

VIGO: A Millisecond-Scale Transit Routing Engine for Disaggregate Accessibility Analysis

Hanyang Tang^{1,*}, Yishu Wang², Michael Leong¹ and Hongyi Lin^{3,*}

¹Massachusetts Institute of Technology · ²Independent Researcher · ³Tsinghua University

*Corresponding authors: Hanyang Tang and Hongyi Lin

2026-10-15

Abstract

Disaggregate accessibility analysis measures access for individual locations and departure times, so it repeats transit routing between many coordinate origins and destinations. Its cost lies in complete coordinate-to-coordinate requests, not in timetable search alone. VIGO, an open-source routing engine, reduces this cost with a two-stage search. A fast first pass, temporal propagation, finds the earliest arrival or latest departure, keeping one time per search state. A second pass, journey selection, then minimizes boardings and walking time among the journeys that attain this time, while bounds discard partial journeys that can no longer attain it. Compact event records, reusable search storage and search shared across destinations reduce the remaining work. With prepared networks held in memory, VIGO’s median end-to-end time on six U.S. networks ranges from 0.67 ms in Anchorage to 2.11 ms in New York, and its 99th percentile from 1.35 ms to 6.29 ms, on the requests that all nine compared routing interfaces answer. On 1,152 feasible requests, the bounds cut the partial journeys that journey selection creates to under 1% of an unbounded search and make the complete two-stage search 4.5 to 8.1 times faster. Independent whole-ride enumeration agrees with VIGO on all 2,400 cases from 150 small generated networks. VIGO answers fewer requests than some engines because its walking-access test is stricter; on the 216 New York requests that R5 answers and VIGO blocks, the source OpenStreetMap and GTFS data support VIGO’s decision in 187. With timed legs, a request from one origin to 1,000 destinations takes a median of under 160 ms in every city.

Keywords: coordinate-to-coordinate transit routing; journey selection; accessibility; connection scan; customizable contraction hierarchies

1. Introduction

Public transport accessibility depends on when and where a journey starts. A short walk decides which departure a traveler catches, and one missed transfer changes the destinations reachable within a travel-time threshold (Geurs and van Wee, 2004; Owen and Levinson, 2015). Planners use such measures to see who can reach jobs, schools and health care, and how a service change would alter that reach. Disaggregate analysis repeats these calculations for each individual location and departure time, often under several walking constraints or service scenarios. Fast timetable algorithms answer stop-to-stop queries in under a millisecond on large networks (Großmann et al., 2026), but an accessibility request also pays for walking access, multicriteria journey selection and output, and analyses repeat the same search across destinations

and departure times. Scaling such analyses requires removing this repeated and unnecessary work without changing any individual answer.

VIGO is an open-source transit routing engine built for this workload: it prepares OSM street and GTFS timetable data once and then answers coordinate-to-coordinate requests from memory. It targets three sources of cost: unnecessary search states, expensive state processing and repeated computation across requests. *Efficient search* finds the earliest arrival or latest departure with one time label per state and only then runs the more expensive journey selection, which looks for fewer boardings and less walking time. *Efficient representation* rejects unreachable departures from a compact record and reuses search storage across requests. *Efficient orchestration* keeps the prepared network in memory, shares the search for the best time across compatible cells of a Matrix call, a single request for many origin–destination pairs, and reconstructs journeys only when they are requested.

Most of the algorithmic saving comes from postponing the comparison of alternative journeys until the best time is known. Suppose two journeys both reach the destination at 08:30, the earliest possible arrival: one boards two vehicles and walks ten minutes, the other boards one vehicle and walks fifteen. VIGO returns the second, because fewer boardings take priority over walking time. The first pass establishes 08:30 cheaply, keeping a single best time for each stop and travel state. The second pass then chooses among the journeys that reach 08:30. Along the way it keeps partial journeys that reach an intermediate stop later but with less walking, because they may still catch the same vehicle. It discards, before they cost memory or comparisons, the partial journeys that would need too many boardings or could not reach the destination by 08:30. Section 4.4 measures this saving.

This paper makes three contributions. First, it formulates a two-stage bounded search for the ordered coordinate-to-coordinate objective and states the conditions under which its bounds preserve the optimum. Second, it describes an execution model in which a long-running process keeps the network in memory, reusable workspaces and groups of compatible requests that share one search remove repeated work, and two runtimes, Node.js and a standalone Rust executable, share one Rust search and output layer. Third, it connects algorithmic savings to complete routing workloads through ablations, independent enumeration, Matrix experiments and an audit of reachability disagreements.

Section 2 reviews related work, Section 3 describes the method and architecture, Section 4 evaluates performance and correctness, Section 5 discusses the implications and limitations, and Section 6 concludes.

2. Literature Review

Accessibility resolution. Accessibility describes how easily people can reach opportunities such as jobs or shops, often counted as the destinations reachable within a travel-time threshold. Accessibility measures differ in their treatment of opportunities, travel effort and activity constraints (Geurs and van Wee, 2004; Kwan, 1998). For public transport, departure-time variation (Owen and Levinson, 2015; Järv et al., 2018), uncertainty (Conway et al., 2018) and spatial representation are distinct modeling choices. Departure-time sampling changes public transport accessibility estimates (Stepniak et al., 2019); building-level and aggregated origins can likewise yield different measurements, particularly under lower service levels and tighter travel-time thresholds (Qu et al., 2025).

Timetable search. A transit timetable is a set of dated vehicle trips, each a sequence of stop departures and arrivals; the basic routing task is to find the earliest arrival at a destination from a given departure. The principal timetable algorithms organize search around different units (Bast et al., 2016). The Connection Scan Algorithm (CSA) processes scheduled connections in time order; its variants include profile search and journey reconstruction, with starting and stopping criteria that shorten the scanned interval (Dibbelt

et al., 2018). The Round-based Public Transit Optimized Router (RAPTOR) scans transit routes in boarding rounds and retains arrival-time tradeoffs with boarding count without timetable preprocessing (Delling et al., 2012, 2015). Trip-Based Routing prepares feasible transfers between vehicle trips and searches those relations (Witt, 2015).

FLASH-TB is among the fastest published stop-to-stop methods. It uses destination-specific preprocessing to shorten Trip-Based search: Arc-Flags precompute, for each region of destinations, which transfers can lie on an optimal journey to that region. Its fastest configurations report sub-millisecond stop-to-stop queries on countrywide networks (Großmann et al., 2026). Complete coordinate-to-coordinate requests add street attachment and journey reconstruction, which together with timetable search determine end-to-end time.

Departure-window methods address repetition across time. CSA profile search computes nondominated departure–arrival pairs over an interval. Range RAPTOR (rRAPTOR) processes candidate departures from latest to earliest, retaining labels so that later-departure journeys can prune earlier searches (Dibbelt et al., 2018; Delling et al., 2015). VIGO instead shares search across destinations at one fixed departure time (Section 3.4).

Bounded multicriteria search. Several journeys can reach the destination at the same earliest time with different boarding counts or walking times. Multicriteria search retains the Pareto set of nondominated journeys: no feasible alternative is at least as good in every criterion and strictly better in one. Multicriteria RAPTOR (McRAPTOR) computes such sets in boarding rounds (Delling et al., 2015). Bounded McRAPTOR defines a restricted Pareto set through allowances in time and boarding count relative to reference optima and computes that set exactly (Delling et al., 2019). Its variant Tight-BMRAP uses forward and reverse reference searches to obtain bounds for each stop and boarding round, excluding labels that cannot contribute to the requested set (Delling et al., 2019).

Potthoff and Sauer (2021) combine three-criteria journey planning with ULTRA, a preprocessing step that precomputes walking transfers between vehicle trips (described below), and restricted Pareto sets, optimizing arrival time, trip count and transfer-mode time. VIGO applies reference bounds to a single journey chosen by an ordered objective, which ranks journeys by arrival time, then boardings, then walking: fixing the earliest arrival first limits the alternatives that must be compared.

Walking and one-to-many routing. Walking connects coordinates to transit stops and connects successive rides. In the CSA formulation, a transitively closed footpath graph, in which every pair of stops within walking reach is linked directly, permits direct transfer relaxation, but closure can become dense in large walking components (Dibbelt et al., 2018). ULTRA prepares transfer shortcuts that preserve Pareto-optimal journeys under its multimodal model, while street searches supply initial and final transfers (Baum et al., 2019, 2023).

Endpoint attachment can use road-network hierarchies. Customizable Contraction Hierarchies (CCH) separate topology preparation from the assignment of directed costs (Dibbelt et al., 2016). Bucket-based many-to-many search stores target distances at hierarchy vertices; a source search reads these records to obtain distances to many targets (Knopp et al., 2007). VIGO prepares stop buckets in a directed CCH for this purpose.

Sauer et al. (2020) combine ULTRA with street searches for a target set in one-to-many multimodal routing, demonstrating the approach with CSA and RAPTOR. VIGO likewise shares one timetable search across destinations, with directed walks to and from each endpoint and prepared walking transfers between rides; when full journeys are requested, it also shares the multicriteria rounds and selects each destination’s journey under its own objective.

Routing systems. Several open-source routing systems combine OSM street networks with GTFS timetables to answer coordinate-to-coordinate transit requests. R5 was built for accessibility analysis (Conway et al., 2017), and its R and Python packages, `r5r` and `r5py`, bring it into analysis workflows (Pereira et al., 2021; Fink et al., 2022). Its range routing reuses search state across departure minutes,¹ so a departure-window accessibility task reuses work differently from repeated requests at fixed times. OpenTripPlanner (OTP) and MOTIS are multimodal journey planners (OpenTripPlanner Project, 2026; MOTIS Project, 2026), while Valhalla and GraphHopper are street routing engines that also support scheduled transit (Valhalla Project, 2026; GraphHopper, 2026). Because each system has its own routing model and implementation, the same request can produce different journeys and accessibility values (Higgins et al., 2022).

What remains unmeasured is the cost of complete requests. Published speedups concern stop-to-stop timetable search, while an accessibility workload pays for walking access, journey selection and output on every request, and the systems analysts use differ in what they treat as reachable (Higgins et al., 2022). Existing methods establish that reference bounds accelerate multicriteria routing and that timetable propagation can be shared across destinations; neither principle is new here. VIGO's contribution is to organize both around a single coordinate-to-coordinate journey chosen by an ordered objective. A scalar temporal pass establishes the reference optimum and a feasible boarding count, conservative forward/reverse bounds restrict the journey selection that follows, and selection returns one exact lexicographic optimum; sharing extends through the complete request, from endpoint attachment to output. The evaluation measures whether this combination lowers the cost of complete requests.

3. Routing Method and Architecture

This section follows a request through VIGO: what is prepared in advance (3.1), how a two-stage search finds the journey (3.2), how compact data keeps that search cheap (3.3), and how Matrix calls share work across many cells (3.4). VIGO compiles General Transit Feed Specification (GTFS) and OpenStreetMap (OSM) data into a *prepared network*, stored on disk as a City. Route handles a single origin–destination request, and Matrix evaluates many origin–destination cells in one call. A Rust kernel performs all street and timetable search for two runtimes: a Node.js runtime that calls the kernel through Node's addon interface (N-API), and a standalone Rust executable with command-line, streaming and HTTP transports. Both runtimes share the kernel and the Rust components that build transit lines, walking legs, fares and the public output. The Python package, VIGO-py, sends requests to a long-running Node.js process. Figure 1 summarizes the design.

Requests and the ordered objective. Every request asks for one best journey under a fixed ranking. A point request gives origin and destination coordinates, a service date, a depart-at time t_0 or arrive-by deadline U , an endpoint walking limit and a boarding limit B , the maximum number of vehicle boardings. For a transit journey, let a be arrival time, d departure time, b the number of boardings and w walking time. Depart-at minimizes (a, b, w) and arrive-by minimizes $(-d, b, w, a)$, comparing entries in sequence; we call this lexicographic comparison the *ordered objective*. The optimum of its first entry is the *temporal optimum*: the earliest arrival a^* for depart-at or the latest departure d^* for arrive-by. Informally, a depart-at request prefers the earliest arrival, then fewer boardings, then less walking; an arrive-by request prefers the latest departure that still meets the deadline, hence $-d$, with the same tie-breakers and finally the earlier arrival. A feasible journey uses at most B boardings, permitted directed walking paths and scheduled events within the search horizon; depart-at journeys leave at or after t_0 , and arrive-by journeys arrive by U . The endpoint

¹R5 7.4, FastRaptorWorker: <https://github.com/conveyal/r5/blob/v7.4/src/main/java/com/conveyal/r5/profile/FastRaptorWorker.java>.

walking limit constrains walking distance, including station paths. Walking time w is a modeled quantity: it sums the prepared durations of access, transfer and egress paths, excluding waiting and riding, and where a published minimum transfer time exceeds the platform walk, the minimum is charged instead. When a request permits a street-only walking journey, the walk is evaluated under its own limits and wins ties on the temporal optimum because it uses no boardings.

3.1 Prepared network and endpoint attachment

Preparation does the expensive work that does not depend on any request, once, so that each request only has to search. It stores a street index for fast walking searches, an index from streets to nearby stops and compact timetable arrays, alongside the source data in SQLite. A *resident process* keeps the prepared network and search workspaces in memory across requests. For each request it loads a compatible timetable or builds one for the requested service day, placing after-midnight events and service from neighboring dates on one elapsed-time axis. The timetable is reused while its date, transfer settings and time coverage remain compatible.

Endpoint attachment determines, for each endpoint, which stops can be reached on foot and how long each walk takes. It is needed because requests start and end at arbitrary coordinates. Attachment snaps each coordinate to a street vertex within 160 m or a closer point on a two-way edge within 80 m (Table 8); if that street has no walking route to any stop, a bounded search looks for a nearby street that does. Customizable Contraction Hierarchies (CCH) then compute directed walking costs from the origin to access stops and from egress stops to the destination. Prepared stop buckets let one hierarchy search reach many candidate stops; the bucket scan stops once the remaining distances exceed the walking limit, and the destination search uses backward weights so that walking direction is preserved. Because a faster station path can be longer, an endpoint alternative is discarded only when another is both no slower and no longer.

Walking between rides is prepared in advance as transfers: GTFS transfers and pathways, supplemented by directed OSM walks between stops. Transfer durations enter the search as walking time; published transfer minima and prohibitions apply, and no global boarding buffer is added.

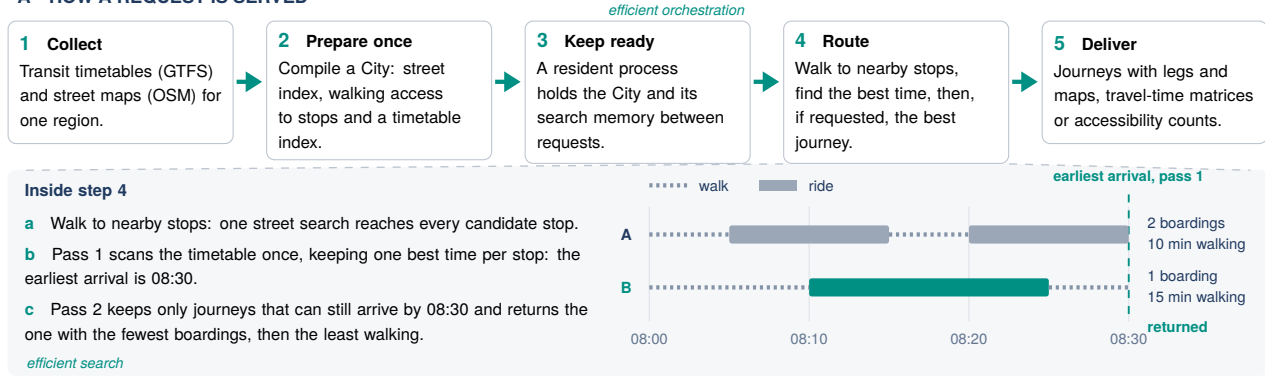
3.2 Two-stage search

VIGO searches the timetable in two passes. The first pass, temporal propagation, finds only the best time; the second, journey selection, chooses among the journeys that achieve it.

Temporal propagation. Temporal propagation finds the earliest possible arrival, and nothing else, as cheaply as possible. For depart-at, it keeps a *label* for each stop and travel state: the earliest known arrival time there. It starts at the access stops with t_0 plus the walking time to each, and scans the day's connections, scheduled vehicle movements between consecutive stops, in departure-time order, as in CSA. A connection is catchable if the traveler reaches its stop in time or is already aboard the same vehicle run, a part of a dated trip that allows continuation without another boarding. When walking transfers before the first ride are disabled, a binary search over each access stop's departures finds its first catchable connection, and the scan starts at the earliest of these. Destination stopping ends a point scan once the next departure plus the minimum egress time exceeds the best arrival found, keeping equal-arrival candidates; Matrix scans continue to the search horizon. Arrive-by runs the same procedure backward from the deadline.

Together, the stop and travel state form a *routing state*. The travel state records whether a ride has occurred, whether egress is permitted and whether a same-stop boarding minimum still applies; onboard states add the vehicle run and position. Two labels at the same stop with different onward actions cannot be merged

A HOW A REQUEST IS SERVED



B WHAT RUNS WHERE

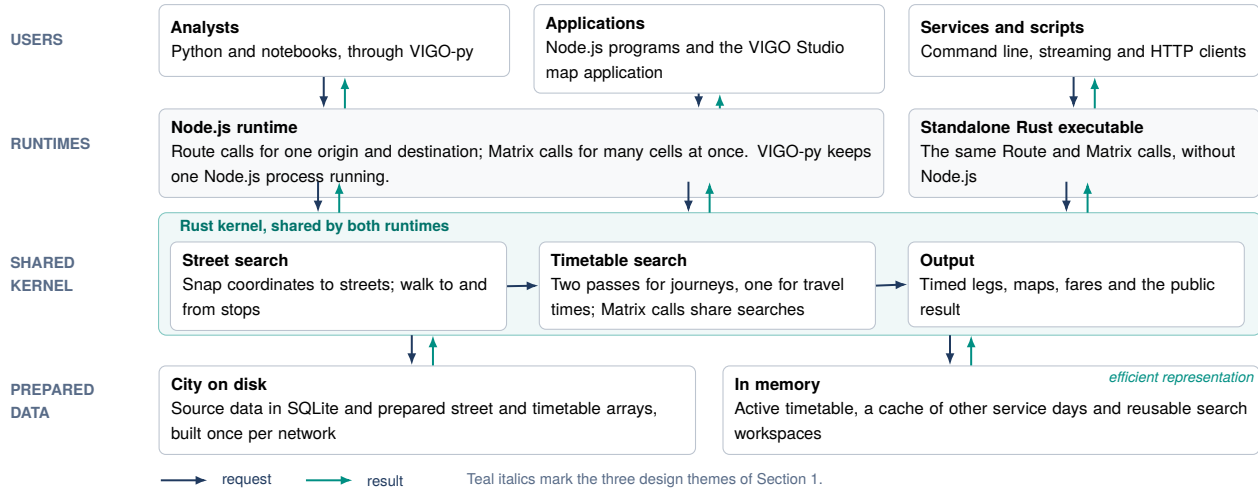


Figure 1: How VIGO works. (A) The path of a request; the inset shows the two passes on the example of Section 1, where journey B wins with one boarding instead of two. (B) Where each step runs.

by time alone. Connections with the same departure time are scanned repeatedly until no label improves, because a connection later in the group can make an earlier one boardable.

Boarding limit. This step makes the earliest arrival respect the request’s boarding limit B . Depart-at first propagates without the limit. If the winning journey uses at most B boardings, its arrival is also optimal under the limit (Appendix A). Otherwise VIGO reruns propagation with one layer per boarding count, $0, \dots, B$. Arrive-by accepts the unrestricted latest departure only after a forward search confirms a journey from it within the deadline and boarding limit; otherwise it uses layered reverse propagation.

Journey selection. Journey selection picks, among the journeys that reach the destination at the temporal optimum, the one with the fewest boardings and then the least walking. It adds one boarding per round and keeps, within each routing state and boarding layer, labels that trade arrival time against walking time. A later arrival with less walking can still catch the same service: labels that reach a stop at time 10 after eight minutes of walking and at time 12 after three minutes can both board a departure at time 15, and only the second reaches the destination with less walking. One label dominates another only if it arrives no later, walks no more and permits the same onward actions. The selected journey lies on the journey Pareto frontier, because any journey that dominates it would also come first in the ordered comparison; fixing the temporal optimum first excludes later journeys with fewer boardings or less walking.

Bounds. A bound is a limit, taken from temporal propagation, that lets journey selection skip partial journeys that cannot become the answer. Suppose the earliest feasible arrival is 08:40, reached with two boardings. A journey arriving after 08:40 cannot win, and among journeys arriving at 08:40, one with more than two boardings cannot win either. Many partial journeys within these limits are still useless because they cannot reach the destination by 08:40; reachability computed forward from the origin and backward from the destination identifies them. Every bound is conservative: a state whose fate is uncertain is kept, not discarded.

Formally, three bounds restrict journey selection. The arrival bound is a^* for depart-at and U for arrive-by. The boarding bound is the boarding count b_0 of a journey attaining a^* , since a journey with more boardings cannot improve on it at the same arrival. Forward/reverse bounds keep only vehicle runs reachable from the origin and stop times from which the destination can still be reached in time; they reuse the runs reached by temporal propagation and keep every run in any interval that destination stopping skipped. The arrival bound alone leaves many partial journeys that arrive early but need too many boardings, ride unreachable runs or cannot continue to the destination; the other two bounds remove them before they create selection labels. If the bounds leave no starting state, Route retries with the arrival and boarding bounds only. For arrive-by, journey selection runs forward from d^* under the original deadline and stops after the first round that reaches the minimum feasible boarding count. Algorithm 1 summarizes the depart-at case, and Appendix A gives the conditions under which the bounds preserve the optimum.

Algorithm 1 Depart-at journey selection

Require: Prepared model, directed endpoint alternatives, t_0 , horizon, boarding limit B

- 1: Compute the unrestricted earliest arrival and a journey attaining it
 - 2: **if** no transit journey exists in the prepared model **then**
 - 3: **return** no transit journey
 - 4: **end if**
 - 5: **if** that journey uses more than B boardings **then**
 - 6: Recompute earliest arrival using boarding layers $0, \dots, B$
 - 7: **end if**
 - 8: **if** no journey satisfies the boarding limit **then**
 - 9: **return** no transit journey
 - 10: **end if**
 - 11: Let a^* be the feasible earliest arrival and b_0 the boardings of a journey attaining it
 - 12: Set the selection arrival bound to a^* and boarding bound to b_0
 - 13: Obtain forward run admissibility; admit every run in an unscanned suffix
 - 14: Construct conservative reverse completion bounds through a^*
 - 15: **for** each boarding round up to b_0 **do**
 - 16: Extend retained complete states, close equal-time dependencies and apply dominance
 - 17: Evaluate the destination after completing the round, including all objective ties
 - 18: **end for**
 - 19: Select the minimum (a, b, w) and reconstruct its predecessor chain
-

Cost of bounding. An extra pass pays off when it saves more than it costs. Let T_{temporal} include any boarding-layer rerun and $T_{\text{additional bounds}}$ the forward/reverse bound construction. End-to-end time is then

$$T_{\text{two-stage}} = T_{\text{attachment}} + T_{\text{temporal}} + T_{\text{additional bounds}} + T_{\text{bounded selection}} + T_{\text{output}}.$$

Against unbounded journey selection with identical attachment and output work, the two-stage search is faster when

$$T_{\text{unbounded selection}} - T_{\text{bounded selection}} > T_{\text{temporal}} + T_{\text{additional bounds}}.$$

Reconstruction and output. During the search, each label stores only times and a pointer to the label it came from. Reconstruction turns the winning label back into a journey a person can follow: it walks these pointers from the destination back to the origin and recovers the sequence of walks and rides. Both runtimes then build timed legs with shared Rust components: transit lines follow the trip shape between the boarding and alighting stops, walking legs follow the street and station paths, and one component prices GTFS fares and writes the public result. Time-only output stops after temporal propagation and reports travel time for depart-at, including initial waiting, or departure lead time for arrive-by.

3.3 Compact state and reusable storage

Most connections the scan visits cannot be used, so the search is fast only if they are rejected cheaply and memory is not cleared between requests. The timetable scan keeps each connection’s source stop, vehicle run and permissions in an eight-byte record. Two bit sets, over reached stops and active runs, reject a connection whose run is inactive and whose source stop is unreached or forbids boarding. The common rejection thus reads one record and two bits, and arrival and journey fields sit in separate arrays that only surviving connections touch. Integer identifiers index labels directly, and each stop’s transfers are stored contiguously. Workspaces keep their memory across requests. A generation tag marks the entries written by the current request, so stale entries count as empty without being cleared, and lists of touched states limit other resets to visited entries. Street-search memory is allocated in pages on first write and reused.

3.4 Shared propagation and workload cost

Accessibility analyses ask for many origin–destination cells at once; shared propagation avoids repeating the same search for each cell. Each Route call starts a fresh search on the resident prepared network, reusing workspace storage and endpoint and shape caches. A Matrix call instead groups compatible cells. A *propagation group* shares initial labels, costs, transition rules and service in one search direction: depart-at groups share an origin and requested time, and arrive-by groups share a destination and deadline. One temporal propagation then serves the whole group, so a compatible m -by- n matrix needs m forward or n reverse propagations instead of mn , and each cell keeps its own temporal optimum. Proposition 3 in Appendix A gives sufficient conditions for this sharing.

For journey output, boarding rounds run forward from each origin or backward from each destination. The group’s temporal optima bound the search interval, and journey selection stops once every reachable destination has attained its own temporal optimum, since later rounds only add boardings. Each round reads a fixed set of labels from the previous round, so equal-time connections cannot use two boardings in one round. Large batches attach endpoints and write results in parallel without changing input order or results.

Shared propagation removes repeated timetable work, but writing Q cells and L leg or geometry records still costs $\Omega(Q + L)$, and a shared scan may cover a longer interval than a point scan with destination stopping. Its benefit is therefore largest for time-only output, such as accessibility counts, where per-cell reconstruction disappears.

Workload cost. For a whole analysis, costs add up as follows. Let P_0 be base preparation, U_k the k th service-day or network-update preparation, A endpoint attachment, S_g and J_g temporal propagation and journey selection for group g , E endpoint evaluation and Z output. Then

$$T_{\text{workload}} = P_0 + \sum_{k=1}^K U_k + A + \sum_{g=1}^G (S_g + J_g) + E + Z.$$

where $J_g = 0$ for time-only output and K counts actual preparations, including repeats. The standalone runtime keeps one active timetable and a bounded cache of inactive ones that share immutable columns. Cache capacity and request order therefore set the trade-off between memory and repeated preparation, and the number of distinct routing settings sets the number of propagation groups.

4. Computational Performance

Four questions guide the evaluation: how fast VIGO answers point requests and why engines disagree on which requests can be answered (Section 4.2); how fast it handles Matrix calls (Section 4.3); how much each search mechanism contributes (Section 4.4); and whether the results are correct (Section 4.5).

4.1 Setup

We evaluate VIGO 0.5.2 on six OSM/GTFS networks ([Haklay and Weber, 2008](#); [General Transit Feed Specification, 2026](#)), from a small one (Anchorage) to a large one (New York), on one eight-core, 16 GiB arm64 desktop (Table 1). Network counts distinguish all-calendar preparation from the connections, trips and transfer edges active on the service date. Each city has 3,000 origin–destination (OD) pairs, divided equally among random street, near-stop and stop-offset points. The strata vary how far endpoints lie from transit: random street points often have no stop within walking range, near-stop points start close to service, and stop-offset points require a longer walk. The workload is deliberately not filtered for reachability, because accessibility matrices contain unreachable cells and an engine must also establish quickly that no journey exists.

Two measurement campaigns supply the results. Depart-at point performance, resource use and runtime agreement use this release through the Node.js command-line interface (CLI) and VIGO-py. Matrix, arrive-by, ablation and enumeration results come from the preceding release, which implements the same objective, search and bounds; the ablations are ratios measured within that release. On the depart-at workload that both releases ran, VIGO’s median end-to-end time was equal or lower with the newer release in every city (Appendix B); no such comparison exists for the other experiments.

End-to-end time starts before request serialization and ends after transport, decoding and adapter normalization. An eligible journey satisfies the benchmark’s walking, boarding and duration limits, and an interface *answers* a request when it returns an eligible journey. A *blocked* outcome is either a no-route response or a *policy rejection*, a journey that violates these limits. Query errors are counted separately.

Nine *interfaces* enter the point comparison: VIGO; the routing systems of Section 2, R5, OTP, MOTIS, Valhalla and GraphHopper; ULTRA-OSM, a coordinate interface around the ULTRA-RAPTOR implementation; and the Python wrappers VIGO-py and R5py. Each engine uses its own routing model and settings; R5, for example, adds 60 s of boarding slack where VIGO adds none (Appendix B). The other engines were measured once, in earlier sessions on the same workload, and were not rerun. The *matched cohort* contains the requests that every interface answers with a timed journey: between 860 and 1,298 requests per city, or 29% to 43% of the workload. Two things shrink it. Random street points often lie outside transit coverage

or farther than the 1.2 km walking limit from a useful stop, so VIGO finds eligible journeys for only 104 to 769 of the 1,000 random-street pairs per city, against 774 to 975 of the near-stop pairs (Table 9). And a request stays in the cohort only if all nine interfaces answer it. The matched cohort therefore measures how fast each interface answers identical requests, and the coverage table measures how often it answers. Speed rankings below apply to the cohort, not to the full request distribution.

Table 1: Prepared networks.

	Anchorage	Denver	Las Vegas	Boston	Washington, DC	New York
Stops	682	7,624	3,900	10,306	9,695	15,920
Prepared connections	97,834	785,817	1,367,058	2,449,494	7,464,386	7,749,151
Service-date connections	33,495	201,931	151,566	416,087	442,734	1,267,703
Service-date trips	847	5,471	3,122	16,187	11,587	40,356
Transfer edges	4,188	47,960	18,479	83,192	70,770	235,688
Street nodes (M)	1.56	14.00	5.42	5.76	14.93	9.51

4.2 Point requests

On all eligible requests, VIGO’s median end-to-end time ranges from 0.67 ms in Anchorage to 2.43 ms in New York, and its 99th percentile from 1.67 ms to 9.23 ms (Table 2). Arrive-by requests on the same OD pairs with a noon deadline have eligible medians between 0.80 ms in Anchorage and 5.92 ms in New York.

Table 2: Depart-at end-to-end time (ms).

	All p50	Eligible p50	Eligible p99	Blocked p50
Anchorage	0.59	0.67	1.67	0.41
Denver	1.08	1.48	4.76	0.48
Las Vegas	0.84	0.96	2.20	0.44
Boston	1.42	1.85	6.20	0.48
Washington, DC	1.38	1.85	5.45	0.48
New York	2.19	2.43	9.23	0.68

All columns exclude each process’s first response. Blocked: no route or policy rejection.

On the matched cohort, VIGO has the lowest median end-to-end time in every city, from 0.67 ms in Anchorage to 2.11 ms in New York; through VIGO-py, the same requests take 1.24 ms to 2.99 ms (Table 3). Panel (c) compares journey times, final arrival minus the requested departure, with VIGO’s: VIGO-py reproduces VIGO, and R5py reproduces R5. The largest disagreement is MOTIS in New York, with a median absolute difference of 8.12 minutes and a correlation of 0.912. The ranking does not depend on the random street points: on the near-stop pairs of the cohort alone, VIGO’s median ranges from 0.66 ms in Anchorage to 1.98 ms in New York and is the lowest in every city, as it is on the stop-offset pairs. Among the cohort’s random-street pairs, ULTRA-OSM is slightly faster in Anchorage, at 0.73 against 0.82 ms over 64 pairs.

Where the time goes. Timetable search, temporal propagation plus journey selection, takes 7.7% of summed end-to-end time in Anchorage and 56.7% in New York (Table 4); endpoint attachment, result conversion, journey assembly, serialization and transport take the rest. Eligible New York requests inspect a median of 67,174 connections, 5.3% of the day’s connections, against 1,235 (3.7%) in Anchorage; the count includes repeat inspections across both stages.

Resources. Table 5 reports equal-city geometric means. VIGO uses less of all four resources than ULTRA-OSM. MOTIS prepares faster and uses less memory and storage, while VIGO answers matched-cohort

Table 3: End-to-end time and journey-time agreement with VIGO.

Interface Matched ODs <i>n</i>	Anchorage 1,258	Denver 860	Las Vegas 1,298	Boston 1,113	Washington, DC 869	New York 1,109
(a) Median end-to-end time (ms)						
VIGO	0.67	1.32	0.94	1.69	1.64	2.11
ULTRA-OSM	0.74	18.65	8.49	17.55	19.15	70.56
MOTIS	3.16	12.70	19.88	13.85	28.52	19.00
R5	8.63	30.33	22.34	33.66	42.24	75.67
Valhalla	12.18	131.35	82.20	80.88	129.53	221.72
OTP	6.92	21.36	21.03	35.46	37.45	137.12
GraphHopper	7.62	56.81	50.06	145.74	215.26	1,041.11
<i>Python wrappers</i>						
VIGO-py	1.24	2.42	1.66	2.58	2.95	2.99
R5py	13.89	36.87	27.77	40.18	49.51	86.28
(b) 99th-percentile end-to-end time (ms)						
VIGO	1.35	4.29	2.20	4.21	4.16	6.29
ULTRA-OSM	1.94	109.80	38.66	97.00	103.60	395.40
MOTIS	5.84	29.10	36.06	29.84	55.77	37.99
R5	41.08	146.35	84.16	129.29	189.37	538.16
Valhalla	25.20	371.06	203.04	227.47	443.70	870.71
OTP	20.87	421.15	389.75	298.31	466.94	1,692.90
GraphHopper	34.90	262.79	263.59	739.43	1,554.58	18,074.27
<i>Python wrappers</i>						
VIGO-py	2.47	5.87	3.13	5.90	8.78	8.02
R5py	48.27	163.77	82.91	206.46	220.58	502.16
(c) Median absolute journey-time difference (min) / Pearson correlation						
VIGO	0.00 / 1.000	0.00 / 1.000	0.00 / 1.000	0.00 / 1.000	0.00 / 1.000	0.00 / 1.000
ULTRA-OSM	0.42 / 0.968	0.55 / 0.979	0.57 / 0.970	0.93 / 0.955	1.00 / 0.958	0.48 / 0.980
MOTIS	0.53 / 0.962	0.85 / 0.940	0.90 / 0.959	1.38 / 0.933	1.17 / 0.948	8.12 / 0.912
R5	0.68 / 0.925	0.58 / 0.966	0.58 / 0.964	0.90 / 0.952	0.97 / 0.961	0.80 / 0.974
Valhalla	0.63 / 0.967	0.72 / 0.976	1.29 / 0.932	1.29 / 0.937	1.05 / 0.961	0.64 / 0.974
OTP	0.22 / 0.977	0.27 / 0.981	0.25 / 0.980	0.32 / 0.976	0.35 / 0.987	2.35 / 0.974
GraphHopper	0.40 / 0.944	0.46 / 0.931	0.65 / 0.924	0.52 / 0.953	1.49 / 0.942	0.35 / 0.978
<i>Python wrappers</i>						
VIGO-py	0.00 / 1.000	0.00 / 1.000	0.00 / 1.000	0.00 / 1.000	0.00 / 1.000	0.00 / 1.000
R5py	0.68 / 0.925	0.58 / 0.966	0.58 / 0.964	0.90 / 0.952	0.97 / 0.961	0.80 / 0.974

All panels use the matched cohort; agreement is relative to VIGO.

Table 4: Query stages and scan work.

	Anchorage	Denver	Las Vegas	Boston	Washington, DC	New York
Eligible requests	1,832	1,651	2,123	1,830	1,738	2,458
Median connection inspections	1,235	15,374	9,048	60,889	27,353	67,174
<i>Median time (ms)</i>						
Endpoint attachment	0.053	0.068	0.050	0.074	0.075	0.109
Timetable search	0.051	0.353	0.202	0.742	0.540	1.269
Journey assembly	0.091	0.171	0.127	0.201	0.234	0.212
<i>Share of summed end-to-end time (%)</i>						
Timetable search	7.7	28.2	22.0	48.1	33.5	56.7

All eligible VIGO requests except each process's first response, a larger sample than the matched cohort. Shares divide summed stage time by summed end-to-end time.

requests 10.4 times faster; R5 and OTP use less storage. A configuration is Pareto-dominated when another has no higher cost on any axis and a lower cost on at least one. VIGO, MOTIS, R5, OTP and the two Python wrappers are nondominated, MOTIS dominates Valhalla and GraphHopper, and VIGO has the lowest end-to-end time among the nondominated configurations.

Coverage and the New York audit. VIGO returns fewer eligible journeys than ULTRA-OSM, R5 and OTP in every city (Table 9). In New York, 216 requests are eligible in R5 but blocked in VIGO. In 213 of them VIGO finds no legal walk to transit at one end: in 184 the endpoint cannot be connected to a walkable street that reaches a stop, and in 29 the nearest reachable stop lies beyond the 1.2 km limit. The other three find no timetable path. To see which engine is right, we audited these 216 requests against the OSM extract both engines use (Appendix B). Every stop R5 boards at these endpoints exists in VIGO’s network, so the disagreement is about walking, not transit. In 187 requests the source OSM or GTFS data do not support R5’s returned journey: the endpoint lies more than 160 m from any publicly walkable way; R5’s walk crosses ways or gates closed to pedestrians; R5 moves the endpoint to its street network without counting that distance, and counting it would take the walk past 1.2 km; or, once, R5 boards a bus where the feed forbids pickup. In 9 requests VIGO is too strict, rejecting paths whose posted opening hours include the requested time. The remaining 20 are unresolved: VIGO’s legal walk is longer than R5’s and no single rule explains why.

Table 5: Equal-city geometric means of end-to-end time and resource use.

Interface	Median end-to-end time (ms)	Preparation (s)	Query RSS (MiB)	Disk (MiB)
VIGO	1.30	58.0	669.5	663.0
ULTRA-OSM	11.85	1,356.4	2,099.6	1,615.6
MOTIS	13.47	17.5	231.9	531.5
R5	29.27	45.5	2,802.2	409.3
Valhalla	82.07	234.3	371.8	903.5
OTP	28.76	91.2	3,057.3	141.9
GraphHopper	94.39	30.4	2,780.1	774.4
VIGO-py	2.20	60.0	652.2	663.3
R5py	36.70	54.8	2,626.5	411.5

End-to-end time aggregates the medians of Table 3; RSS is the peak memory of the query process.

4.3 Matrix calls

A *batch* is one Matrix call from an origin to many destinations. Each city contributes six fixed origins, drawn equally from the OD strata, and nested sets of 1, 10, 100 and 1,000 destinations; the benchmark compares time-only output, timed legs and journeys with geometry on the same cells. At 1,000 destinations, median batch times with timed legs stay below 160 ms in every city, about 10 times faster than the sum of individual 1-by-1 Matrix calls in Anchorage and 61 times faster in New York (Table 6, panel A). Geometry narrows the gain, and against individual Route calls the city medians range from 0.85 in Washington to 1.75 in New York (panel B): Route stops propagation once later connections cannot improve the arrival, which leaves less repeated work to share. In Washington, individual Route calls are faster for 4 of the 6 journey-producing origins.

Output size separates the origins in Figure 2. The 29 origins that produce journeys return 11 to 38 MB per batch, and their Route/batch ratios stay below 2. The seven origins whose cells are all blocked return about 0.3 MB, and for them the batch is 8 to 11 times faster than individual Route calls.

Table 6: End-to-end time for 1,000 destinations.

<i>A. Batch versus individual Matrix calls</i>					
City (<i>n</i>)	Eligible / 6,000 pairs	Timed legs		Legs with geometry	
		Batch ms	Individual / batch time	Batch ms	Individual / batch time
Anchorage (4)	2,933	35.1	9.95	421.6	1.54
Denver (3)	2,160	60.7	29.38	1168.4	2.08
Las Vegas (6)	5,026	43.4	21.38	762.5	1.78
Boston (4)	2,989	67.1	53.26	1193.0	3.55
Washington, DC (6)	4,231	80.4	41.27	1978.6	2.11
New York (6)	5,422	151.7	61.33	1574.4	6.30

<i>B. Journeys with geometry: batch versus individual Route calls</i>					
City	<i>n</i>	Batch ms	Individual Route ms	Route / batch time	Tied ride alternatives
Anchorage	4	428.2	585.8	1.35	0
Denver	3	1158.9	1260.0	1.10	3
Las Vegas	6	736.7	888.9	1.23	0
Boston	4	1152.8	1758.9	1.50	3
Washington, DC	6	1932.0	1626.7	0.85	0
New York	6	1736.0	2941.9	1.75	4

n: origins that return journeys; coverage counts all 6,000 pairs.

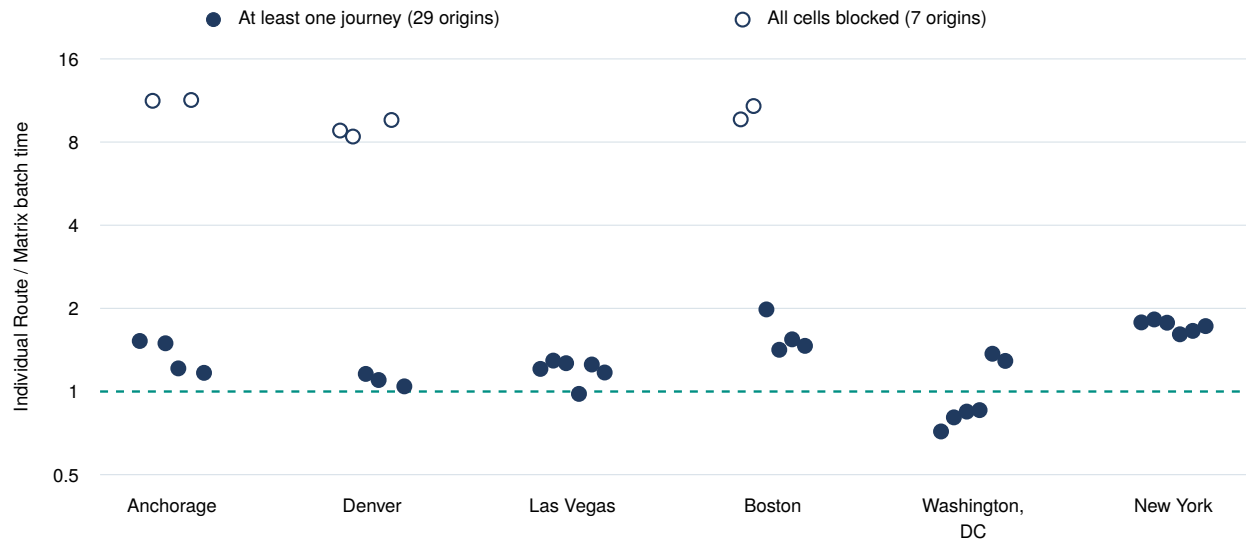


Figure 2: Route time divided by batch time for each of the 36 origins (1,000 destinations, with geometry); values above one favor the batch.

4.4 Mechanism ablations

Each ablation runs the same OD pairs with one mechanism enabled and disabled and compares the paired results (Table 7; protocols in Appendix B).

Selection bounds. On its own, the arrival bound produces almost as many selection labels as unbounded journey selection: it limits completion time but keeps partial journeys that arrive early yet cannot usefully continue. The boarding bound removes surplus rounds, and forward/reverse bounds remove runs and stop states that cannot reach the destination in time. With all bounds, journey selection creates between 0.06% (New York) and 0.60% (Anchorage) of the labels of unbounded journey selection, and the two-stage search is 4.5 (Boston) to 8.1 (New York) times faster, even after paying for the extra temporal propagation (panel A).

Stopping, arrive-by bounds and shared propagation. Destination stopping reduces connection inspections and two-stage kernel time in every city (Table 7, panel B). Without forward/reverse bounds, arrive-by journey selection creates more labels and takes longer in every city. Split into eight calls, a 64-by-64 depart-at matrix needs 64 forward propagations when each call holds a block of origins, but 512 when each call holds a block of destinations, because every call repeats all 64 origin propagations; arrive-by is symmetric. The measured speedups, about 4 in Anchorage and between 6.2 and 7.6 in the other cities, stay below this eight-fold difference in propagation count.

Table 7: Mechanism ablations.

A. Depart-at selection bounds						
Network	Requests	Unbounded / two-stage time	Labels (% of unbounded selection)			
			Arrival only	Arrival + boarding limit	Arrival + forward/reverse	All
Anchorage	182	6.25	99.07	34.10	4.15	0.60
Denver	168	5.92	96.40	52.70	1.04	0.25
Las Vegas	209	6.09	97.53	32.96	1.37	0.18
Boston	176	4.53	96.64	52.07	0.60	0.18
Washington, DC	172	6.56	95.86	52.24	0.44	0.16
New York	245	8.10	96.73	51.92	0.21	0.06

B. Destination stopping, arrive-by bounds and shared propagation						
Network	Destination stopping disabled		Arrive-by bounds disabled		Split / shared propagation	
	Inspections	Kernel time	Labels	Kernel time	Depart-at	Arrive-by
Anchorage	6.56	3.05	29.9	3.22	4.21	4.34
Denver	2.75	2.11	140.1	11.82	6.52	6.56
Las Vegas	3.31	2.50	123.7	7.39	6.46	6.22
Boston	2.54	3.18	146.4	10.84	7.13	7.35
Washington, DC	3.50	2.78	217.8	11.35	7.09	7.11
New York	3.82	2.77	462.3	15.32	7.55	7.06

Panel A: medians over 1,152 OD pairs. Panel B: disabled/enabled or split/shared ratios.

4.5 Correctness and consistency

Correctness is checked at three levels: optimality of the timetable search on small generated networks, consistency of returned rides with the source GTFS schedules on the six city networks, and agreement between VIGO's two runtimes, VIGO-py and Matrix calls. The first isolates the routing optimization from the scan and bound implementation. None of the three verifies the walking model or proves optimality on

the city networks; the New York audit in Section 4.2 checks walking access against the OSM data for the disputed requests.

Optimality. Independent whole-ride enumeration, the *oracle*, agrees with VIGO on all 2,400 depart-at and arrive-by cases across 150 generated networks of six stops and ten trips each: 1,684 feasible optima and 716 infeasible cases. Unbounded journey selection also matches all 1,200 depart-at cases. The oracle enumerates rides directly from raw trips, without connection scans, dominance or bounds, and checks the temporal optimum, boardings and walking, including permissions, directed transfers, same-stop minima, repeated run positions and equal-time dependencies. Its endpoint alternatives are synthetic, so it tests timetable optimization given endpoint costs.

Schedules. Source replay re-checks every returned ride leg against the original GTFS schedule. Across the 23,270 eligible journeys of the preceding release, 11,635 each for depart-at and arrive-by, all 59,428 ride legs match active GTFS trips, stop sequences, scheduled times and permissions.

Agreement between runtimes. Three comparisons check that the two runtimes, VIGO-py and Matrix calls return the same results. The standalone Rust and Node.js runtimes agree on outcomes and ordered quantities, to the reported time precision, for all 36,000 depart-at and arrive-by requests of the preceding release, and return byte-identical public results, apart from the measured compute time, for all 18,000 depart-at requests of the evaluated release. VIGO and VIGO-py agree on outcome and journey for all 18,000 depart-at requests. Within the Matrix experiment, batches and individual Matrix calls return the same journey content, and Route keeps the ordered quantities but selects different rides for 10 tied OD pairs, 3 each in Denver and Boston and 4 in New York (Table 6).

5. Discussion

VIGO's results rest on one design choice: settle the temporal optimum cheaply, then spend multicriteria work only where it can change the answer. The ablations show that an arrival bound by itself is not enough. The boarding and forward/reverse bounds remove the partial journeys that would otherwise fill journey selection, and the saving exceeds the cost of the extra temporal propagation in every city.

Once search is cheap, the remaining cost moves elsewhere. Timetable search accounts for 56.7% of summed end-to-end time in New York but 7.7% in Anchorage, so in small networks endpoint attachment and output dominate. Matrix calls show the same shift: shared propagation makes timed-leg batches 10 to 61 times faster than individual calls, but with geometry the output becomes the bottleneck, and individual Route calls can win. An engine for accessibility work should therefore be judged on end-to-end time for the output an analysis needs, not on timetable search alone. Accessibility counts need only travel times, so time-only Matrix output keeps the full benefit of shared propagation.

Speed and coverage have to be read together. VIGO is fastest on the requests that every interface answers, but it answers fewer requests overall because its walking-access test is stricter. In New York the OSM data support that stricter decision in most disputed requests, though not in all of them. For accessibility work this matters more than a millisecond: an origin whose walk to transit is judged infeasible contributes no opportunities, so the access model shapes the measured result as much as the timetable search does.

Each engine occupies a different point of the resource trade-off. VIGO has the lowest per-request time, MOTIS prepares faster and uses less memory, and R5 and OTP need less storage.

5.1 Limitations and research directions

First, the evidence comes from two measurement campaigns: point performance from the evaluated release, and Matrix, arrive-by and ablation results from its predecessor. The releases share the objective, search and bounds, and the newer one was no slower on the depart-at workload, but only a single campaign on one release would settle the question for the Matrix, arrive-by and ablation results.

Second, correctness is established for the timetable search. The oracle covers 150 small generated networks with synthetic endpoints, and source replay confirms schedules on the city networks. Walking access is checked against the OSM data, not on the ground. In the New York audit, VIGO's stricter access test agrees with OSM in most disputed requests, but it wrongly rejects paths with opening hours, and 20 cases remain unresolved. Interpreting time-conditional access and surveying unresolved cases in the field are the next steps.

Third, the workloads are narrow: six U.S. networks, four of them on Sunday service, and sequential requests on one desktop. Weekday service, concurrent requests, realtime updates and cache behavior under mixed dates and routing settings remain to be measured.

Fourth, the evaluation covers one ordered objective. VIGO can also return up to nine Pareto or ranked alternatives at a fixed time, and full departure windows and many-to-many workloads would test how far shared propagation extends; neither is evaluated here.

6. Conclusions

Disaggregate accessibility analysis repeats transit routing across many locations and departure times, so its cost is set by complete coordinate-to-coordinate requests, of which timetable search is one part. VIGO addresses this cost with two-stage search, compact search state and shared propagation. Fixing the temporal optimum first lets the boarding and forward/reverse bounds cut selection labels to under 1% of unbounded journey selection and make the two-stage search 4.5 to 8.1 times faster, and independent enumeration agrees on all 2,400 tested cases. On the matched cohort, median end-to-end times range from 0.67 ms to 2.11 ms, the lowest among the nondominated configurations of the resource comparison. VIGO answers fewer requests than some engines, and where it and R5 disagree in New York, the source OSM and GTFS data support VIGO's decision in 187 of 216 requests.

Analyses that need only travel times can use time-only Matrix output, which keeps the benefit of shared propagation that journey output loses to reconstruction and geometry. Further gains lie in endpoint attachment and output for small networks, and in output size for large Matrix calls. Extending the evaluation to weekday and concurrent workloads, departure windows and many-to-many accessibility tasks would test how far these conclusions carry.

Appendix A. Search arguments and model parameters

Under the conditions below, the shortcuts of Section 3 return the same result as an unrestricted search.

Model and complete state. Fix a finite prepared timetable for the duration of a query. Movement costs are nonnegative, waiting is permitted, and temporal propagation and journey selection use the same endpoint alternatives and transition rules. A routing state x contains the stop and the flags that determine legal continuation: prior-ride status, egress permission and boarding readiness. An onboard state also records the vehicle run and position. Boarding count is a separate layer. Labels may be merged only when their states

permit the same onward actions. All optima are relative to the prepared network, its endpoint attachments and its transfer edges.

Proposition 1 (temporal optimum under the boarding limit). Time-ordered propagation with complete state and boarding layers returns the temporal optimum among journeys with at most B boardings. *Proof.* Induct over departure-time groups, taking the fixed point of all admitted equal-time transitions before advancing to the next group. Access initializes feasible states. A boarding moves from layer $b - 1$ to b ; continuation on the same legal run stays in that layer. Within a complete state and layer, an earlier arrival can wait for every departure available to a later arrival, so retaining the earliest time preserves every possible temporal improvement. Equal-time closure is necessary because a connection encountered later in a group can make an earlier connection in that group boardable. Endpoint evaluation takes the minimum over admitted egress alternatives. Reverse propagation uses the corresponding latest feasible state times and reversed legal transitions. $\square$

Proposition 1 thus justifies temporal propagation: keeping a single time per routing state and boarding layer loses no earliest arrival.

This argument also justifies the boarding-limit shortcut of Section 3.2. Without the limit, the depart-at optimum is a lower bound on the limited one, and a journey attaining it within B boardings proves equality; otherwise layered propagation must start with an infinite arrival bound. Arrive-by uses the dual argument, accepting an unrestricted latest departure only after a feasible journey from it meets the limit and deadline. Destination stopping must bound every remaining completion, including endpoint walking, and keep equal-arrival candidates until the later criteria are resolved.

Lemma 1 (conservative bound construction). A forward or reverse relaxation that retains every feasible complete-state transition, initial alternative and terminal alternative supplies admissible bounds. A partially computed reachability restriction remains conservative if all unexamined transitions are admitted. *Proof.* Every journey in the original problem remains a journey in the relaxation, so its earliest time can only decrease and its latest feasible completion-start time can only increase. Replacing unknown reachability by admission enlarges this set and preserves the inequalities. For bounds by remaining boarding count, apply the same argument at each layer. $\square$

Lemma 1 thus shows when a bound is safe to use: a bound computed on a relaxed version of the problem never discards a feasible journey.

In VIGO, a temporal propagation without the boarding limit admits a superset of the runs reachable with it, and after destination stopping every run in the unscanned interval up to the arrival bound stays admitted. A recorded first boarding position may restrict a run only if continuing from it covers all onward actions it replaces, which requires complete run identity and equal-time closure. Reverse bounds keep every legal completion of the admitted forward states, with deadlines rounded outward.

Proposition 2 (journey selection within admissible bounds). Let $F_b(x)$ be the earliest time at routing state x with at most b boardings, and $D_{B-b}(x; U)$ the latest time from which completion by arrival bound U is possible with at most $B - b$ further boardings. Bounds are admissible when $\underline{F}_b \leq F_b$ and $\overline{D}_{B-b} \geq D_{B-b}$. A prefix, or partial journey, reaching x at time t after b boardings can complete by U within B boardings only if

$$\underline{F}_b(x) \leq F_b(x) \leq t \leq D_{B-b}(x; U) \leq \overline{D}_{B-b}(x; U).$$

Membership in this interval is necessary, not sufficient, for a feasible completion. *Proof.* A prefix outside this interval cannot complete within the time and boarding limits. In the same complete state and boarding layer, a dominating prefix arrives no later, has no greater walking time and can take every continuation of the dominated prefix, so substituting it cannot worsen the ordered objective. At a completed destination,

pruning uses a lower bound on the entire remaining ordered tuple and keeps a candidate whenever an unresolved tie could improve a later criterion. A feasible depart-at journey at a^* with b_0 boardings excludes all candidates with more than b_0 boardings; it does not exclude equal-boarding candidates with less walking. For arrive-by, fix d^* , retain all completions by the original deadline and minimize (b, w, a) . Completing the winning boarding round is necessary to resolve its walking and arrival ties. $\square$

Proposition 2 thus justifies bounded journey selection: with safe bounds and complete-state dominance, journey selection still returns the ordered optimum.

Proposition 3 (sufficient conditions for sharing). Suppose requests have identical initial labels, complete-state transition rules, costs, boarding limit and search coverage in one direction, while terminal evaluation may differ by destination. A shared temporal propagation covering all destinations returns the same temporal values as independent propagations. Shared journey selection also preserves each destination’s ordered optimum if its retained states cover every destination’s feasible journey-selection region and dominance is valid for every terminal evaluation. *Proof.* The propagations have the same initial state and update operator, so induction over closed time groups gives the same labels, and applying each destination’s terminal evaluation gives its independent value. For journey selection, retain the union of the destination-specific feasible regions. In a forward group, the latest finite arrival encloses every destination’s time interval; in a reverse group, the earliest finite departure does so. Complete-state dominance from Proposition 2 preserves each destination’s optimum within this union, and reverse journey selection must also retain actual destination arrival for the final tie. Once a complete boarding round has supplied every reachable destination with its temporal optimum, any later round adds a boarding and cannot improve an ordered result. Stopping at the first successful label within that round would not have this guarantee. $\square$

Proposition 3 thus justifies shared propagation in Matrix calls: one search can serve many destinations without changing any destination’s result.

Search completion. All propositions assume completed searches that meet the state, closure and bound conditions above. The kernel caps journey-selection label insertions at two million and reports resource exhaustion at that limit. A journey attaining the temporal optimum certifies that value; the full ordered optimum and a no-route result both require a completed search. Benchmark policy rejection is applied separately, and Section 4.5 reports the implementation checks.

Model parameters. Table 8 lists the routing parameters that define a feasible walk in every experiment. Station paths use published traversal times where available, and otherwise one second per stair or five seconds per gate, with a floor set by path length.

Table 8: Routing parameters.

Parameter	Implemented meaning
Endpoint attachment	Vertices within 160 m and reciprocal-edge projections within 80 m. Connector cost charged
Endpoint walking limit	Uses represented distance, including station paths; unknown lengths contribute lower bounds. CCH stop buckets up to 5 km, original-graph search above it
Intermediate OSM walks	Default 500 m stop-to-stop radius, supplemented by GTFS transfers and pathways
Parent-station links	Published minimum transfer times set a lower bound on covered platform-edge durations; unspecified links use at least 120 s or modeled walking time
Search horizon	Admits scheduled boarding and alighting events. Depart-at egress may finish later; arrive-by egress must meet the deadline
Complete-duration ceiling	The point benchmark rejects returned journeys over 240 min, independently of the search horizon
Direct-walking comparison	When permitted, walking wins a primary-time tie in either direction

Appendix B. Experimental protocols

Workload. Easy and hard access conditions are both represented. Each city has 1,000 random street pairs, 1,000 pairs within 400 m of a stop and 1,000 pairs 800 to 1,200 m from stops, drawn without reachability filtering so that blocked requests are measured too. Requests require transit, with a 1.2 km endpoint walking limit, a walking speed of 4.8 km/h, at most four boardings and a 240-minute duration limit; depart-at requests leave at 08:00, and arrive-by requests use a noon deadline. Service dates are 2026-07-14 (Anchorage, Tuesday), 2026-06-07 (Denver, Sunday), 2025-10-12 (Las Vegas, Sunday), 2026-07-06 (Boston, Monday), 2025-06-29 (Washington, Sunday) and 2026-09-06 (New York, Sunday). One date and time per city gives every interface the same requests.

Execution and measures. Timing mirrors analysis use: one resident process handles requests in turn. Measurements run on an eight-core arm64 desktop with 16 GiB RAM and Node.js 26.7.0. VIGO and VIGO-py each build a fresh prepared network and run the 3,000 depart-at OD pairs once per city, sequentially, with endpoint and street-path caches enabled and no cache of complete results. End-to-end time excludes offline preparation. Point tables drop each process's first response, which includes one-time loading; in Anchorage, Washington and New York that response is eligible, so Table 4 counts 11,632 eligible requests against 11,635 in Table 9. Quantiles use the nearest rank $x_{(\lceil pn \rceil)}$, so each is an observed time, and journey agreement compares final arrival minus 08:00 with VIGO's by median absolute difference and Pearson correlation. Kernel timers cover endpoint attachment and timetable search, and journey assembly includes geometry. Preparation is full-build wall time, memory is the peak resident set size (RSS) of the query process, and disk is the size of the prepared working directory. Aggregates are equal-city geometric means, so each city counts equally; no outlier filter is applied.

Interfaces and releases. Each engine keeps its own model and settings. ULTRA-OSM exposes ULTRA-RAPTOR through a coordinate interface on VIGO's exported walking graph and returns straight segments between stops. R5 routes at fixed departures with 60 s of boarding slack, and R5py also builds its own itinerary and GeoPandas objects. Adapters normalize the formats of engines queried over HTTP; MOTIS returns one candidate journey, which is checked against the benchmark limits without requesting another. Valhalla journey times for Washington and New York come from a replay with corrected service calendars, whose timings are not used. Depart-at point measurements, resources and VIGO-py agreement come from the evaluated release, and all other VIGO results from the preceding release. On the shared depart-at workload, VIGO's median over all responses was 0.597, 1.232, 0.865, 1.661, 1.444 and 2.682 ms with the preceding release and 0.595, 1.079, 0.839, 1.421, 1.376 and 2.185 ms with the evaluated release, in the city order of Table 1.

Ablations. Each ablation replays identical inputs and changes one switch. The bound ablation uses stored timetables and endpoint attachments for 1,152 feasible OD pairs, queried depart-at at 08:00 with a noon horizon and four boardings. After one warm-up, five measured passes run in rotating order so that caching and ordering affect every mode equally, and every mode starts from the same untimed temporal propagation; the two-stage time includes its own propagation, and unbounded journey selection uses the horizon plus the longest egress time as its arrival bound. Ratios are medians of paired per-request medians, which compare each request with itself, and label counts measure insertions, not peak memory. The stopping ablation adds a runtime switch for destination stopping and excludes attachment and output; the arrive-by ablation compares the arrival bound alone with forward/reverse bounds; and the shared-propagation ablation splits a 64-by-64 matrix into eight calls by origin or by destination, with caches disabled, reporting the median ratio over three endpoint sets. Every ablation preserves arrival, boardings and walking time.

Table 9: Full-workload coverage: eligible journeys by interface and OD stratum.

Interface / VIGO stratum	Anchorage	Denver	Las Vegas	Boston	Washington, DC	New York
<i>Each interface: 3,000 requests per city</i>						
VIGO	1,833	1,651	2,123	1,830	1,739	2,459
ULTRA-OSM	1,896	1,727	2,227	1,997	1,795	2,737
MOTIS	1,794	1,603	2,136	1,867	1,581	1,475
R5	1,865	1,679	2,206	1,984	1,782	2,666
Valhalla	1,467	1,072	1,569	1,351	1,265	2,285
OTP	1,879	1,677	2,224	1,980	1,795	2,666
GraphHopper	1,621	1,342	1,937	1,647	1,456	2,401
<i>VIGO by stratum: 1,000 requests per cell</i>						
Random street	104	243	377	217	156	769
Near-stop	975	774	951	901	870	923
Stop-offset	754	634	795	712	713	767

One query error (GraphHopper, New York); all other ineligible outcomes are blocked.

Matrix experiments. To isolate output cost, each city starts from a fresh resident process with caches disabled and uses six origins, two per OD stratum, with destinations drawn across the strata. Batches, individual 1-by-1 Matrix calls and individual Route calls use the same output mode and run one warm-up and three measured passes in rotating order; each timed request covers all destinations, blocked cells included. City times are medians of per-origin medians, and ratios are medians of per-origin ratios over origins that return a journey. Tied alternatives count OD pairs where Route and the batch select different rides with equal ordered quantities.

Correctness checks and disagreement audit. Small generated networks let the oracle enumerate every journey; on each, it checks both travel directions with boarding limits of one to four and two time windows. The disagreement audit asks, for every New York request that R5 answers and VIGO blocks, whether the source data support R5’s journey, comparing both engines’ recorded responses with the benchmark’s OSM extract. A way counts as publicly walkable unless it is a motorway or trunk without foot access or carries access or foot restrictions such as private or no; restricted gates and other nodes are checked separately. R5’s walking geometry is matched to OSM ways within 8 m to allow for small geometric differences, its unreported endpoint move is the distance from the requested coordinate to the start of its walk, and GTFS boarding permissions come from the source feed. Two of VIGO’s three no-path requests succeed with a 1.5 km walking limit.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

We used OpenAI Codex, Cursor’s coding agent and Anthropic’s Claude Code to review source material, write the analysis scripts that derive measurements from recorded responses, prepare the architecture illustration and edit the manuscript. Every reported value is computed by these scripts from retained records, and an audit script checks the tables against the same records. We reviewed and edited the tools’ output and take full responsibility for the article’s content.

Data and Code Availability

Engine source is available at <https://github.com/hytangs/vigo>; the evaluated release is tag v0.5.2. The benchmark harness, engine adapters and measurement methodology are available at <https://github.com/hytangs/vigo-bench> (commit 3572d87). City inputs, prepared networks and retained responses are not part of that repository and are held by the authors. The whole-ride oracle uses generated inputs independent of the city datasets.

References

- Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. In *Algorithm Engineering*, LNCS 9220, pp. 19–80, 2016. https://doi.org/10.1007/978-3-319-49487-6_2.
- Moritz Baum, Valentin Buchhold, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. UnLimited TRAnsfers for multi-modal route planning: An efficient solution. In *Proceedings of ESA 2019, LIPIcs 144*, Article 14, 2019. <https://doi.org/10.4230/LIPIcs.ESA.2019.14>.
- Moritz Baum, Valentin Buchhold, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. ULTRA: Unlimited transfers for efficient multimodal journey planning. *Transportation Science*, 57(6), pp. 1536–1559, 2023. <https://doi.org/10.1287/trsc.2022.0198>.
- Matthew Wigginton Conway, Andrew Byrd, and Marco van der Linden. Evidence-based transit and land use sketch planning using interactive accessibility methods on combined schedule and headway-based networks. *Transportation Research Record*, 2653(1), pp. 45–53, 2017. <https://doi.org/10.3141/2653-06>.
- Matthew Wigginton Conway, Andrew Byrd, and Michael van Eggermond. Accounting for uncertainty and variation in accessibility metrics for public transport sketch planning. *Journal of Transport and Land Use*, 11(1), pp. 541–558, 2018. <https://doi.org/10.5198/jtlu.2018.1074>.
- Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-based public transit routing. In *Proceedings of ALENEX 2012*, pp. 130–140, 2012. https://www.microsoft.com/en-us/research/wp-content/uploads/2012/01/raptor_alenex.pdf.
- Daniel Delling, Thomas Pajor, and Renato F. Werneck. Round-based public transit routing. *Transportation Science*, 49(3), pp. 591–604, 2015. <https://doi.org/10.1287/trsc.2014.0534>.
- Daniel Delling, Julian Dibbelt, and Thomas Pajor. Fast and exact public transit routing with restricted Pareto sets. In *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, pp. 54–65, 2019. <https://doi.org/10.1137/1.9781611975499.5>.
- Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 21, Article 1.5, 2016. <https://doi.org/10.1145/2886843>.
- Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Connection Scan Algorithm. *ACM Journal of Experimental Algorithmics*, 23, Article 1.7, 2018. <https://doi.org/10.1145/3274661>.
- Christoph Fink, Willem Klumpenhouwer, Marcus Saraiva, Rafael H. M. Pereira, and Henrikki Tenkanen. r5py: Rapid realistic routing with R5 in Python. Software release, 2022. <https://doi.org/10.5281/zenodo.7060438>.
- General Transit Feed Specification. GTFS Schedule Reference. 2026. <https://gtfs.org/documentation/schedule/reference/>.
- Karst T. Geurs and Bert van Wee. Accessibility evaluation of land-use and transport strategies: Review and research directions. *Journal of Transport Geography*, 12(2), pp. 127–140, 2004. <https://doi.org/10.1016/j.jtrangeo.2003.10.005>.
- GraphHopper. GraphHopper routing engine: Public transit support. 2026. <https://github.com/graphhopper/graphhopper#public-transit>.
- Ernestine Großmann, Jonas Sauer, Christian Schulz, Patrick Steil, and Sascha Witt. FLASH-TB: Integrating Arc-Flags and Trip-Based Public Transit Routing. *Transportation Science*, 60(2), pp. 242–263, 2026. <https://doi.org/10.1287/trsc.2023.0481>.
- Mordechai Haklay and Patrick Weber. OpenStreetMap: User-generated street maps. *IEEE Pervasive Computing*, 7(4), pp. 12–18, 2008. <https://doi.org/10.1109/MPRV.2008.80>.
- Christopher D. Higgins, Matthew Palm, Amber DeJohn, Yang Luna Xi, James Vaughan, Steven Farber, Michael Widener, and Eric J. Miller. Calculating place-based transit accessibility: Methods, tools and algorithmic dependence. *Journal of Transport and Land Use*, 15(1), pp. 95–116, 2022. <https://doi.org/10.5198/jtlu.2022.2012>.
- Olle Järv, Henrikki Tenkanen, Maria Salonen, Rein Ahas, and Tuuli Toivonen. Dynamic cities: Location-based accessibility modelling as a function of time. *Applied Geography*, 95, pp. 101–110, 2018. <https://doi.org/10.1016/j.apgeog.2018.04.009>.
- Mei-Po Kwan. Space-time and integral measures of individual accessibility: A comparative analysis using a point-based framework. *Geographical Analysis*, 30(3), pp. 191–216, 1998. <https://doi.org/10.1111/j.1538-4632.1998.tb00396.x>.
- Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner. Computing many-to-many shortest paths using highway hierarchies. In *Proceedings of ALENEX 2007*, pp. 36–45, 2007. <https://doi.org/10.1137/1.9781611972870.4>.
- MOTIS Project. MOTIS: Multimodal routing, geocoding, and map tiles. Software documentation, 2026. <https://github.com/motis-project/motis>.
- OpenTripPlanner Project. OpenTripPlanner 2 documentation: Data sources and routing algorithms. Software documentation, 2026. <https://docs.opentripplanner.org/en/latest/>.
- Andrew Owen and David M. Levinson. Modeling the commute mode share of transit using continuous accessibility to jobs. *Transportation Research Part A: Policy and Practice*, 74, pp. 110–122, 2015. <https://doi.org/10.1016/j.tra.2015.02.002>.
- Rafael H. M. Pereira, Marcus Saraiva, Daniel Herszenhut, Carlos Kaue Vieira Braga, and Matthew Wigginton Conway. r5r: Rapid realistic routing on multimodal transport networks with R5 in R. Findings, Article 21262, 2021. <https://doi.org/10.32866/001c.21262>.
- Moritz Potthoff and Jonas Sauer. Fast multimodal journey planning for three criteria. arXiv:2110.12954, 2021. <https://arxiv.org/abs/2110.12954>.
- Yanda Qu, Graham Currie, and Geoffrey I. Webb. Spatial resolution impact on public transport accessibility measurement error. *Journal of Transport Geography*, 129, Article 104408, 2025. <https://doi.org/10.1016/j.jtrangeo.2025.104408>.
- Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. An efficient solution for one-to-many multi-modal journey planning. In *Proceedings of ATMOS 2020, OASIs 85*, Article 1, 2020. <https://doi.org/10.4230/OASIs.ATMOS.2020.1>.

- Marcin Śpęniak, John P. Pritchard, Karst T. Geurs, and Sławomir Goliszek. The impact of temporal resolution on public transport accessibility measurement: Review and case study in Poland. *Journal of Transport Geography*, 75, pp. 8–24, 2019. <https://doi.org/10.1016/j.jtrangeo.2019.01.007>.
- Valhalla Project. Valhalla routing API documentation. Software documentation, 2026. <https://valhalla.github.io/valhalla/api/>.
- Sascha Witt. Trip-based public transit routing. In *Proceedings of ESA 2015, LNCS 9294*, pp. 1025–1036, 2015. https://doi.org/10.1007/978-3-662-48350-3_85.