp and check whether it can replace the current optimal result μ. Before inserting cp into the queue in Line 34 for future expansion, we conduct the following checking:

Feasibility Checking. Circle-free is a basic criterion in planning bus routes, and every edge can be crossed once in a bus route. Turn-checking will check how many turns the candidate path already has and will discard it if the number exceeds a threshold Tn.

Domination Checking. In Line 33, all candidates will go to a domination table DT composed of the checked paths, to compare the objective value and conduct the domination checking. This step can avoid repetitive expansion on the path that shares the same beginning edge be and ending edge ee but with a smaller objective value. We call the method without this optimization as **ETA-DT**.

4.3 Bottlenecks to Make ETA Work Efficiently

The above expansion-based algorithm applies a common methodology in solving route planning problems [35]. However, there are two main bottlenecks to get the CT-Bus answered efficiently.

Bottleneck 1. There will be intensive connectivity computations (Line 10 and 13), where a single operation will cost minutes (see Column 2 of Table 2), and CT-Bus always needs thousands of iterations to terminate according to our experiments.

Bottleneck 2. To differentiate candidates in Q, we estimate their *upper bounds* in line 26 and 31 to predict the best case, and choose the one with the highest upper bound to conduct expansion. The bottleneck here is how can we get tight upper bounds $O^{\uparrow}$ efficiently.

To overcome Bottleneck 1, we convert the connectivity computation to fast trace computation using the *Lanczos method* [45, 55], combined with Hutchinson's stochastic trace estimator [36]. We then derive tight upper bounds on the objective values based on the estimated connectivity in Section 5. To overcome Bottleneck 2, we pre-compute connectivity increment for every edge in Section 6, that can further improve the performance of **ETA** dramatically.

5 FAST CONNECTIVITY AND BOUND ESTIMATION

In this section, we first show how to efficiently calculate the natural connectivity of a transit network by estimating the adjacency matrix's trace (Equation 5), and prove that its approximation error can be bounded within 1% (Lemma 2). Subsequently, we derive two upper bounds on the connectivity of a new network enhanced with a path μ (Lemma 3 and 4). Finally, we introduce a fast way to incrementally update the upper bound based on the previous bound of a path, without recalculating it from scratch (Algorithm 2).

5.1 Lanczos-based Connectivity Estimation

Given an updated graph G_r, estimating its connectivity $\lambda(G_r)$ efficiently and precisely is crucial to answering CT-Bus. We solve this problem by converting it to a *matrix trace estimation problem*, which can be rapidly approximated by combining *the Lanczos method* [27, 45, 55, 56] with *Hutchinson's stochastic trace estimator* [16, 36]. These techniques are often used together in the applied mathematics literature and turn out to be a winning combination for accurately estimating natural connectivity in transit networks in this paper. Specifically, we start with the observation that:

$$\lambda(G_r) = \ln(\frac{1}{n}\sum_{j=1}^{n} e^{\lambda_j}) = \ln(\frac{1}{n}\operatorname{tr}(e^{\boldsymbol{A}})) \tag{5}$$

where tr is the matrix trace and $e^{\boldsymbol{A}} \in \mathbb{R}^{n\times n}$ is the standard matrix exponential of the adjacency matrix $\boldsymbol{A}$. So our task reduces to approximating $\operatorname{tr}(e^{\boldsymbol{A}})$. One approach to do so, which was taken in prior work [21, 23], is to compute only the largest eigenvalues of $\boldsymbol{A}$ using an iterative Krlyov subspace method (like the Lanczos method) and to estimate Equation 5 using a truncated sum. However, for transit networks, which are typically planar or nearly planar, the eigenvalues of $\boldsymbol{A}$ decay very slowly, so many eigenvalues are needed to accurately approximate $\lambda(G_r)$.

Fortunately, the Lanczos method can be used in a far more economical way. Hutchinson [36] made the powerful observation that, for any $\boldsymbol{M} \in \mathbb{R}^{n\times n}$,

$$\mathbb{E}(\boldsymbol{v}^T \operatorname{tr}(\boldsymbol{M})\boldsymbol{v}) = \operatorname{tr}(\boldsymbol{M}) \tag{6}$$

when $\boldsymbol{v}$ is a vector with unit variance random Gaussian entries. Accordingly, if we draw s random Gaussian vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_s$, we can estimate $\operatorname{tr}(\boldsymbol{M})$ by:

$$\gamma = \frac{1}{s}\sum_{i=1}^{s} \boldsymbol{v}_i^T \operatorname{tr}(\boldsymbol{M})\boldsymbol{v}_i \tag{7}$$

It is possible to prove that, if $\boldsymbol{M}$ is positive semi-definite and $s = O(\log(1/\delta)/\epsilon^2)$, then with probability $(1-\delta)$, γ is within a multiplicative $(1 \pm \epsilon)$ of $\operatorname{tr}(\boldsymbol{M})$ [50]. Since $e^{\boldsymbol{A}}$ is always positive semi-definite, this bound immediately applies to our problem.

What's more, the required computation of $\boldsymbol{v}^T \operatorname{tr}(e^{\boldsymbol{A}})\boldsymbol{v}$ can be accelerated by *iteratively* approximating $\operatorname{tr}(e^{\boldsymbol{A}})\boldsymbol{v}$ using the Lanczos method for the matrix exponential. Each iteration of this method requires a matrix vector multiply with $\boldsymbol{A}$, which takes just $O(m)$

Table 2: Running time of connectivity & bound estimation.

City	Eigen NumPy	Lanczos NumPy	Lanczos Matlab	General bound	Path bound
Chicago	28.65s	0.610s	0.035s	0.102s	0.049s
NYC	225.03s	2.412s	0.094s	0.204s	0.099s

Table 3: Tightness comparison of connectivity upper bound.

City	Estrada bound [25]	General bound	Bound path	Increment bound
Chicago	104.205	1.576	0.167	0.034
NYC	156.459	0.655	0.067	0.010

time, where m is the number of edges in G_r. To bound the number of iterations of Lanczos needed for an accurate approximation, we state a corollary[5] of Theorem 15 in Musco et al. [45]:

Lemma 2 (**Lanczos Approximation Bound**). *After $t = O(\|A\|_2 + \log(1/\epsilon))$ iterations, the Lanczos method returns an approximation s to $e^{A}v$ satisfying:*

$$\|s - e^{A}v\|_2 \le \epsilon\,\mathrm{tr}(e^{A})\|v\|_2$$

Above $\|A\|_2$ denotes the spectral norm of A. Even when A is large, this is typically very small for transit networks (and planar graph adjacencies more generally). For example, for the Chicago and NYC transit networks analyzed in this paper, $\|A\|_2$ equals 5.46 and 4.79, respectively. Accordingly the number of iterations required to accurately approximate $e^{A}v$ essentially depends just logarithmically on the desired accuracy ϵ.

Combined with the stated bound on the number of samples needed for Hutchinson's estimator, and using that for a scaled Gaussian random vector $\|v\|_2 = O(\sqrt{n})$ with high probability, we conclude that $\mathrm{tr}(e^{A})$ can be estimated to multiplicative $(1 \pm \epsilon)$ error with $O(\log(1/\delta)/\epsilon^2)$ approximate computations of $v^T e^{A} v$, each of which takes just $O(\|A\|_2 + \log(n/\epsilon))$ iterations. This translates to an additive $\pm\epsilon \ln(\mathrm{tr}(e^{A}))$ approximation to $\lambda(G_r)$.

Experimentally, we confirm the low complexity of the Lanczos + Hutchinsons method. In this paper, we use a default setting of $s = 50$ repetitions of Hutchinson's estimator, each computed using $t = 10$ iterations. We typically obtain an approximation to $\lambda(G_r)$ accurate to with **1% error**. Table 2 shows the time comparing with eigenvalues-based full computation.

5.2 Connectivity Upper Bound Estimation

De La Peña et al. [25] estimated bounds for the Estrada index, which can be converted to the maximum natural connectivity with k arbitrary edges:

$$\lambda(G'_r) \le \ln(1 + \frac{e^{\sqrt{2(|E_r|+k)}} - 1}{|V_r|})$$

However, we found that this bound is much bigger than the real connectivity, and it is too loose to be used as a normalization value (see a comparison in Table 3 where $k = 15$).

We further propose a tighter upper bound on the connectivity after adding k edges, which will depend on the connectivity of the original graph, and top-k eigenvalues of the original graph adjacency matrix A. This can be computed quickly using a Lanczos method that we have mentioned. Further, one of the advantages of the expression is that it gets much tighter if the k edges added in CT-Bus form a path. This bound will also help estimate the upper bound of a candidate to be filled with less than k edges, and solve existing or new network optimization problems [21, 23] in future.

Lemma 3 (**General Upper Bound**). *If G'_r is obtained by adding k arbitrary unweighted edges to G_r, the natural connectivity satisfies:*

$$\lambda(G'_r) \le \ln\left(e^{\lambda(G_r)} - \sum_{i=1}^{2k} e^{\lambda_i} + \frac{e^{\lambda_1}}{n}\left[e^{\sqrt{2k}} + 2k - 1\right]\right)$$

where $\lambda_1 \ge \ldots, \lambda_{2k}$ are the $2k$ algebraically largest eigenvalues of G_r's adjacency matrix.

Proof. Let A be the adjacency matrix of our original transit network G_r and let A' be the adjacency matrix of the updated network G'_r, which is obtained by adding k edges. Let $K = A' - A$. Let $\lambda_1 \ge \ldots \ge \lambda_n$, $\lambda'_1 \ge \ldots \ge \lambda'_n$, and $\sigma_1 \ge \ldots \ge \sigma_n$ be the eigenvalues of A, A', and K, respectively.

Let $\mathrm{tr}(\cdot)$ denote the matrix trace, we have $\lambda(G'_r) = \ln\left(\frac{1}{n}\,\mathrm{tr}(e^{A'})\right)$. Then it suffices to upper bound $\mathrm{tr}(e^{A'})$. To do so, we apply the Golden–Thompson inequality: $\mathrm{tr}(e^{A'}) = \mathrm{tr}(e^{A+K}) \le \mathrm{tr}(e^{A}e^{K})$. Next, we apply a trace inequality of Lassere [39] to bound $\mathrm{tr}(e^{A}e^{K}) \le \sum_{i=1}^{n} e^{\lambda_i}e^{\sigma_i}$. Since K is a graph adjacency matrix with at most $2k$ nodes, it has a rank of at most $2k$, so we have $\sum_{i=1}^{n} e^{\lambda_i}e^{\sigma_i} \le \sum_{i=2k+1}^{n} e^{\lambda_i} + e^{\lambda_1}\sum_{i=1}^{2k} e^{\sigma_i}$. Since this expression is maximized exactly when K is chosen to maximize the Estrada index $\sum_{i=1}^{2k} e^{\sigma_i}$, we can apply the upper bound of De La Peña et al. [25] to obtain $\sum_{i=1}^{2k} e^{\sigma_i} \le 2k - 1 + e^{\sqrt{2k}}$.

Our final result is that $\mathrm{tr}(e^{A'}) \le \mathrm{tr}(e^{A}) - \sum_{i=1}^{2k} e^{\lambda_i} + e^{\lambda_1}[2k - 1 + e^{\sqrt{2k}}]$, which gives the bound of the lemma after renormalizing and taking a log. □

To obtain a tighter upper bound when edges are specifically added into a path, we rely on Fan's powerful generalization of Weyl's inequality [20, 29]. This bound tells us that, if $\lambda_1 \ge \ldots \ge \lambda_n$, $\lambda'_1 \ge \ldots \ge \lambda'_n$, and $\sigma_1 \ge \ldots \ge \sigma_n$ are any non-increasing ordering of the eigenvalues of A, A', and K, respectively, then:

$$\text{For all } q \in 1, \ldots, n \qquad \sum_{i=1}^{q} \lambda'_i \le \sum_{i=1}^{q} \lambda_i + \sum_{i=1}^{q} \sigma_i \tag{8}$$

Lemma 4 (**Upper Bound for Paths**). *If G'_r is obtained by adding a k edge simple path to G_r, then the natural connectivity satisfies:*

$$\lambda(G'_r) \le \ln\left(e^{\lambda(G_r)} + \frac{1}{n}\sum_{i=1}^{\lfloor\frac{k+1}{2}\rfloor} (e^{\sigma_i} - 1)e^{\lambda_i}\right)$$

where $\sigma_i = 2\cos\left(\frac{i\pi}{k+2}\right)$ is the i^{th} eigenvalue of the path graph adjacency matrix.

[5] This corollary follows from simple algebraic manipulation of Theorem 15 in [45] (see also [48]), combined with the fact that $\mathrm{tr}(e^{A}) \ge e^{\|A\|_2}$.

Proof. For $i = 1, \ldots, n$, let $\Delta_i = \lambda'_i - \lambda_i$. We have that:

$$e^{\lambda(G'_r)} = e^{\lambda(G_r)} + \frac{1}{n}\sum_{i=1}^{n}(e^{\Delta_i} - 1)e^{\lambda_i}. \tag{9}$$

We are going to choose $\Delta^*_1, \ldots, \Delta^*_n$ to maximize this expression given the constraints of implied by Equation 8:

$$\text{For all } q = 1, \ldots, n, \qquad \sum_{i=1}^{q}\Delta_i \leq \sum_{i=1}^{q}\sigma_i.$$

It is clear that any solution which maximizes Equation 9 under these constraints must set $\Delta^*_q = \sum_{i=1}^{q}\sigma_i - \sum_{i=1}^{q-1}\sigma_i = \sigma_q$.

The non-zero eigenvalues of $\boldsymbol{K} = \boldsymbol{A'} - \boldsymbol{A}$ are simply the well-known eigenvalues of an unweighted simple path graph, which are equal to $2\cos\left(\frac{i\pi}{k+2}\right)$ for $i = 1, \ldots, k+1$. The remaining $n-k-1$ eigenvalues of $\boldsymbol{K}$ are equal to 0. Noting that only the first $m = \lfloor \frac{k+1}{2} \rfloor$ path graph eigenvalues are positive, we immediately have that $\Delta^*_1, \ldots \Delta^*_m = 2\cos\left(\frac{1\pi}{k+2}\right), \ldots, 2\cos\left(\frac{m\pi}{k+2}\right)$, $\Delta^*_{m+1}, \ldots, \Delta^*_{n-k+m} = 0$, and $\Delta^*_{n-k+m+1}, \ldots, \Delta^*_n = 2\cos\left(\frac{(m+1)\pi}{k+2}\right), \ldots, 2\cos\left(\frac{(k+1)\pi}{k+2}\right)$.

The lemma follows by noting that $(e^{\Delta}_i - 1) \leq 1$ for all $\Delta_i \leq 0$. □

Note that computing the bound of Lemma 4 requires computing the top $O(k)$ eigenvalues of $\boldsymbol{A}$, which can be done to high accuracy in roughly $O(|E| \cdot k \cdot \log n)$ time for a graph with $|E|$ edges [44]. In Algorithm 1, $O^{\uparrow}_{\lambda}(cp)$ and $O^{\uparrow}_{\lambda}(e_i)$ are both computed based on Lemma 4.

5.3 Incremental Demand Bound Estimation

We set the initial demand upper bound as the sum of the top-k edge demands in L_d:

$$O_d(\mu) \leq \sum_{i=1}^{k} L_d(i)$$

When a new edge is added, the path is updated, and we need to re-estimate the demand upper bound. A baseline is to re-scan the whole path and enrich the rest uncovered edges by $k - len(cp)$ top edges in L_d, which should not be in cp, where $len(cp)$ denotes number of edges in cp, and $E_r(i)$ denotes i-th edge in L_d.

$$O^{\uparrow}_d(cp) = \sum_{e \in cp} L_d[e] + \sum_{i=1 \ \& \ E_r(i) \notin cp}^{k-len(cp)} L_d(i) \tag{10}$$

However, this is not efficient when it needs to be conducted for every candidate. Instead of scanning, we propose a dynamic strategy to update $O^{\uparrow}_d(cp)$ while returning the same bound. When a new edge is added, if its weight $L_d[e]$ is smaller than the cur-th top edge's demand $L_d(cur)$, it means we can replace one top edge with the inserted one, then we update $O^{\uparrow}_d(cp)$ by reducing the gap $L_d(cur) - L_d[e]$, the cursor value cur will decrease by one; otherwise, $O^{\uparrow}_d(cp)$ and cur will not change. Each cursor cur is initialized as k at the beginning and will be inherited in the iteration with the upper bound. Based on this incremental method, we can dynamically update the bound, without scanning the whole path cp and ranking list L_d, which will be more efficient and space-saving. Algorithm 2 presents the details.

Algorithm 2: Incremental update on bound & turn

```
Input: $e$, $O^{\uparrow}_d(cp)$, $tn(cp)$, $cur$
1 if $L_d(cur) > L_d[e]$ then
2     $cur \leftarrow cur - 1$;
3     $O^{\uparrow}_d(cp) \leftarrow O^{\uparrow}_d(cp) - (L_d(cur) - L_d[e])$;
4 $angle \leftarrow$ ComputeAngle($e, cp.end$);
5 if $angle > \frac{\pi}{4}$ then
6     $tn(cp) \leftarrow tn(cp) + 1$;
7 if $angle > \frac{\pi}{2}$ then
8     $tn(cp) \leftarrow Tn$;
9 return $(O^{\uparrow}_d(cp), tn(cp), cur)$;
```

6 PRE-COMPUTATION FOR FASTER ETA

Although employing the Lanczos method can boost the efficiency of estimating connectivity and upper bounds in Algorithm 1, a single iteration's runtime is still nonnegligible, as shown in Table 2. Hence with thousands of iterations, it is slow to terminate this algorithm (see our experiment results in Column 2 & 4 of Table 7). Thus, in this section, by reducing the connectivity to a linear aggregation function for fast updates with new edges, pre-computation is conducted on each edge's demand and connectivity, which can accelerate the algorithm drastically by using a greedy strategy.

6.1 Linear Connectivity and Bound Increment

Recall Figure 1 the connectivity of a network keeps decreasing as edges are removed. It inspires us to explore the possibility of having a linear increase in connectivity by adding edges. To facilitate our exploration, we define a new concept as below.

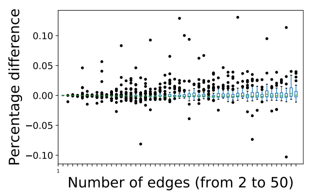
Definition 7. ***(Edge Connectivity Increment)*** *By adding an edge e into the transit network G_r, which generates a new graph $G^{'}_r$, the edge connectivity increment is denoted as:*

$$\Delta(e) = \lambda(G^{'}_r) - \lambda(G_r) \tag{11}$$

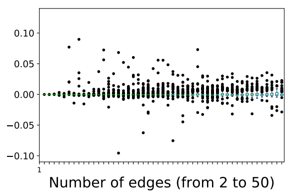
By adding a new route μ with multiple edges, the connectivity increment is denoted as $O_\lambda(\mu)$ in Definition 6. Discovering the relation between the sum of μ's each individual edge $\sum_{e\in\mu}\Delta(e)$ and $O_\lambda(\mu)$ is a fundamental problem in *set selection problem*, which is known as *submodularity* [46]. However, we have the observation: *Natural connectivity is monotonically increasing when adding new edges, but a non-submodular function.* The analysis is as below.

Along with the definition, Wu et al. [67] have also proved its monotonicity. To prove it is sub-modular, with the above definition, we just need to verify whether $O_\lambda(\mu) < \sum_{e\in\mu}\Delta(e)$ holds. We use counterexamples to verify that natural connectivity is non-submodular. As shown in Figure 3, we randomly sample a set of new edges and plot the percentage difference of two connectivity scores: $\theta = \frac{O_\lambda(\mu) - \sum_{e\in\mu}\Delta(e)}{\sum_{e\in\mu}\Delta(e)}$, by increasing the number of selected edges in NYC and Chicago. The box plot shows that $O_\lambda(\mu) > \sum_{e\in\mu}\Delta(e)$ holds in most time especially when more edges are included in μ, so natural connectivity is non sub-modular, and there is no guaranteed bound [46] for a greedy algorithm when answering CT-Bus.

Even though natural connectivity is non sub-modular, we still observe that $O_\lambda(\mu)$ is highly close to $\sum_{e\in\mu}\Delta(e)$. Then, we can use



(a) Chicago



(b) NYC

Figure 3: Distribution of percentage difference θ between $O_\lambda(\mu)$ and $\sum_{e\in\mu}\Delta(e)$ with the increasing number of edges.

$\sum_{e\in\mu}\Delta(e)$ to estimate the potential connectivity increment to fulfill all the edges in μ, and have: $O_\lambda(\mu) \approx \sum_{e\in\mu}\Delta(e)$.

6.2 Improvements to Algorithms 1 and 2

With the fast Lanczos method, we are able to pre-compute $\Delta(e)$ for all the candidate edges in L_d. Then, we rank them by their connectivity increment $\Delta(e)$ as another descending sorted list L_λ, i.e., $L_\lambda[e] = \Delta(e)$, and $L_\lambda(i)$ returns the i-th edge's demand. Further, the increment upper bound can be estimated using $O_\lambda^{\uparrow}(\mu) = \sum_{i=1}^{k} L_\lambda(i)$, similar to $O_d^{\uparrow}$, and its tightness can be observed from the last column of Table 3. Now, O_d and O_λ can be incrementally computed in the same way, and we further combine them into one.

Integrated Objective Value Increment. We compute a new objective value composed of the normalized connectivity and demand on each edge, same in Definition 6, including each existing edge (the connectivity increment is set as 0) and each new edge. Then we rank all the edges by this objective and create a new sorted list L_e in descending order, and convert CT-Bus to optimize a single objective instead of two. With an edge e being added into cp, $O(cp)$ will increase by:

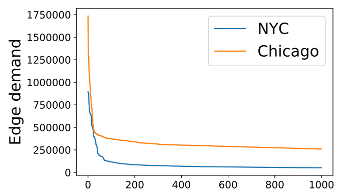
$$L_e[e] = w \cdot \frac{L_d[e]}{d_{max}} + (1-w)\cdot\frac{L_\lambda[e]}{\lambda_{max}} \tag{12}$$

Next we show how to optimize the algorithms earlier presented in Section 4. First, we replace all the L_d with L_e, and $O^{\uparrow}$ with $O_d^{\uparrow}$ in Algorithm 1 and 2, respectively. Then, we revise Line 21 as: *Update μ by neighbor e_i with the highest $L_e(e_i)$*. Further, we remove Line 26 and 31 of Algorithm 1. Lastly, we can simplify line 13 as: *Increasing $O(cp)$ by $L_e[e]$*, correspondingly.

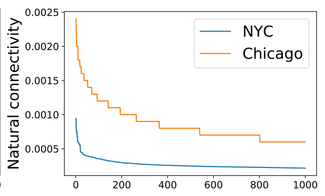
Selective Edges for Seeding. After our increment computation on edges in L_d using the Lanczos method, we observe that a minority of edges can lead to a large increment on connectivity and demand of the objective functions, as shown in Figure 4. To reduce the candidate pool size, we choose top-sn edges in the list L_e as initial seeding paths in Line 19, where sn denotes the seeding number.

6.3 More Discussions

Effect of $|D|$ and Pre-processing. Firstly, our method is independent of the number of trajectories $|D|$, since all the trajectories are mapped to the road network and each edge on the road network gets a demand weight. The pre-processing on mapping and building transit network edge weight is related to the number of new edges, as each edge will call one-time shortest path search, and connectivity increment estimation using the Lanczos method.



(a) Demand increment



(b) Connectivity increment

Figure 4: Top-1000 new edges in terms of increment.

Effect of Increasing τ. According to the above analysis, the complexity of CT-Bus is highly related to the number of candidate edges. A direct parameter to this is the neighbor stop interval distance τ in Definition 6; if we increase τ, there will be more candidate edges. In this paper, we set τ as a fixed constant (τ=0.5km), which is big enough w.r.t. the current statistics in NYC. However, the number of edges will not sharply increase according to our initial numerical analysis, the running time of pre-computation and ETA will increase linearly and slightly when increasing τ in a proper range.

Planning Multiple Routes. It is worth mentioning that CT-Bus can be employed to plan multiple routes as follows – After planning a new route, we update the graph and its adjacent matrix with the new edges. Then, we can set all the covered edges' demand value as zero, as our previously-planned new bus routes have covered them. This step can also be skipped if there is no requirement on whether a new edge should be crossed only once. At the end, we conduct the new route searching using our algorithm.

7 EXPERIMENTS

7.1 Setup

7.1.1 Datasets. We conduct experiments in two cities, New York City (NYC) and Chicago (Chi), where the road network with travel distance and travel time on each edge is obtained from DIMACS [4], and the transit network is extracted from shapefiles [3, 8]. Figure 5 presents an overview of these four networks.

Trajectories are obtained from real-world taxi trip records ([7] for Chicago and [12] for NYC) as below: each trip record consists of a pickup and a drop-off location, travel time and travel distance; for each trip we find its shortest path, and if it has a similar travel distance and time (within 5% error rate) with this trip, we treat it as an approximation of the trip's real trajectory. The above way is also used in trajectory-driven site selection problem [69]. Detailed dataset statistics can be found in Table 5.

7.1.2 Implementation. We use Python 3 to clean data and search the shortest path between two stops, and MATLAB to compute the connectivity, which is much faster for large matrix computation. NetworkX [10] is used to store the graph and the shortest path, and Mapv [9] is used to visualize the networks and the returned results in maps. All experiments are conducted on a laptop with 2.6 GHz 6-Core Intel Core i7 and 32 GB 2400 MHz DDR4 running MacOS Catalina. Our cleaned datasets, MATLAB code and visualization tools are available on GitHub for **reproducibility** [11].

7.1.3 Pre-processing. Table 4 shows the number of new edges of our two datasets and their pre-processing time. Pre-processing

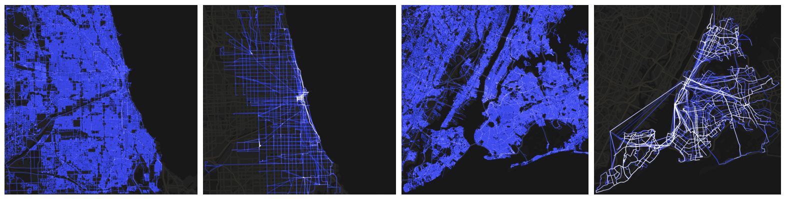

(a) Chicago-Road (b) Chicago-Transit (c) NYC-Road (d) NYC-Transit

Figure 5: An overview of road network and bus network.

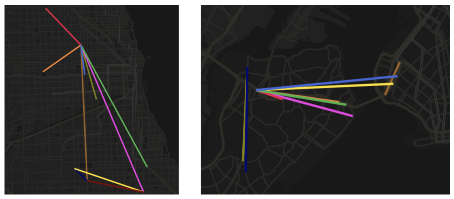

(a) Chicago (b) NYC

Figure 6: Top-10 edges of connectivity-first method [21].

Table 4: Running time of pre-computation on new edges.

Dataset	#New edges	Connectivity	Shortest path
Chicago	95,304	1857s	15,322s
NYC	160,790	7332s	33,241s

Table 5: An overview of datasets.

Dataset	$\|R\|$	$len(R)$	$\|V\|$	$\|V_r\|$	$\|E\|$	$\|E_r\|$	$\|D\|$
Chicago	146	47	58,337	6171	89,051	6892	555,367
NYC	463	30	264,346	12,340	365,050	13,907	407,122

time on the candidate new edges includes the time spent on the shortest path search of selective new edges, and their connectivity increment based on the Lanczos method. Each new edge conducted the shortest path between its two ends, then we put the edge demand by summing up edges in the road network. Although this pre-processing is costly, it is called only once for each dataset but will benefit all the algorithms with various parameter settings.

7.1.4 Evaluation Metrics. For effectiveness evaluation, we first compare the increased objective value and connectivity value by our route, and then introduce three metrics to measure the transfer convenience of the new transit network, as shown in Table 6. We also conduct a visual analysis on our taxi trajectory dataset to plan new bus routes through visualization (Figure 7).

For efficiency evaluation, we compare the running time (Table 7) and verify the optimizations on accelerating the convergence, where we alter the parameter $k = [10, 20, \underline{30}, 40, 50]$, the weight $w = [0.3, \underline{0.5}, 0.7]$, the seeding number $sn = [3000, \underline{5000}, 7000]$, and the number of turns $Tn = [1, \underline{3}, 5]$ to observe the convergence performance. The default value is highlighted by an underline. The objective values are recorded every 100 times, and the iteration number is set as $100,000$ in Figures 9-11.

For the sake of unity of objective value in Definition 6, we choose same values for normalization. Specifically, we choose top-k edges and sum up their demands to set as the λ_{max} for the connectivity normalization in Equation 3. Similar operation applies to the demand normalization based on L_d.[6]

$$\lambda_{max} = \sum_{i=1}^{k} L_\lambda(i), \ d_{max} = \sum_{i=1}^{k} L_d(i) \qquad (13)$$

[6]Since $\sum_{i=1}^{k} L_\lambda(i)$ and $\sum_{i=1}^{k} L_d(i)$ are also used as upper bounds which are much bigger than O_d and O_λ, the objective values of the final results are usually small (e.g., Figure 9 & 10).

7.2 Effectiveness

We conduct both quantitative analysis and visual analysis, showing CT-Bus with **ETA** has the potential to increase connectivity and meet commuters' demand for Chicago and five boroughs of NYC.[7]

7.2.1 Comparisons. We implemented the following two most related approaches (see the last paragraph of Section 2) which optimize the connectivity [21, 62] and demand [59] by setting w as 0 and 1, respectively. Other approaches [24, 40, 58] which aim to optimize an existing bus route are essentially different from our work in term of problem formulation, hence we will not compare with them.

1) Connectivity-First Approaches. Chan et al. [21] proposed to maximize the natural connectivity of a graph when adding k new edges (not a path). We can use a greedy algorithm proposed by [21] to generate $l < k$ new edges to maximize the natural connectivity first, connect and order them using *travelling salesman search*, and then enrich the two ends by the shortest path. However, Figure 6 shows 10 edges returned by this method, and they are hard to be connected as a smooth bus route. In addition, this greedy algorithm needs several hours to complete. Hence, we will not conduct further comparison with this approach.

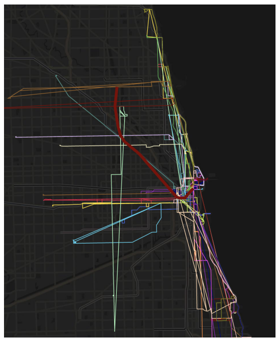
2) Demand-First Approaches. When optimizing the trajectory-based demand alone, similar to trajectory clustering [59], the problem will be a variant of k-TSP: *maximizing the sum of edges' demands with at most k new edges*. The difference with **k-TSP** is that the edges in the path should be newly connected. To increase the connectivity simultaneously, we set a constraint of new edges only here, as adding existing edges will not increase the network connectivity. We denote this baseline as **vk-TSP**. Note that Algorithm 1 applies a classical greedy methodology that can also work for the **vk-TSP**. To have a fair comparison when verifying the increments on connectivity in Section 7.2, we implemented it with the same configuration and made some minor changes: 1) setting $w = 1$; 2) only considering new edges during the initialization and expansion.

7.2.2 Results. **Analysis of Objective Values.** Table 6 shows the estimated connectivity and the objective value increase of new path by running CT-Bus in different areas, with our two algorithms. The connectivity values here are normalized by λ_{max} for a better illustration. We also compare the number of existing bus routes that can transfer to the route returned by our two methods, in order to verify whether the pre-computation sacrifices precision for efficiency. In each cell that has two numbers, the left is returned by online computation (**ETA**), the central is returned by pre-computation

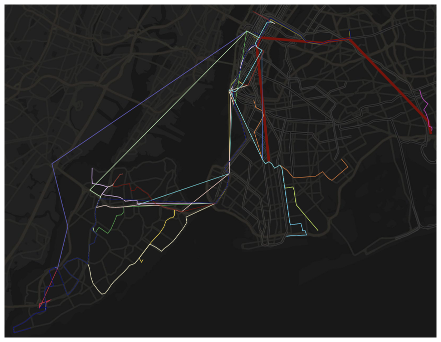
[7]In NYC, the bus transit system is planned independently in each borough.

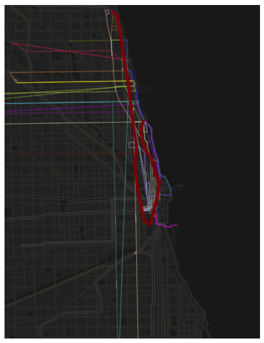
Table 6: Effectiveness analysis of planned routes. (normal cells: ETA | ETA-Pre | vk-TSP; gray cells: ETA-Pre with $w = 0 \mid 0.3 \mid 0.7$)

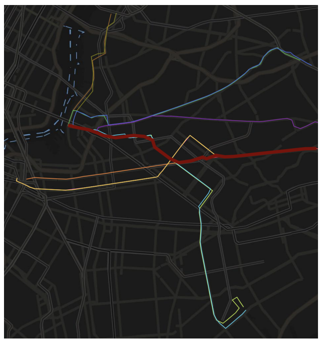
City	Improvement on Defined Metrics			Transfer Convenience Metrics		
	#New edges	Objective $O(\mu)$	Connectivity	#Transfer avoided	Distance ratio $\zeta(\mu)$	#Crossed routes
Chicago	29 | 29 | 22	0.22 | 0.22 | 0.06	0.20 | 0.19 | 0.05	3.02 | 3.15 | 2.33	5.35 | 5.90 | 5.45	41 | 30 | 25
	29 | 29 | 29	0.29 | 0.27 | 0.16	0.24 | 0.22 | 0.15	3.43 | 3.27 | 2.89	5.95 | 5.91 | 5.67	60 | 45 | 27
Manhattan	19 | 23 | 21	0.08 | 0.07 | 0.06	0.17 | 0.18 | 0.13	1.43 | 1.40 | 1.32	1.86 | 1.91 | 1.47	5 | 7 | 4
Queens	13 | 20 | 8	0.09 | 0.09 | 0.12	0.14 | 0.17 | 0.03	4.22 | 4.39 | 2.76	1.60 | 1.59 | 1.93	31 | 37 | 22
Brooklyn	26 | 26 | 6	0.11 | 0.10 | 0.04	0.22 | 0.23 | 0.03	1.39 | 1.36 | 1.25	2.44 | 2.85 | 1.16	13 | 17 | 5
Staten Island	11 | 11 | 6	0.09 | 0.09 | 0.08	0.16 | 0.16 | 0.05	1.93 | 1.89 | 1.67	3.66 | 3.83 | 3.64	42 | 40 | 34
Bronx	21 | 19 | 4	0.08 | 0.08 | 0.01	0.16 | 0.16 | 0.02	4.78 | **4.73** | **1.60**	6.38 | 7.07 | 1.32	20 | 17 | 8

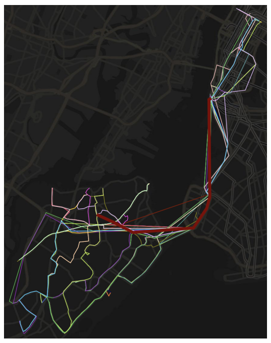

(a) Chicago

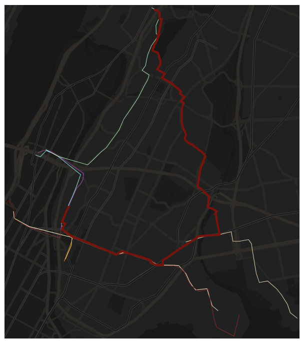
(b) Manhattan


(c) Queens


(a) $w = 1$ (vk-TSP)


(d) Brooklyn


(e) Staten Island


(f) Bronx

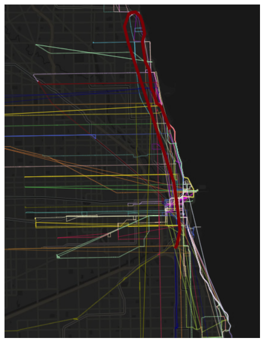

(b) $w = 0$

Figure 7: Visualization of a new bus route (bold red) and its connected existing routes ($w = 0.5$). **Figure 8: $w = 1$ & 0@Chicago.**

(**ETA-Pre**), and the right is returned by our baseline (**vk-TSP**). It shows that **ETA-Pre** and **ETA** have similar performance, and **ETA-Pre** is dominant in most cases.

We compare with **vk-TSP**, which plans a route by adding new edges to maximize the demand increment only and hence should also have a considerable connectivity increment, as shown in Table 6. However, we find **ETA-Pre** has a larger connectivity increment as we append new edges with high connectivity, while **vk-TSP** appends edges with high demand but may have low connectivity increment. We ignore the efficiency comparison as they use the same procedure in Algorithm 1.

Transfer Convenience. Since transfer convenience is one of the main performance indicators of connected transit networks [53], we further evaluate the newly planned route's effect to the commuters along it, which are composed of an origin stop and a destination stop in the new route. For every possible trip of these commuters, we run the shortest path search on the old and new bus networks, respectively. We use the following three metrics widely adopted in transportation evaluation area [14, 71], also shown in the right part of Figure 7, and a higher value indicates more convenience. Then, we calculate the average value for each metric.

Firstly, we calculate how many transfers are needed in the old bus network. Since there is no direct path like our new route, passengers need multiple transfers, e.g., it is 3.15 in Chicago. Secondly, for a group of trips, we calculate the ratio of shortest-path travel distance using the new bus network over that using the old bus network, as defined below:

$$\zeta(\mu) = \frac{1}{l(\mu) \cdot (l(\mu) - 1)} \cdot \sum_{\forall O,D \in \mu} \frac{|G_r(O,D)|}{|G'_r(O,D)|} \tag{14}$$

where O and D are any two different stops in μ, then there are $l(\mu) \cdot (l(\mu) - 1)$ possible pairs, and $l(\mu)$ is the number of stops in μ;

$|G_r(O, D)|$ denotes the travel distance of shortest path from O to D in G_r. The ratio $\zeta(\mu)$ is always bigger than one as passengers can directly commute without detour anymore in the new network $G_r^{'}$, and can have shorter travel distance. Thirdly, we count how many existing bus routes share common stops with the newly planned one. More crossed routes mean that passengers can easily transfer and get to more destinations in the network by taking other routes.

Effect of Varying the Weight w. To investigate the effect of w on the resulted routes, we set w as $\{0, 0.3, 0.5, 0.7, 1\}$ on Chicago, and observe how the metrics change w.r.t. w. We do not elaborate other five cities here due to space limit. Since **ETA-Pre** and **vk-TSP** were set with $w = 0.5$ and 1, the rest results of other values (0, 0.3, 0.7) can be found from the grayed row of Table 6.

Visual Analysis. In Figure 7, we visualize the planned route based on our algorithms and its existing connected routes in different colors to make sure it can present a complete profile of each route. Our newly planned routes are highlighted in bold red. By default, we set $w = 0.5$ in Figure 7. To further investigate the visualized effect of w, we set it as 0 and 1 respectively and compare the results in Figure 8 with those in Figure 7(a).

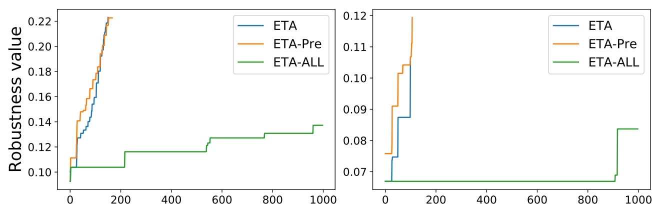
Insight 1: 1) CT-Bus based on our method can generate a valid path with a high objective value comparable to the one with online computation, and keep a balance between demand and connectivity. 2) Compared with **vk-TSP**, our method gets a much higher connectivity increment; we further verify that new routes with higher connectivity increments can dramatically avoid transfers for commuters with direct routes, and also provide more transfer choices especially in Chicago and Bronx. 3) The planned routes are very smooth in the map, and it also indicates the emerging trends and valuable suggestions for new bus routing planing in each city. 4) With more weight on the connectivity (i.e. smaller w), the connectivity increases linearly, such that it becomes easier to transfer (linearly increased metrics) and meanwhile more existing routes are connected.

Insight 2: For Chicago, CT-Bus suggests that one more route should be built to connect the northwest part (Avondale) to the city, as we can see, most existing bus lines are near the lakeside. Different with $w = 0.5$ in Figure 7(a), we observe that the route planned with $w = 1$ (considering demand only) crosses the city and coast area to meet high demands from passengers (see Figure 8(a)), but it intersects with much fewer routes (only 25) than the planned route of $w = 0$ (considering connectivity only), which connects 60 routes but crosses interior area mostly (see Figure 8(b)). The above analysis tells that a choice of $w = 0.5$ can make a good trade-off.

Insight 3: For NYC, more routes need to be built between Queens and Brooklyn, which will further connect more routes to Staten island. For Manhattan, existing subway and bus systems are very mature and connectivity increase will not be obvious, and newly planned bus routes are not necessary. This is also consistent with the fact that NYC is redesigning bus routes in the other four boroughs except for Manhattan. However, more routes should be planned to connect Manhattan with Staten Island which highly depends on buses, while there is only one internal subway line on the island. The Bronx also needs to connect north and south to form a circle from Yankee Stadium, Hunts Point Av, to Kingsbridge.

Table 7: Running time (s) comparison with increasing k.

	Chi-ETA	Chi-ETA Pre	NYC-ETA	NYC-ETA Pre
$k = 10$	22234.21	55.45	15011.55	37.55
$k = 20$	28291.92	76.88	16468.02	43.14
$k = 30$	30828.44	82.45	16567.51	41.17
$k = 40$	31967.53	88.32	16671.96	41.13
$k = 50$	32435.84	94.14	16686.87	44.97



(a) Chicago (b) NYC

Figure 9: Convergence comparison of ETA and ETA-Pre.

7.3 Efficiency

7.3.1 Comparisons. Since CT-Bus is proposed for the first time, we will mainly compare our proposed **ETA** with Lanczos and pre-computation optimizations introduced in Section 5 and 6, respectively. We denote them as **ETA** and **ETA-Pre**. Since ETA is based on the classical expansion-based traversal framework [59] which takes all edges as candidate routes to start expansion, we denote it as **ETA-ALL** for comparison.

7.3.2 Results. **Running Time of ETA.** To compare two algorithms fairly, we apply the same initialization with selective edges in Section 6.2 to avoid unnecessary scanning. Table 7 shows the running time using pre-computed connectivity and the **ETA** with our fast Lanczos method, by various k. We can observe that **ETA** with pre-computation (**ETA-Pre**) proposed in Section 6 is almost 400 times faster than **ETA** with online connectivity computation, as it has computed core information for fast connectivity and bound estimation off-line, and online computation is very limited.

Convergence of ETA. Figure 9 shows the convergence of our two methods with the increase of iteration number, where we estimate the final results of **ETA-Pre** using the Lanczos method and plot it as the last point. We observe that **ETA-Pre** has comparable and even higher objective values due to our tight upper bounds, and initializing all edges leads to slow convergence.

Parameter Sensitivity Test. To further verify the parameter-sensitivity of domination table (DT) and enqueueing best neighbors rather than all neighbors (AN), we add two more comparisons which mute them respectively. We test the sensitivity to four parameters: k, w, Tn, and sn. Also, we use **ETA-Pre** only since **ETA** is too slow to run such multiple rounds' tests. Figure 10 shows that despite the choice of w, our algorithm converges well and terminates at an early stage, because the feasibility checking has pruned all the candidate paths and the queue is empty. Figure 11 shows the sensitivity result on the rest parameters.

We find that the objective values drop with an increase of k (see Figure 10(b) and Figure 11(a)(b)), because our normalization values

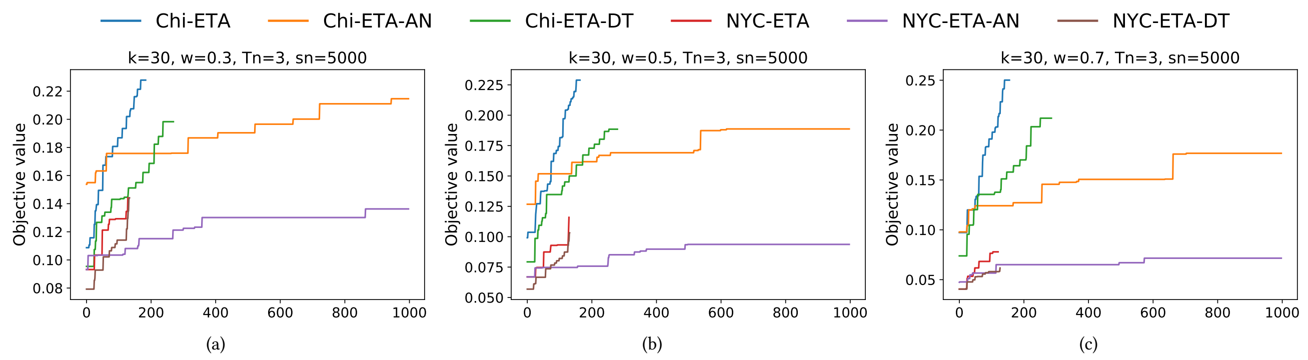


Figure 10: Parameter-sensitivity experiments on w.

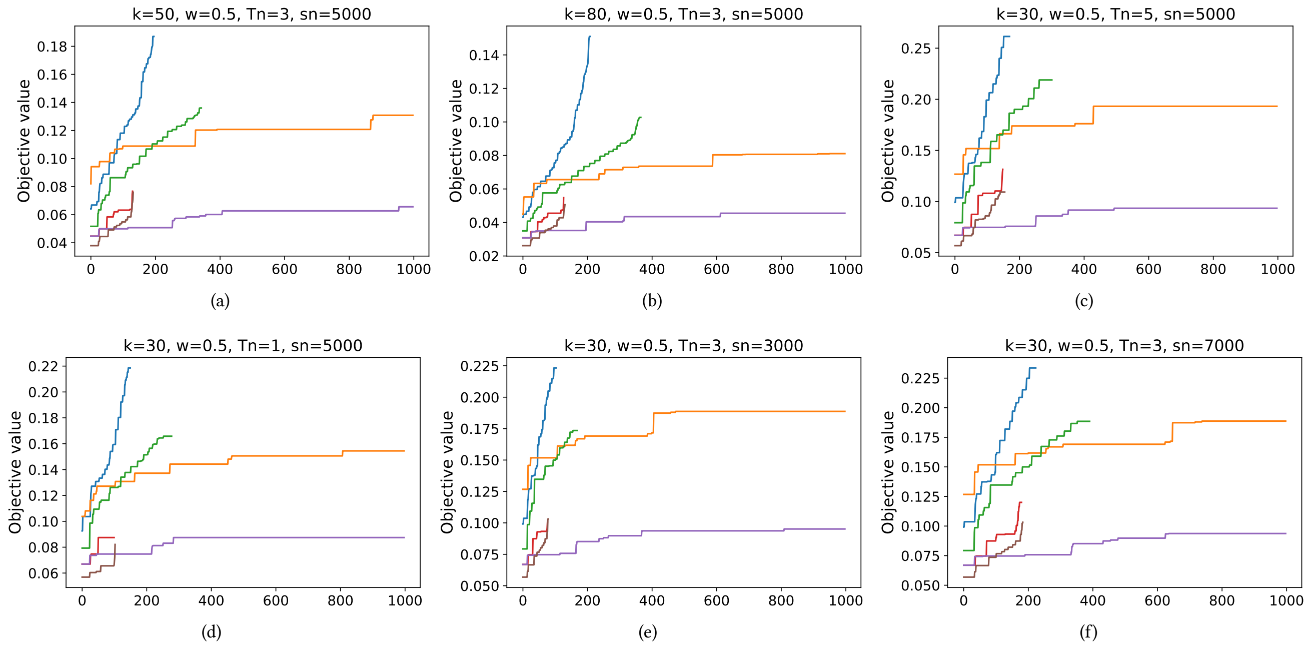


Figure 11: More parameter-sensitivity experiments on k, *Tn* (number of turns), and *sn* (seeding number).

d_{max} and λ_{max}, which are related to k in Equation 13, also rise but with a bigger increase rate than O_d and O_λ. For other parameters like w (see Figure 10), *Tn* (see Figure 10(b) and Figure 11(c)(d)), and *sn* (see Figure 10(b) and Figure 11(e)(f)), none of them has much impact to the convergence and efficiency.

Insight 4: 1) Our **ETA-Pre** method (i.e. with pre-computation) can converge in a short time and it is robust to various parameter settings. It also returns a highly similar objective score to the one with online connectivity computation, which is much slower. 2) Both the best neighbor only optimization strategy and the domination table optimization strategy can effectively prune candidates. 3) Pre-computation can be done in hours but it contributes to the high performance for interactive route planning [64].

8 CONCLUSIONS

We investigated a public transport route planning problem CT-Bus, which aims to plan a bus route to improve the connectivity of the transit network and also to meet the demand of commuters. We formulated CT-Bus as an optimization problem and proposed a practical heuristic method to solve it. To avoid computationally expansive matrix operations, we used the Lanczos method to estimate the natural connectivity of transit network with bounded error. We derived upper bounds on the objective values when adding edges, and used the derived upper bounds to select edges for greedy expansion. Our experiments showed that CT-Bus could plan effective routes in two of the US's most complicated bus transit systems.

In future, we will investigate how to update the connectivity efficiently in the pre-computation stage based on perturbation theory, and use our derived upper bounds to solve existing and new network connectivity optimization problems [21, 23]. Further, we will consider other alternative approaches for optimizing the two objectives in the CT-Bus problem, such as the *skyline* operator-based method [63] or simply treating one of the objectives as a constraint [58]. For small-scale cities that do not have sophisticated transit systems, the optimal site selection for deploying new bus stops based on trajectories and connectivity will be another interesting direction for future research.

REFERENCES

[1] 2014. The Rise and Fall of Manhattan's Density. https://urbanomnibus.net/2014/10/the-rise-and-fall-of-manhattans-density.

[2] 2019. MTA's Bronx bus redesign will chop 400 stops and add new routes. https://ny.curbed.com/2019/10/22/20926765.

[3] 2020. 2016 (May) Manhattan Bus Stops. https://geo.nyu.edu/catalog/nyu-2451-34693.

[4] 2020. 9th DIMACS Implementation Challenge - Shortest Paths. http://users.diag.uniroma1.it/challenge9/download.shtml.

[5] 2020. Bus Network Redesign. https://new.mta.info/system_modernization/bus_network.

[6] 2020. Bus stop spacing and location. www.transitwiki.org/TransitWiki/index.php/Bus_stop_spacing_and_location.

[7] 2020. Chicago Taxi Trips. https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew.

[8] 2020. CTA - Bus Routes - Shapefile. https://catalog.data.gov/dataset/cta-bus-routes-shapefile.

[9] 2020. Mapv. https://github.com/huiyan-fe/mapv.

[10] 2020. NetworkX. https://networkx.github.io.

[11] 2020. Repository of CT-Bus. https://github.com/rp4ps/ct-bus.

[12] 2020. TLC Trip Record Data. https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page.

[13] 2020. Urban Bus Toolkit. www.ppiaf.org/sites/ppiaf.org/files/documents/toolkits/UrbanBusToolkit/assets/1/1d/1d4.html.

[14] Hatem Abdelaty, Moataz Mohamed, Mohamed Ezzeldin, and Wael El-Dakhakhni. 2020. Quantifying and classifying the robustness of bus transit networks. *Transportmetrica A: Transport Science* 16, 3 (2020), 1176–1216.

[15] Sanjeev Arora. 2003. Approximation schemes for NP-hard geometric optimization problems: a survey. *Mathematical Programming* 97, 1 (2003), 43–69.

[16] Haim Avron and Sivan Toledo. 2011. Randomized Algorithms for Estimating the Trace of an Implicit Symmetric Positive Semi-Definite Matrix. *J. ACM* 58, 2, Article 8 (April 2011), 34 pages.

[17] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. 2016. Route Planning in Transportation Networks. In *Algorithm Engineering*. 19–80.

[18] Gernot Veit Batz and Peter Sanders. 2012. Time-dependent route planning with generalized objective functions. In *ESA*. 169–180.

[19] Gabriela Beirão and J. A. Sarsfield Cabral. 2007. Understanding attitudes towards public transport and private car: A qualitative study. *Transport Policy* 14, 6 (2007), 478–489.

[20] Rajendra Bhatia. 2001. Linear Algebra to Quantum Cohomology: The Story of Alfred Horn's Inequalities. *The American Mathematical Monthly* 108, 4 (2001), 289–318.

[21] Hau Chan, Leman Akoglu, and Hanghang Tong. 2014. Make it or break it: Manipulating robustness in large networks. In *SDM*. 325–333.

[22] Lijun Chang, Xuemin Lin, Lu Qin, Jeffrey Xu Yu, and Wenjie Zhang. 2015. Index-based optimal algorithms for computing steiner components with maximum connectivity. In *SIGMOD*. 459–474.

[23] Chen Chen, Ruiyue Peng, Lei Ying, and Hanghang Tong. 2018. Network Connectivity Optimization: Fundamental Limits and Effective Algorithms. In *KDD*. 1167–1176.

[24] Chao Chen, Daqing Zhang, Nan Li, and Zhi Hua Zhou. 2014. B-planner: Planning bidirectional night bus routes using large-scale taxi GPS traces. *IEEE Transactions on Intelligent Transportation Systems* 15, 4 (2014), 1451–1465.

[25] José Antonio De La Peña, Ivan Gutman, and Juan Rada. 2007. Estimating the Estrada index. *Linear Algebra and Its Applications* 427, 1 (2007), 70–76.

[26] Floridea Di Ciommo and Yoram Shiftan. 2017. Transport equity analysis. *Transport Reviews* 37, 2 (2017), 139–151.

[27] Kun Dong, Austin R. Benson, and David Bindel. 2019. Network density of states. In *KDD*. 1152–1161.

[28] Ernesto Estrada. 2000. Characterization of 3D molecular structure. *Chemical Physics Letters* 319, 5-6 (2000), 713–718.

[29] Ky Fan. 1949. On a Theorem of Weyl Concerning Eigenvalues of Linear Transformations. I. *Proceedings of the National Academy of Sciences* 35, 11 (1949), 652–655.

[30] Wei Fan and Randy B Machemehl. 2006. Optimal Transit Route Network Design Problem with Variable Transit Demand: Genetic Algorithm Approach. *Journal of Transportation Engineering* 132, 1 (2006), 40–51.

[31] Miroslav Fiedler. 1973. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal* 23, 2 (1973), 298–305.

[32] Naveen Garg. 2005. Saving an Epsilon: a 2-approximation for the k-MST problem in graphs. In *STOC*. 396–402.

[33] Robert Geisberger, Moritz Kobitzsch, and Peter Sanders. 2010. Route planning with flexible objective functions. In *ALENEX*. 124–137.

[34] Valérie Guihaire and Jin-Kao Hao. 2008. Transit network design and scheduling: A global review. *Transportation Research Part A: Policy and Practice* 42, 10 (2008), 1251–1273.

[35] Aldy Gunawan, Hoong Chuin Lau, and Pieter Vansteenwegen. 2016. Orienteering Problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research* 255, 2 (2016), 315–332.

[36] M. F. Hutchinson. 1990. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics - Simulation and Computation* 19, 2 (1990), 433–450.

[37] Junfeng Jiao and Maxwell Dillivan. 2013. Transit deserts: The gap between demand and supply. *Journal of Public Transportation* 16, 3 (2013), 23–39.

[38] Sigal Kaplan, Dmitrijs Popoks, Carlo Giacomo Prato, and A. Ceder. 2014. Using connectivity for measuring equity in transit provision. *Journal of Transport Geography* 37 (2014), 82–92.

[39] Jean B Lasserre. 1995. A trace inequality for matrix product. *IEEE Trans. Automat. Control* 40, 8 (1995), 1500–1501.

[40] Yanchi Liu, Chuanren Liu, Nicholas Jing Yuan, Lian Duan, Yanjie Fu, Hui Xiong, Songhua Xu, and Junjie Wu. 2016. Intelligent bus routing with heterogeneous human mobility patterns. *Knowledge and Information Systems* 50, 2 (2016), 383–415.

[41] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. 2009. Map-matching for low-sampling-rate GPS trajectories. In *GIS*. 352–361.

[42] Songsong Mo, Zhifeng Bao, Baihua Zheng, and Zhiyong Peng. 2020. *Bus Frequency Optimization: When Waiting Time Matters in User Satisfaction.* Technical Report. arXiv:2004.07812v1

[43] Songsong Mo, Zhifeng Bao, Baihua Zheng, and Zhiyong Peng. 2020. FASTS : A Satisfaction-Boosting Bus Scheduling Assistant. *PVLDB* 13, 12 (2020), 2873–2876.

[44] Cameron Musco and Christopher Musco. 2015. Randomized Block Krylov Methods for Stronger and Faster Approximate Singular Value Decomposition. In *NeurIPS*. 1396–1404.

[45] Cameron Musco, Christopher Musco, and Aaron Sidford. 2018. Stability of the Lanczos method for matrix function approximation. In *SODA*. 1605–1624.

[46] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. 1978. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming* 14, 1 (1978), 265–294.

[47] Zeev Nutov. 2009. Approximating connectivity augmentation problems. *ACM Trans. Algor.* 6, 1 (2009), 1–19.

[48] Lorenzo Orecchia, Sushant Sachdeva, and Nisheeth K Vishnoi. 2012. Approximating the exponential, the lanczos method and an $\tilde{O}(m)$-time spectral algorithm for balanced separator. In *STOC*. 1141–1160.

[49] Fabio Pinelli, Rahul Nair, Francesco Calabrese, Michele Berlingerio, Giusy Di Lorenzo, and Marco Luca Sbodio. 2016. Data-driven transit network design from mobile phone trajectories. *IEEE Transactions on Intelligent Transportation Systems* 17, 6 (2016), 1724–1733.

[50] Farbod Roosta-Khorasani and Uri Ascher. 2015. Improved bounds on sample size for implicit matrix trace estimators. *Foundations of Computational Mathematics* 15, 5 (2015), 1187–1212.

[51] Preston L Schiller and Jeffrey R Kenworthy. 2017. *An introduction to sustainable transportation: Policy, planning and implementation.* Routledge.

[52] Sushant Sharma, Satish V Ukkusuri, and Tom V Mathew. 2009. Pareto Optimal Multiobjective Optimization for Robust Transportation Network Design Problem. *Transportation Research Record* 2090, 1 (2009), 95–104.

[53] Daniel (Jian) Sun, Shukai Chen, Chun Zhang, and Suwan Shen. 2016. A bus route evaluation model based on GIS and super-efficient data envelopment analysis. *Transportation Planning and Technology* 39, 4 (may 2016), 407–423.

[54] Yongxin Tong, Yuxiang Zeng, Zimu Zhou, Lei Chen, Jieping Ye, and Ke Xu. 2018. A unified approach to route planning for shared mobility. *PVLDB* 11, 11 (2018), 1633–1646.

[55] Shashanka Ubaru, Jie Chen, and Yousef Saad:. 2017. Fast Estimation of tr(f(A)) via Stochastic Lanczos Quadrature. *SIAM J. Matrix Analysis Applications* 38, 4 (2017), 1075–1099.

[56] Shashanka Ubaru and Yousef Saad. 2018. Applications of trace estimation techniques. In *HPCSE*. 19–33.

[57] Jiachuan Wang, Peng Cheng, Libin Zheng, Chao Feng, Lei Chen, Xuemin Lin, and Zheng Wang. 2020. Demand-aware route planning for shared mobility services. *PVLDB* 13, 7 (2020), 979–991.

[58] Sheng Wang, Zhifeng Bao, J Shane Culpepper, Timos Sellis, and Gao Cong. 2018. Reverse k nearest neighbor search over trajectories. *IEEE Transactions on Knowledge and Data Engineering* 30, 4 (2018), 757 – 771.

[59] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Timos Sellis, and Xiaolin Qin. 2019. Fast large-scale trajectory clustering. *PVLDB* 13, 1 (2019), 29–42.

[60] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Zizhe Xie, Qizhi Liu, and Xiaolin Qin. 2018. Torch: A search engine for trajectory data. In *SIGIR*. 535–544.

[61] Sibo Wang, Wenqing Lin, Yi Yang, Xiaokui Xiao, and Shuigeng Zhou. 2015. Efficient Route Planning on Public Transportation Networks : A Labelling Approach. In *SIGMOD*. 967–982.

[62] P. Wei, L. Chen, and D. Sun. 2014. Algebraic connectivity maximization of an air transportation network: The flight routes' addition/deletion problem. *Transportation Research Part E: Logistics and Transportation Review* 61 (jan 2014), 13–27.

[63] Di Weng, Ran Chen, Jianhui Zhang, Jie Bao, Yu Zheng, and Yingcai Wu. 2020. Pareto-Optimal Transit Route Planning With Multi-Objective Monte-Carlo Tree Search. *IEEE Transactions on Intelligent Transportation Systems* (2020), 1–11.

[64] Di Weng, Chengbo Zheng, Zikun Deng, Mingze Ma, Jie Bao, Yu Zheng, Mingliang Xu, and Yingcai Wu. 2020. Towards Better Bus Networks: A Visual Analytics Approach. *IEEE Transactions on Visualization and Computer Graphics* (2020). arXiv:2008.10915

[65] Douglas Brent West and Others. 1996. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, NJ.

[66] Guojun Wu, Yanhua Li, Jie Bao, Yu Zheng, Jieping Ye, and Jun Luo. 2018. Human-Centric Urban Transit Evaluation and Planning. In *ICDM*. 547–556.

[67] Jun Wu, Barahona Mauricio, Yue Jin Tan, and Hong Zhong Deng. 2010. Natural connectivity of complex networks. *Chinese Physics Letters* 27, 7 (2010).

[68] Da Yan, James Cheng, Kai Xing, Yi Lu, Wilfred Ng, and Yingyi Bu. 2014. Pregel algorithms for graph connectivity problems with performance guarantees. *PVLDB* 7, 14 (2014), 1821–1832.

[69] Ping Zhang, Zhifeng Bao, Yuchen Li, Guoliang Li, Yipeng Zhang, and Zhiyong Peng. 2018. Trajectory-driven influential billboard placement. In *KDD*. 2748–2757.

[70] Yaoming Zhou, Junwei Wang, and Jiuh Biing Sheu. 2019. On connectivity of post-earthquake road networks. *Transportation Research Part E: Logistics and Transportation Review* 123 (mar 2019), 1–16.

[71] Zhiyun Zou, Yao Xiao, and Jianzhi Gao. 2013. Robustness analysis of urban transit network based on complex networks theory. *Kybernetes* 42, 3 (2013), 383–399.