Multi-criteria journey planning on the passenger's device: design and validation for fifteen merged regional transit networks

Yeray Garrido

***Abstract*—Journey planners for public transport usually compute routes on a server. This paper describes and evaluates one that computes them entirely on the passenger's device, in the browser, over fifteen merged transit networks in and around the Basque Country (8,242 stops, 54,896 trips). Offline it answers from the last timetables downloaded; online it repairs each answer with the real-time state of the trips shown. Its search is a multi-criteria range variant of round-based public transit routing on arrival and walking time, with bags of four labels that never discard the earliest arrival and lower bounds on the remaining time and rides computed once per target; a selection layer turns the result into five labelled options. On the production data, four labels find 88 % of the exact restricted Pareto set without changing the first option of 2,000 plan queries, and the bounds make plan queries 2.6 times faster. Earliest arrivals agree with an independent Connection Scan search and with an exhaustive enumeration of small networks. Against OpenTripPlanner on identical data and without pruning of walking options, each planner returns almost every journey of the other that its own walking rules allow, to within 60 s (ours) and 120 s (OpenTripPlanner); with the production walking table, ours returns 96 % of those of OpenTripPlanner within 60 s. Delaying or cancelling a third of the trips adds half to the query time. In single runs, the median plan query took 16 ms on a recent iPhone and 65 ms on a 2021 mid-range Android phone.**

***Index Terms*—Journey planning, client-side computation, optimization and control, public transportation, multimodal transportation networks.**

I. Introduction

A DOOR-TO-DOOR journey planner answers a simple question, how to get from here to there, with a few options that trade arrival time against the number of vehicles, the walking and the risk of missing a connection. Most deployed planners compute that answer on a server. This paper describes and evaluates one that computes it entirely on the passenger's device: the planner of Bide+[1], a web application that brings together the timetables and real-time data of the public transport networks of the Basque Country and its surroundings.

Running the search on the device was a design requirement. The application is served by a stateless, pay-per-request hosting platform, where a server-side planner would make the cost grow with every query; on the device the server only serves static files and the real-time state of the trips shown, the device's position stays on it, and the planner works offline with the last data it downloaded. The price is a tight budget: the search must run in a browser, on slow hardware, with the memory and download size a phone can afford, and with no preprocessing beyond static files rebuilt every night. The planner merges fifteen networks of different operators (metro, commuter, regional and long-distance trains, two tramways, urban and interurban buses, and the trains and buses of the French Basque Country), with 8,242 stops and 54,896 trips in the 45-day data of 21 November 2026; a journey may combine any of them, walking between stops of different operators, and delays, cancellations and extra trips must be applied as soon as they are known.

Its core is McRAPTOR [1], the multi-criteria version of the Round-bAsed Public Transit Optimized Router (RAPTOR), run over a departure window with labels on arrival and walking time; a selection layer turns the result into the five options a passenger sees. Most of its ingredients have precedents (Section II); the contributions are their combination under a phone's budget and the evidence that it works:

1) A range McRAPTOR with separate *arrive* and *ready-to-board* labels and bags of bounded size whose eviction rule keeps the earliest arrival per departure and number of rides exact, which we prove for range queries. On the production data, four labels per bag find 88% of the exact restricted Pareto set and left the first option shown unchanged in 2,000 plan queries (Sections IV and VII-E).
2) Lower bounds on the remaining time and rides computed once per target on a time-independent relaxation, valid for every departure of a range query and under real-time data, which make plan queries 2.6 times faster. Indexing them by remaining rides adds only about 5% over a single time bound with the minimum number of remaining rides (Sections IV-D and VII-F).
3) Real-time delays, cancellations and extra trips without preprocessing, with the cost measured up to a third of the network disrupted (Sections IV-F and VII-C).
4) A validation against an independent Connection Scan search, an exhaustive enumeration, the exact restricted set and OpenTripPlanner on identical data, an audit and a sensitivity analysis of the selection layer, and timings on phones with identical answers on every device (Section VII).
5) A deployment description for practitioners: the data life cycle on the device, the channels through which a query can still leak, and the cost of a planner without a routing

Y. Garrido is an independent researcher in Bilbao, Biscay, Spain (e-mail: garridoparrayeraytx@gmail.com; ORCID: 0009-0008-0459-0294).

This work received no specific grant from any funding agency in the public, commercial or not-for-profit sectors.

[1]https://www.bideplus.eus, an independent application not affiliated with the operators or authorities whose data it uses.

server (Section VI).

Sections II to VI review related work, define the problem and describe the search, the options and the deployment; Section VII evaluates them and Section VIII discusses limitations.

II. Related work

a) Timetable routing: Early timetable information systems ran variants of Dijkstra's algorithm on time-expanded or time-dependent graphs [2], [3]; multi-criteria versions keep a set of non-dominated labels per node [4], [5], with relaxed dominance for trains [6]. Bast *et al.* [7] survey the field. RAPTOR [1], [8] scans routes in rounds without preprocessing; McRAPTOR adds Pareto bags and rRAPTOR answers range queries by reusing the labels of later departures. The Connection Scan Algorithm (CSA) [9], [10] scans connections sorted by time. Transfer Patterns [11], Trip-Based routing [12] and partitioning [13] trade preprocessing for speed; ULTRA [14], [15] precomputes shortcuts for unrestricted walking, which restricted walking links can miss [16]. OpenTripPlanner 2 [17] runs a multi-criteria range RAPTOR that, like ours, carries labels across the departures of the window.

b) Restricted Pareto sets and lower bounds: Full Pareto sets can be very large [4]; ε-Pareto sets bound them with a guarantee [18], [19], and planners score or thin them for passengers [20]–[22]. Our restriction is not an ε-set: it keeps the exact earliest arrival per number of rides and thins the walking trade-off heuristically. Lower bounds that prune multi-criteria timetable search are older than RAPTOR: Disser *et al.* [23] obtain them from a backward search on a graph whose time-dependent costs are replaced by their minima, and SUBITO [24] computes per-query bounds that stay valid under delays. Bounded McRAPTOR [25] computes, by backward RAPTOR searches, a time-dependent bound per stop and number of remaining trips, and Potthoff and Sauer [26] extend it to arrival, trips and walking, the criteria used here. OpenTripPlanner 2 prunes with a reverse minimum-duration search that yields a time and a minimum number of transfers per stop, and MOTIS [27] with one bound per stop. Our bounds combine two of these ideas: they are indexed by remaining rides, like those of Bounded McRAPTOR, but computed on a time-independent relaxation, like those of Disser et al., so that one computation per target serves every departure of a range query and real-time data.

c) Real-time data and reliability: Delays were first handled on graph models updated on a server [28]; transfer patterns proved robust to them [29], and Delay-ULTRA [30] adds a fast update phase. Searches without preprocessing apply delays at query time; MOTIS scans the trips that real-time data modify apart from the route scan, as we do (Section IV-F). Reliability has been modelled from observed travel times [31], delay distributions [32] and stochastic delays [33]; our expected loss of a risky transfer is a lighter heuristic used only for ranking.

d) Passenger information and computing on the client: Real-time arrival information reduces perceived and actual waits [34], [35]; this journal has published frameworks for delivering it [36] and web planners that compute routes on a server [37]. Transfers cost more than their time [38], which our ranking approximates with a fixed penalty. Linked Connections [39]–[41] runs CSA in the user agent on pages of connections from a server; compact timetables [42] and compressed road graphs [43] make on-device data smaller. WebAssembly [44] brings compiled code to browsers, about 45–55% slower than native code [45]. Our planner ships the whole timetable, so it also works offline, and runs a compiled multi-criteria search in the browser.

III. Problem definition

A. Timetable

A timetable consists of a set of stops S and a set of trips. A trip visits a sequence of stops with an arrival and a departure time at each, in seconds from the midnight of its *service day* (possibly beyond 24 hours), and runs on the days of its calendar. Trips that visit the same stops and never overtake one another form a *route*; the data generator splits a sequence of stops into several routes when trips overtake, so this first-in-first-out (FIFO) property holds by construction, and Section IV-F deals with real-time data that breaks it. A query names a service day d; trips of days $d-1$ and $d+1$ are also usable, shifted by $\mp 86{,}400$ s. The fifteen networks are merged into one timetable; stops of different networks are different stops, and passengers move between them by walking.

B. Walking

Walking from the origin (access), between rides (transfer) and to the target (egress) is never computed geometrically during the search. A precomputed table gives, for every stop, the stops within 1,500 m with their distance along the street network, and, for every cell of a 100 m grid, the stops reachable from its centre; for the few stops away from any street (115 of 8,242 with the data of 21 November) it holds straight-line distances. The search uses transfers of up to 600 m and access and egress walks of up to 1,200 m, at a walking speed $v = 1.2$ m/s. To keep the table small, the links of a stop keep, in order of distance, only the stops that share a route with it or bring a route that no nearer linked stop brings, and each cell keeps, in the same way, at most 128 stops. A farther stop of a route that a nearer stop already serves is thus dropped, which can cost a journey that would board that route later at the farther stop; Section VII-G measures the loss. Distances from a cell centre are corrected to the exact point by projecting the offset onto the direction of the stop, never below the straight-line distance. When no stop is within 1,200 m, the nearest stop whose straight-line distance plus 30% is at most 2,500 m is used, and the answer says so. A second table, built on step-free street segments, serves accessible journeys.

C. Transfers

After alighting, a passenger can board another vehicle at the same stop after a margin of 60 s, or walk a transfer link and board after the walking time plus 60 s; a longer minimum transfer time chosen by the passenger replaces the margin. In the *comfortable* pass of Section V-A, every transfer also

requires a slack of two minutes after alighting at a stop of a bus or coach network, whose arrivals are less certain than a train's, and of one minute after walking; because every stop belongs to one network, the slack depends on the stop, not on the label, and dominance at a stop remains valid. In plan queries, a vehicle that would make the passenger wait more than two hours after a transfer is not boarded.

D. Journeys and criteria

A journey is a sequence of legs, alternating walking and rides, evaluated on four criteria: the departure time from the origin τ_{dep} (later is better), the arrival time at the target τ_{arr}, the number of rides k and the total walking time w (lower is better for the last three).

Definition 1 (Dominance): A journey J' *dominates* a journey J if $\tau_{\mathrm{dep}}(J') \geq \tau_{\mathrm{dep}}(J)$, $\tau_{\mathrm{arr}}(J') \leq \tau_{\mathrm{arr}}(J)$, $k(J') \leq k(J)$ and $w(J') \leq w(J)$, with at least one strict inequality.

A *range query* asks for the journeys that depart within a window $[t, t + \Delta]$ and that no other journey departing within the window dominates, with at most $K = 6$ rides. A *plan query* asks for up to five options for a departure (or arrival) time; the planner runs range queries over a window that grows from 30 minutes to four hours, backwards for arrive-by queries, and selects among their results (Section V).

The full Pareto set is large and much of it useless: a journey that saves twenty seconds of walking by arriving an hour later would never be chosen. The search therefore computes a restricted set. For each departure (to the minute) and number of rides the earliest arrival is exact; the *restricted Pareto set* adds the journeys that walk less and arrive at most 30 minutes after the fastest journey that departs no earlier with the same or fewer rides (the search tests this against the journeys it already knows), and bags of at most $B = 4$ labels keep part of that trade-off between arrival and walking. Section IV-G states the properties of the search.

IV. The search

This section describes the search; Sections S1 and S4 of the supplementary material give its pseudocode and its constants.

A. Background

RAPTOR [1] computes earliest arrivals in rounds: round k scans once every route that serves a stop improved in round $k - 1$, riding the earliest trip that can be boarded, and then relaxes transfers. Its range variant, rRAPTOR, runs one search per departure of a window, from the latest to the earliest, without resetting the labels, and McRAPTOR replaces the arrival per stop and round by a *bag* of Pareto-optimal labels. The planner keeps two criteria in the bags, arrival and walking time; the departure comes from the range iteration and the rides from the rounds.

B. Labels

A label is a point (τ, w), a time and the walking since the origin, with a *stamp* that identifies the departure that created it and a pointer to its parent. The search keeps three kinds of bags: an *arrive* bag $\mathcal{A}_k(p)$ for every stop p and round $k = 1, \ldots, K$, where the passenger has just alighted after k rides; a *ready* bag $\mathcal{R}_k(p)$ for $k = 0, \ldots, K - 1$, where the passenger can board the next vehicle at time τ, margins and transfer walks included ($\mathcal{R}_0$ holds the access walks); and a *target* bag $\mathcal{T}_k$, with the egress walk included. Separating arriving from being ready lets transfer margins be applied once, when moving from $\mathcal{A}_k$ to $\mathcal{R}_k$: a stop reached by walking and a stop reached by alighting have different readiness rules, and one time per stop would force either double margins or none. Each bag is a *staircase*, sorted by increasing time and strictly decreasing walking; a new label is rejected if an existing one is no later and walks no more, and otherwise removes the labels it dominates.

a) Bounded bags: Bags hold at most $B = 4$ labels. When a bag is full and a new label dominates none of its labels, the label whose walking saving over the next earlier one is smallest goes, ties to the latest; the first (earliest) and last (least walking) labels never go, and with $B = 1$ the bag keeps only the earliest label and the search compares arrival times only (Algorithm S1 of the supplementary material). The rule keeps the extremes of the trade-off and thins it where a passenger would least notice the difference.

b) Stamps and shadows: Labels are not cleared between departures. Each departure increments a stamp, and only *live* labels, those with the current stamp, are ridden, transferred or extracted as journeys; labels of later departures stay in the bags and prune. When a label enters a bag of round k, an unstamped copy, a *shadow*, enters the bags of the same kind for rounds $k + 1, \ldots, K$, because a way of arriving with k rides is also one with more; shadows are never expanded. Neither idea is new: rRAPTOR and OpenTripPlanner 2 keep labels of later departures, and RAPTOR starts each round from the labels of the previous one [1]; stamps let a label of a later departure prune without being expanded or returned again, and shadows are the multi-criteria form of the round-to-round reuse.

C. Range iteration

The search collects every time at which the passenger must leave the origin to catch a trip from a stop within access distance and runs these departures from latest to earliest, together when less than one minute apart, until no stop improves or $K = 6$ rides are reached; target labels with the current stamp become journeys, each checked against its label. In round 1 a vehicle is boarded only if leaving the origin for it falls within the window, so that a later vehicle cannot dominate a journey of the window. While no journey is known, walking cannot prune, so a departure first runs as a time-only *probe*; if it finds a journey, its live labels are removed (their shadows stay) and the departure runs again with both criteria.

D. Pruning

For a label with time τ and walking w after u rides at stop p, let $\mathrm{lb}_j(p)$ be a lower bound on the time to reach the target from p with at most j more rides. The label is pruned if, for every j from the minimum number of further rides (0 after alighting, 1 before boarding) up to $K - u$, the reach $\rho_j = \tau + \mathrm{lb}_j(p)$ is infinite or past the deadline of an arrive-by query; or the

target bag $\mathcal{T}_{u+j}$, which also holds the shadows of journeys with fewer rides, has a label (τ', w') with $\tau' \le \rho_j$ and $w' \le w$; or ρ_j is more than 30 minutes after the earliest arrival known with $u + j$ rides or fewer (the *walking slack*), which restricts the Pareto set on purpose (Section III). In arrive-by queries the same applies to the departure.

a) Lower bounds per remaining rides: The bounds are computed once per target on a time-independent relaxation that keeps, for every route and pair of consecutive stops, the fastest travel time of any trip, on any day. Two tables are filled backwards from the target, round by round as in RAPTOR but without clock times: $\mathrm{lb}_j^{\mathrm{alight}}(p)$, from having alighted at p with j rides left, and $\mathrm{lb}_j^{\mathrm{board}}(p)$, from being ready to board (Algorithm S5 of the supplementary material). Waiting and margins are ignored, and extra trips enter as one-trip routes.

Proposition 1 (Admissibility): For every stop p and $j \le K$, $\mathrm{lb}_j^{\mathrm{alight}}(p)$ and $\mathrm{lb}_j^{\mathrm{board}}(p)$ do not exceed the time of any journey from p (alighted, respectively ready to board) to the target with at most j rides, with or without real-time data.

Sketch. By induction on j, from the egress walks of the target stops. Each ride travels along consecutive stops of a route or of an extra trip, and each segment takes at least the fastest time of that route over all its trips; real-time data give one delay per trip, which shifts all its times alike and leaves every segment time unchanged, cancellations only remove trips, and the bounds are recomputed when the set of extra trips changes. Waiting and margins are non-negative, and every walk is a link of at most 600 m or an egress walk. □

Pruning with these bounds removes only labels that the same rules would remove at the target. With $B = 1$ every bag keeps only its earliest label, and the bounds prune such a label only when the target bags already hold what it could reach (Section S2 of the supplementary material); the answers were identical with and without bounds in all 1,000 plan queries of Section VII-F; with larger bags they can differ marginally, because pruning changes which labels compete for a full bag. Indexing by rides makes the bounds account for the rides that remain: reaching the target quickly with five rides says nothing about doing so with the two the target bag still leaves room for. Section VII-F shows that a single time bound together with the minimum number of remaining rides per stop captures almost all of this.

E. Scanning a route

A route scan keeps a small staircase of *riders*, trips being ridden with the walking of the label they were boarded from. At each position it alights the riders into $\mathcal{A}_k$ (and $\mathcal{T}_k$ near the target), then boards from the live labels of $\mathcal{R}_{k-1}$, skipping a label if a rider that walks no more already leaves before it is ready; the trip is found by binary search and one lookup in a table of the next trip of the route that runs that day and is not moved by real-time data. Transfers then produce ready labels at the same stop after the margin and at linked stops after the walk and the margin.

F. Real-time data

Real-time data arrive as one delay per trip and service day, cancellations and extra trips. Using the delay at the next stop of a trip for all its later stops is a design choice: it is much cheaper to apply than a stop-by-stop profile, and we have not measured its error. Delays break the FIFO property the scan relies on: a trip twenty minutes late may leave after its follower. The scan therefore runs in two parts: the route with its on-time trips, where FIFO holds (the next-trip table skips moved trips), and then each moved trip of days $d-1$, d and $d+1$ on its own, boarded where a live ready label is ready before its actual departure and alighting wherever useful. Without real-time data there is no extra work. Extra trips are kept in a fixed memory area (64 trips, 2,048 stop visits) and scanned like moved trips; the timetable is never rebuilt.

G. Properties of the search

The rules give two properties for range queries without a deadline or a wait limit; Section S2 of the supplementary material proves the first, and Section VII-B checks both.

1) For every departure of the window (to the minute) and number of rides $k \le K$, the earliest arrival among the journeys that leave at that time or later within the window with at most k rides is exact, whatever the bag size: the earliest label of a bag is never discarded, and pruning removes a label that could improve it only once the target bags hold that arrival.
2) If no bag overflows, the search returns every journey of the window that no other journey of the window dominates, to the minute, except those arriving more than 30 minutes after the fastest journey departing at the same time or later with the same or fewer rides; it may also return some of the latter, because the slack is tested against the arrivals known when a label is examined.

On the real data bags overflow in most queries (Section VII-E), so the second property rarely applies there; the first does whatever the overflow. Both assume FIFO also across consecutive service days, which the data generator enforces by splitting trips that would overtake. Plan queries also refuse waits of more than two hours after a transfer, which breaks the monotonicity on which dominance relies (Section VII-F).

V. From the Pareto set to five options

A passenger wants a few options that differ in ways that matter, each with a reason to choose it. The planner answers a plan query with at most five labelled options. The rules below are heuristics tuned with an audit tool on queries between random street points (Section VII-H); their constants (Table S2 of the supplementary material) are part of the specification.

A. Passes, window growth and filters

A plan query first finds the stops within access distance of each end (if an end has none, the answer says "too far") and checks with one time-only search whether any journey exists that day. The window then starts at 30 minutes and doubles, up to four hours, until the non-dominated candidates have five distinct arrival times (departure times in arrive-by queries); counting distinct times keeps five variants of one departure from stopping the window too early. Each window

is searched in up to three passes: a *tight* pass on time only, without waits over two hours, which finds what really arrives earliest; a *comfortable* pass with both criteria and the transfer slack of Section III; and, with favourite lines, a comfortable pass restricted to them. A transfer below the comfortable slack is *risky*. Filters (networks with real-time data, minimum transfer time, accessibility, a subset of networks) mark which routes and stops may be used; if they leave no journey, the planner drops them one at a time and says which. When nothing is found, a diagnosis explains why: too far, no service within four hours (with the nearest journey up to 24 hours later), too late for an arrive-by query, or no connection.

B. Short rides and implausible options

A ride of three minutes or less is replaced by walking when the stretch is at most 1,500 m and the walk arrives at most five minutes later at the end, leaves at most five minutes earlier at the start, or reaches the same next vehicle in the middle, counting the wait for the short ride; never if it adds a risky transfer, for accessible queries or for the only ride. An option is then discarded if it waits more than two hours between rides or passes again through its first stop; takes more than 180% of the shortest duration plus 15 minutes or uses more than two rides more than the option with fewest; is practically dominated (another departs no earlier, arrives no later with no more rides and at most three minutes more walking, plus three per ride saved, and is strictly better in time or rides); is matched by an option with fewer rides arriving at most one minute later and departing at most 20 minutes earlier; or has a ride that walking would match.

C. Ranking and expected loss

Candidates are scored by arrival time (departure, reversed, in arrive-by queries) plus five minutes per transfer, plus, for each risky transfer, its *expected loss*: half of how much later the next vehicle of the same line between the same two stops arrives, up to 30 minutes. The factor one half is a coarse, explicit probability of missing the connection, where a model of delays would compute it per transfer [32]. In depart-at queries a risky option that departs later is credited with the wait it avoids.

D. Choosing and labelling

The options are chosen in this order: the best score; the one that truly arrives earliest; the best with a favourite line; the one with fewest rides and the one that walks least, if they improve on the first; the next departure with the same lines; and, up to five, options arriving at most two hours after the first that are not variants of one already chosen. Each receives the first label that applies in Table S3 of the supplementary material (last of the day, tight transfer, delayed, line alert, your line, fastest, fewer transfers, less walking, and so on); Table I shows an answer. The "last of the day" label requires that nothing of the network leaves later towards either stop, which is conservative on trunks shared by several lines.

TABLE I
THE FIVE OPTIONS FROM CENTRAL BILBAO TO CENTRAL DONOSTIA, LEAVING FROM 08:00 ON TUESDAY 24 NOVEMBER. COACH: LURRALDEBUS LINE DO01; TRAIN: EUSKOTREN LINE E1; WALK: IN TOTAL. THE UNLABELLED OPTION FILLS THE LIST; THE TWO-HOUR LIMIT APPLIES ONLY TO SUCH OPTIONS, NOT TO THE ONE WITH FEWEST RIDES.

Label	Leaves	Arrives	Vehicles	Walk
Fastest	08:16	10:03	metro, coach	1.6 km
Less walking	08:09	10:10	bus, coach, bus	0.6 km
Leaves later (30 min)	08:46	10:33	metro, coach	1.6 km
–	08:46	10:41	metro, coach, bus	1.0 km
Fewer transfers	09:37	12:50	train	2.1 km

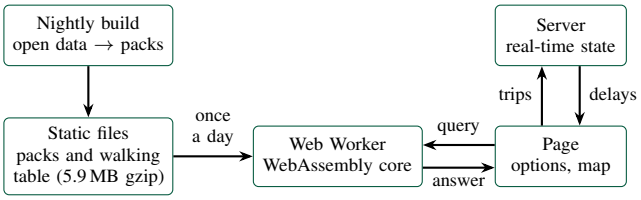

Fig. 1. How the planner runs. The server serves static files and, for each answer, the real-time state of the trips it shows; the search runs in a worker on the device.

E. After choosing

A chosen journey is encoded as the lines, stops and scheduled departures of its rides, because some operators change trip identifiers daily. *Prepare* rebuilds it on the current data with a plan B for each transfer; *Link* encodes it in a few dozen URL-safe characters, and opening the link rebuilds it and says whether each ride is the same, moved to the next vehicle, cancelled or gone.

VI. Implementation and deployment

A. Architecture and data life cycle

The core is written in C11 without dependencies, not even the C library, and compiled to WebAssembly for the browser and natively for testing; both give byte-identical answers. Every night the application rebuilds the timetables of all networks from the operators' open data, mostly General Transit Feed Specification (GTFS) feeds [46], into one *pack* per network covering 45 days, plus the walking table; a gate compares 45 fixed journeys with the deployed planner and keeps the previous day's data if one disappears or becomes much slower (Fig. 1). On first use, a Web Worker downloads the packs (5.4 MB, 1.2 MB with gzip), the walking table (8.9 MB, 4.6 MB with gzip) and the 105 kB module, and opens the core in 160 to 240 ms on the devices of Section VII-C. A service worker keeps the last version of each file and replaces it only when the server's version changes: the packs change every night, the walking table when the maps or the stops do. Offline, the planner answers from the last data it downloaded, without real-time data. Packs are used in place without parsing and checked when opened, and the core requests all its memory at start-up, so no search can run out of it (Section S8 of the supplementary material). The page sends JavaScript Object Notation (JSON) queries (`plan`, `range`, `live`, `prepare`, link encoding and listings) and receives JSON answers.

B. Real-time data in operation

Before showing an answer, the page asks the server for the real-time state of the trips that appear in the options (those of the next two hours, up to 40 per request), from GTFS Realtime and SIRI feeds [47] of the fourteen networks that publish them. If the state arrives within 2.5 s and changes anything, the query is repeated with the delays, cancellations and extra trips received before the options are shown; otherwise the options are shown from the timetable and redrawn when the state arrives. The state is refreshed every 30 s while the page is visible, and a delay not renewed for 60 s is dropped. This is a local repair, not network-wide real-time planning: a delayed trip that does not appear in the first answer, such as an earlier service that becomes catchable, is only seen if a later answer includes it. The core accepts the state of whole networks (Section VII-C); fetching it on every query would trade data volume for completeness. The application does not log queries, so how often the wait is exceeded or the repair changes the options is not known.

C. Privacy and cost

The search runs on the device and the device's position never leaves it: when the origin is the current location, the page records only that fact, and its address and shared links say "current location" rather than coordinates. Four channels reveal something about a query. The real-time request names the trips of the options, and thus, approximately, the origin, destination and time, to the application's server. Map tiles come from OpenStreetMap's tile servers, which see the area shown. The page writes a chosen stop, address or point on the map into its address, which reaches the server when the page is reloaded or a shared link is opened; the page-view analytics of the hosting platform receives addresses with these parameters removed. Place search uses an index downloaded with the data. Fetching the real-time state of whole networks would close the first channel. Per plan query the server does no routing: it serves cached static files once a day and answers one real-time request, as it already does for departure boards, whereas a server-side planner runs a search per query (Table S10 of the supplementary material).

VII. Evaluation

A. Setup

All measurements use the production data of 21 November 2026: fifteen packs with 8,242 stops, 561 lines, 2,711 routes and 54,896 trips over 45 days, and the walking table, as the application served them, with the core of engine revision `3b769a2` (the WebAssembly module of 104,699 bytes). Queries are for Tuesday 24 November 2026, and for the following weekend where stated. Origins and destinations are drawn with fixed seeds, half at stops and half at random points within about 500 m of one; plan queries use a random time between 05:00 and 23:00 and a random mode (depart at or arrive by). The native build uses GCC `-O2`, like the WebAssembly build (Clang), which runs in Node.js 22 through the application's wrapper. The machine is a shared four-core Linux virtual machine. Times are wall-clock per query, including the JSON round trip through a pipe; variants compared with each other run in the same session, alternating query by query, and intervals are 95% bootstrap intervals over queries (2,000 resamples). kB and MB are decimal; KiB and MiB binary.

B. Correctness

a) Independent earliest-arrival search: A separate Python program, sharing no code with the planner, reads the timetables from the networks' SQLite databases (not the packs), applies the same calendars, walking table, margins and day shifts, computes earliest arrivals with 1 to 6 rides with a Connection Scan and checks every journey the planner returns against the databases. On 300 random range queries over two days, the 725 earliest arrivals compared, with 1 to 6 rides from the start of the window, and the 7,689 journeys checked agreed in every case, as did 150 queries with a ten-minute minimum transfer time (344 arrivals) and 150 with 30% of the trips delayed or cancelled at random (235 arrivals). Because it uses the same walking table, this check validates the search, not the walking model, which Section VII-G compares with another planner's.

b) Exhaustive enumeration on small random networks: A second program builds 1,000 random worlds of two or three small networks, with walking links, partial calendars and, in some, delays and cancellations, enumerates every journey of up to six rides and compares the non-dominated set of the window with the planner's range search over 8,000 queries (Table S6 of the supplementary material). It checks that every returned journey exists; that for each of the 31,206 non-dominated journeys the planner returns one that departs no earlier, arrives no later and uses no more rides (the first property, per departure); and, where no bag overflows, that the restricted set is returned (the second). Bags rarely overflow here with four labels, so the core also ran with two and one: the first property held for every journey with every capacity, while with two labels bags overflow in 617 queries and 574 journeys that walk less are missing, and with one label, which compares arrivals only, the tool reports the 4,524 journeys that walk less as missing and 312 that walk more as dominated. Four labels also return 204 journeys beyond the slack, which the second property allows; the slack removes 9.9% of the non-dominated journeys of these worlds.

c) Builds, scenarios and malformed input: The native and WebAssembly builds give byte-identical answers, the module rebuilt from the measured revision is byte-identical to the one in production, 186 hand-written scenarios run on both builds and with sanitisers without errors, and a fuzzer found no read outside memory (Section S9 of the supplementary material).

C. Query times

Two sets of 1,000 plan queries were timed, natively and in WebAssembly, the two builds alternating query by query (Table II; Fig. S1 of the supplementary material gives the distribution). In the *region-wide* set both ends are drawn from the whole territory: 327 queries have no service within four hours (the answer then shows the nearest journey up to 24 hours later), 20 have no connection, and 75% of the first options

TABLE II
PLAN-QUERY TIMES IN MILLISECONDS (1,000 QUERIES PER SET, NATIVE AND WEBASSEMBLY IN NODE.JS ALTERNATING QUERY BY QUERY).

Set	Build	Median	90th	95th	99th	Max.
Region-wide	Native	21.4	118.6	173.5	353.9	633
	WebAssembly	32.7	171.3	259.0	497.0	984
Local	Native	16.2	66.7	107.7	245.3	504
	WebAssembly	23.2	93.8	160.6	325.2	728

TABLE III
PLAN-QUERY TIMES IN THE BROWSER WITH THE BENCHMARK PAGE, A TOOL OF THE CORE THAT IS NOT PUBLIC (MS; IDENTICAL ANSWERS ON EVERY DEVICE; RESULTS ARCHIVED). THIS MACHINE: MEDIAN OF FIVE RUNS. SAFARI REPORTS WHOLE MILLISECONDS.

	Region-wide			Local		
Device (browser)	Med.	95th	99th	Med.	95th	99th
This machine (Chromium)	22.7	170.2	433.2	15.8	116.9	267.7
Windows PC (Chrome)	33.3	248.4	604.0	24.4	165.5	387.9
iPhone 15 (Safari)	16	122	285	12	87	197
Redmi Note 10 (Chrome)	65.3	538.7	1,245.4	43.5	337.9	844.2

use three or more rides. In the *local* set the ends are at most 15 km apart, closer to real use: the first option uses one or two rides in 82% of the queries. WebAssembly took 1.49 times as long as native code in total [1.46, 1.51] and 1.40 to 1.53 times at the 50th to 99th percentiles of the region-wide set, in line with standard benchmarks [45]. A plan query ran a median of seven range searches, with 55 departures and 24,000 route scans; times grow with distance up to 40 km (median 13 ms below 5 km, 29 ms between 15 and 40 km) and are shorter for arrive-by queries, whose deadline prunes from the start (19 against 25 ms).

a) Repeated passes, other days and other devices: Five passes of the region-wide set in another session gave medians of 18.2 to 19.7 ms natively and 26.8 to 27.8 ms in WebAssembly: on this shared machine absolute times vary by up to a fifth between sessions, while ratios between builds or variants measured together vary by a few per cent. The same queries on Saturday 28 and Sunday 29 November gave native medians of 20.9 and 17.9 ms (23.1 ms on Tuesday in that session). On phones, a benchmark page runs 2,000 plan queries of the same kind, drawn by the page with a fixed seed, in a Web Worker with the production module and data, and prints their times and a hash of the answers. It was run on this machine (headless Chromium, five runs) and once on a Windows computer (Chrome), an iPhone 15 (Safari) and a Xiaomi Redmi Note 10 of 2021 with 4 GB of memory (Chrome); all gave identical answer fingerprints (Table III). Each device was measured once, without control of temperature or battery, so these are single observations: the iPhone took 0.7 times the median time of the headless browser on the virtual machine and the Redmi Note 10 about three times; half of its region-wide queries took less than 66 ms, 95% less than 0.54 s and 99% less than 1.25 s.

b) Real-time data: To measure the cost of network-wide disruption, the region-wide plan queries were answered with 0, 2, 5, 10, 20 and 30% of the 22,633 trips that run on the query day delayed by 90 s early to 30 min late, one in ten of them cancelled, as in the correctness check. The delays were sent in requests of 500: a request carries at most 1,024, but the core keeps the state of every trip (extra trips, in contrast, are limited to 64). In this session the native median grew from 22.7 ms without real-time data to 27.5 ms with 2% of the trips moved, 28.4 ms with 10% and 34.4 ms with 30% (6,784 trips), and the 95th percentile from 172 to 283 ms: the trips that real-time data moves are scanned one by one, but even a third of the network moved adds about half to the time. The delays are spread uniformly over trips, whereas real disruptions cluster on lines, and only the native build was measured.

D. Memory

With the production data the core uses 15.1 MB for the network, almost all of it the packs and the walking table used in place, and 15.6 MB of working memory for the search, dominated by the bag labels (8,242 stops $\times$ 6 rounds $\times$ 2 bags $\times$ 4 labels $\times$ 32 B); the whole WebAssembly linear memory is 31,981,568 bytes (30.5 MiB) and does not change during a search (Table S4 of the supplementary material). The working memory is about 1,900 bytes per stop and grows linearly with the bag capacity: eight labels per bag would add 12.7 MB.

E. Bag size and the restricted Pareto set

To measure how much of the trade-off between arrival and walking the bag capacity keeps, 200 random range queries with a one-hour window were run with $B \in \{1, 2, 3, 4, 6, 8, 16\}$ alternating query by query, and with $B = 64$, which never overflowed and therefore gives the exact restricted set of the second property of Section IV-G. Table IV gives the share of the pooled frontier of all capacities that each finds, with intervals from resampling the queries. With one label the search finds 14%, the earliest arrival per departure and number of rides; four labels find 87.2% at about twice the time of a time-only search. Against the exact restricted set (3,390 journeys), four labels find 87.6% [84.4, 90.9] (92% on average per query, all of it in at least half of the queries), eight 99.2% and sixteen all of it. The pooled frontier is slightly larger because smaller bags also return some journeys beyond the slack, which is tested against the arrivals known at the time. What four labels miss is small: covering each missing journey with one that is returned requires giving up a median of 34 s of departure, arrival or walking (95th percentile 152 s, at most 5.4 min). The answers a passenger sees barely change: over 2,000 plan queries, the first option with four labels is identical to the one with 64, walking and legs included, and the set of options differs in 32. Bags overflow in 169 of the 177 queries with journeys and in 599 of the 1,000 plan queries, but an overflow never affects the earliest arrivals.

a) The walking slack and the number of rides: Without the walking slack, bags of 64 return the full Pareto set on the four criteria (6,338 journeys in 196 queries): 52% of it arrives more than 30 minutes after a faster journey that departs no earlier with no more rides, which the slack excludes on purpose, and removing the slack changes 162 of the 1,000 plan answers. Allowing seven rides instead of six gives 12 more

TABLE IV
SHARE OF THE POOLED FRONTIER OF ALL CAPACITIES (3,407 JOURNEYS IN 177 QUERIES) FOUND BY EACH BAG CAPACITY, WITH 95% INTERVALS OVER QUERIES, AND RANGE-QUERY TIME (NATIVE, ONE-HOUR WINDOW, 200 QUERIES).

Capacity B	1	2	3	**4**	6	8	16
Found (%)	14.3	53.3	75.2	**87.2**	96.1	98.7	99.5
95% interval	12.1	48.9	71.0	**84.0**	94.5	98.0	99.2
	16.5	57.9	79.7	**90.4**	97.6	99.3	99.8
Median (ms)	8.6	14.9	16.1	**17.1**	17.6	18.2	18.5
95th pct. (ms)	27.8	63.2	82.6	**79.4**	89.5	89.0	101.1

region-wide queries an option and changes the first option in 86, at 1.5 times the time; with five, 173 first options change. Six rides is therefore a product choice that costs some long cross-regional journeys.

F. Effect of the lower bounds

With the bounds replaced by zero in the pruning test, which leaves the usual target pruning, the 1,000 region-wide plan queries took 2.57 times as long in total with four labels [2.43, 2.73] (3.0 times at the median) and 2.06 times with one label [1.96, 2.16] (Table V); with one label all answers are identical, and with four 989 are, the rest differing because pruning changes which labels compete for a full bag. Table V compares the bounds per number of rides with two simpler ones read from the same table: a single bound per stop (the entry for all K rides) and the same bound together with the minimum number of rides with which the target is reached from the stop, as in the time and transfer bounds of OpenTripPlanner and of Disser *et al.* [23]. The single bound is 1.83 times slower than the indexed bounds with four labels [1.71, 1.95]; adding the minimum number of rides recovers almost all the difference (1.05 times [1.03, 1.08], and 1.01 with one label). The benefit therefore comes from bounding both the time and the rides that remain; indexing the time bound by rides adds about 5%. Rerun in WebAssembly with the archived modules of these variants, the ratios were 2.68, 1.87 and 1.08, with the same numbers of identical answers. Computing the indexed bounds took a median of 3.3 ms per target, once per plan query, and 8% of the summed query time. Removing the two-hour wait limit from the search, while the selection layer still removes such waits, changes 18 of the 2,000 first options and gives an earlier one in 2. Both ways of applying the limit can lose a journey, because it breaks the monotonicity on which dominance relies (Section S3 of the supplementary material), so this compares them but does not bound the loss against an exact search with the limit. Run in one interleaved pass with production, the core with sixteen labels and no bounds took 3.2 times as long [3.0, 3.4] for 99% instead of 87% of the pooled frontier, and with bounds sixteen labels cost 1.12 times the time of four: time does not limit the capacity, whereas working memory grows with it (Section S6-0a of the supplementary material).

G. External validation against OpenTripPlanner

To check the search and the walking model against an independent implementation, the same timetables, streets and

TABLE V
LOWER BOUNDS (NATIVE, 1,000 REGION-WIDE PLAN QUERIES, VARIANTS ALTERNATING QUERY BY QUERY; ABSOLUTE TIMES DIFFER FROM TABLE II BETWEEN SESSIONS). SCANS: ROUTE SCANS (M). SAME: ANSWERS IDENTICAL TO PER RIDES.

	Median (ms)	95th (ms)	Total (s)	Scans (M)	Same
$B = 4$**, per rides**	**23.3**	**188.8**	**49.3**	**56.9**	–
$B = 4$, time + min. rides	25.2	191.9	51.9	60.2	1,000
$B = 4$, single	44.7	336.0	90.0	91.6	994
$B = 4$, none	69.8	425.2	126.7	125.4	989
$B = 1$, per rides	16.2	115.1	31.2	46.9	–
$B = 1$, time + min. rides	17.1	117.3	31.5	49.4	1,000
$B = 1$, single	27.6	163.7	48.0	70.9	1,000
$B = 1$, none	41.1	200.1	64.2	97.4	1,000

queries were given to OpenTripPlanner 2.10.0 [OTP; 17], whose transit search is also a multi-criteria range RAPTOR, on the same machine.

a) Setup: The packs of 21 November were exported to one GTFS feed per network with the 22,663 trips that run on 23, 24 or 25 November, which a search on the 24th can use; a separate reader confirmed that the feeds hold the same trips, stops, times and service days. OTP built its street graph from the OpenStreetMap extracts of our walking table and received our constants (Table S1 of the supplementary material): walking at 1.2 m/s, access and egress up to 1,000 s, transfers up to 500 s, a 60 s slack and at most six rides; the extra time its walking model adds at crossings and stairs was switched off, and its default model is reported in the supplementary material. OTP's answers were read before its itinerary filters (in debugging mode) and after them, as it serves them. The main sample has 200 one-hour range queries drawn like those of Section VII-E; a second sample of 1,000 queries, with another seed, checks stability.

b) Agreement under the same walking distances: OTP's fourth criterion is a generalised cost rather than walking, so journeys are compared on departure, arrival and rides; a journey is matched within a tolerance ε by one that departs at most ε earlier, arrives at most ε later and uses no more rides. Each journey of one planner was re-timed with the walking rules of the other, and only journeys that those rules allow were kept: 3,364 of the 3,635 OTP journeys of both samples under our walking distances, and 14,821 of 19,052 of ours under OTP's. Both planners also prune their walking options, our table by keeping only the stops that bring a new route (Section III) and OTP by keeping one transfer per pattern of vehicles. Without these prunings the two searches agree almost completely: our search with an unpruned table returns, within 60 s, all 3,364 OTP journeys (3,361 to the second), and OTP returns all but 17 of the 14,821 of ours (2,320 in the main sample and 12,501 in the second) within 120 s. With the production table, which keeps at most 128 instead of 256 stops per cell and takes 8.9 instead of 15.7 MB, our planner returns 95.6% [90.9, 99.1] of OTP's journeys within 60 s in the main sample and 95.9% [94.4, 97.3] in the second: the pruning costs about 4% of them. Where OTP has the earlier arrival (17 of the 142 queries of the main sample in which both have a journey in the window), the median difference is 3 min, the second largest 36 min and

the largest 11.1 h, also caused by the pruning (Section S12 of the supplementary material).

c) What each planner returns: Without re-timing, the planners differ mainly through their walking models. Table VI gives, per planner, how many journeys it returns, how many of them no other journey of its own beats, the share of those that the other planner does not beat by more than ε (precision), and the share of the pooled frontier of both that it finds (recall). Our range query boards from the origin only within the window, whereas OTP may wait past its end and return a journey that leaves later and hides earlier ones. Two conventions are therefore reported: in the *window* one only journeys that leave within the window compete; in the *edge-neutral* one OTP's later journeys may also beat others but do not count as found. The second favours OTP: under it 69–70% of our journeys are not beaten by OTP, against 95–96% under the first, the difference being journeys that an OTP journey leaving after the window beats. Under either convention our planner returns 1.7 to 1.8 times as many non-dominated journeys as OTP and finds 94–95% of the pooled frontier, against 59–62% (window) or 78–83% (edge-neutral) for OTP. The tolerance $\varepsilon = 120$ s is a difference a passenger would hardly notice; Fig. S2 of the supplementary material gives recall at every tolerance up to 300 s, and at $\varepsilon = 0$ the edge-neutral shares are 57% against 43%. Almost every discrepancy comes from walking: with the same extracts the two planners link stops to the streets and choose paths differently, our limits are distances and OTP's are times, and some stops and places are not reached by OTP's streets (Table S9 of the supplementary material); a hand check of 30 discrepancies by the author, not blind, agreed with this classification after one correction. With OTP's default walking model the recall gap widens (96.9% against 59.7%). These shares therefore measure agreement between two configured systems on a common frontier, not which planner is right about a street.

d) Threats to validity: The comparison covers one region, one working day and one version of OTP; the second sample is larger, not an independent replication. OTP used our constants rather than its defaults, and the re-timing relies on each planner's own walking rules. Queries are uniform over stops, including places outside OTP's street extracts; restricted to the queries in which OTP locates both ends, the recall gap shrinks by a quarter to a third but persists (Section S12 of the supplementary material).

H. The selection layer

a) Plausibility on fresh queries: An audit tool runs plan queries between random street points and counts options a passenger would find odd. The rules of Section V were tuned with it on queries of seed 1; seeds 2 to 4 of the same generator give 1,200 queries that were never used for tuning, which rules out fitting to the tuning queries but not to the way queries are drawn. Per 100 options shown, the fresh queries give 14.5 to 19.2 transfers waiting more than 25 minutes (17.5 with seed 1), 10.0 to 13.7 options with four or more vehicles (13.4), 1.9 to 2.5 consecutive rides on the same line (1.9) and at most 0.3 rides that walking would match (0.2). Without a reference, such as another planner's served options on the same queries, these counts show that the rules behave alike on new queries, not that the options are good. Long waits and many vehicles come from sparse rural timetables, and every option that another of the list beats in time has fewer risky transfers (Section S11 of the supplementary material).

b) Sensitivity of the thresholds: Each of the 13 thresholds of the selection layer was halved and doubled in turn, and the 2,000 plan queries of Section VII-C were answered again (Table S7 of the supplementary material). The departure, arrival and rides of the first option, the one a passenger sees on top, change in at most 3 of the 1,979 answers with options when a filter or choice threshold changes, except the short-ride threshold (41 and 47 answers); counting walking and legs, the first option changes in at most 64. Ranking constants and the comfort slack change it in 1 to 10% of the answers, in the intended direction: doubling the transfer penalty gives a first option with one ride fewer in all 80 answers that change, arriving later in 75 (median 8 min), and doubling the probability of missing a risky transfer gives one with fewer risky transfers in 53 of 54, arriving later in 49 (median 14 min); halving either does the opposite. The order and the rest of the list change more, because the filters decide which alternatives enter it. The constants are product choices: the analysis shows which ones matter and what they trade, not which values passengers prefer.

VIII. Discussion and limitations

a) What the restricted set gives up: The planner computes the exact earliest arrival per departure and number of rides and a trade-off between arrival and walking bounded by the walking slack and by four labels per bag. On the production data the slack excludes about half of the full Pareto set, journeys that arrive more than half an hour later than a faster one to save some walking; four labels find 87.6% of the exact restricted set, each journey they miss within a median of 34 s of one returned, and left the first option of 2,000 plan answers unchanged. Both restrictions are product decisions: eight labels would find 99% at about the same time but double the memory of the bags. The reference set comes from the same core with bags that never overflow, checked against an independent enumeration only on small networks, and, unlike an ε-Pareto set [18], the thinning has no worst-case guarantee.

b) The bounds: The bounds pay for the second criterion under a phone's budget, but their indexing by remaining rides is not what makes them effective: a time bound with a minimum number of remaining rides, as used by OpenTripPlanner and by Disser *et al.* [23], does almost as well. Time-dependent bounds such as those of Bounded McRAPTOR [25] are tighter, at the cost of searches per query; we have not compared them with ours, nor run Bounded McRAPTOR or MOTIS on these data.

c) The selection layer: Its rules were tuned with the audit tool on random queries, not fitted to passenger preferences; the audit with fresh seeds and the sensitivity analysis show that they are not tied to the tuning queries and which constants matter, not that passengers would choose these five options,

TABLE VI
WHAT EACH PLANNER RETURNS (ONE-HOUR RANGE QUERIES, $\varepsilon = 120$ s, 95% BOOTSTRAP INTERVALS OVER QUERIES). NON-DOMINATED: NOT BEATEN BY ANOTHER JOURNEY OF THE SAME PLANNER. PRECISION: SHARE OF THOSE NOT BEATEN BY THE OTHER PLANNER. RECALL: SHARE OF THE POOLED FRONTIER FOUND. CONVENTIONS AS IN THE TEXT; OTP BEFORE ITS ITINERARY FILTERS (AFTER THEM, PRECISION 89% AND EDGE-NEUTRAL RECALL 65.5% AND 61.5%).

	Main sample (200 queries)		Second sample (1,000 queries)	
	Ours	OTP	Ours	OTP
Returned, all	2,995	960	16,057	5,321
Returned, non-dominated	818	458	4,174	2,462
Precision, window (%)	96.2 [94.0, 98.0]	91.0 [86.4, 94.8]	95.4 [94.2, 96.5]	91.6 [89.5, 93.4]
Precision, edge-neutral (%)	69.2 [63.5, 74.1]	91.0 [86.4, 94.8]	70.4 [68.3, 72.4]	91.6 [89.5, 93.4]
Recall, window (%)	95.4 [93.1, 97.3]	58.7 [51.8, 64.9]	95.1 [94.1, 96.1]	61.7 [59.0, 64.2]
Recall, edge-neutral (%)	94.1 [91.0, 96.6]	78.1 [71.8, 83.4]	93.9 [92.6, 95.2]	82.5 [80.2, 84.7]

and no other planner's served options were compared. The fixed 50% probability of missing a risky transfer could be replaced by one learned from the real-time data the application receives, in the spirit of Dibbelt *et al.* [33]. No user study has been made.

d) Real-time data and walking: The deployed page repairs an answer with the real-time state of the trips it shows and does not see delays of other trips; how often the repair changes an answer in use is not known. The core handles the state of whole networks at a cost measured with synthetic delays spread uniformly over trips. One delay per trip ignores stop-by-stop predictions, skipped stops and short turns. Walking comes from a table with a 100 m grid, which does not know on which side of a river or a railway a point lies; against OpenTripPlanner almost every difference comes from walking, and the pruning of the table costs about 4% of its journeys, and 11.1 h of arrival in one query.

e) Scope of the evidence: All measurements use one extraction of the data, with queries on one Tuesday and, for times, a Saturday and a Sunday; one region of 8,242 stops, so nothing shows how the approach scales; ends drawn uniformly over stops rather than from demand; and region-wide queries that often need four or more rides, of which six rides cut some long cross-regional journeys. Accessible queries were not evaluated, and the feeds carry no lift status. Each phone was measured once, without control of temperature, battery or energy use, and two phones are not a sample. The packs cover 45 days from their build and a browser may evict stored files, so a device offline long enough has no timetable; the cost of operation is argued, not measured. The source code of the core and its test tools are not public; to compensate, the paper specifies every rule and constant, and the archive holds the data, the queries, the test logs, the scripts, and the compiled module of every variant with the public loader and a script that reruns the WebAssembly versions of the bound and capacity tables.

IX. CONCLUSION

A multi-criteria journey planner for fifteen merged regional networks can run entirely on the passenger's device, with answers in tens of milliseconds on phones and no routing server. Without pruning of walking options, it and an established server-side planner each return almost every journey of the other that their own walking rules allow, and with the production walking table it returns 96% of them. What makes it fit is a range McRAPTOR with bounded bags that keep the earliest arrival exact, lower bounds on the remaining time and rides computed once per target, and real-time data applied without preprocessing. Two lessons may transfer to other client-side planners: bounding the remaining rides matters more than refining the time bound, and the walking model, not the search, decides most of the differences between planners on the same data.

DATA AVAILABILITY

The planner runs in the Bide+ web application (https://www.bideplus.eus). The source code of the core and its test tools are not public; the paper and its supplementary material specify every rule and constant that affects the results. Archived on Zenodo (https://doi.org/10.5281/zenodo.24442212) are the data of 21 November 2026; the module measured (104,699 bytes, revision `3b769a2`, rebuilt byte for byte from it) and the module of every variant, with their SHA-256, the public loader and a script that runs them with Node.js; the queries and per-query results of every table; the measurement and comparison scripts; the GTFS feeds, OpenTripPlanner 2.10.0 configuration and raw answers of both planners; and the logs of the test tools and the results on each device.

ACKNOWLEDGMENT

The author develops and operates Bide+, an independent application not affiliated with the operators or authorities whose data it uses, and declares no other competing interests. Generative AI assistants, Claude (Anthropic) and Gemini (Google), were used to help write and run the experiment scripts; the author checked them and takes full responsibility for the content.

REFERENCES

[1] D. Delling, T. Pajor, and R. F. Werneck, "Round-based public transit routing," *Transportation Science*, vol. 49, no. 3, pp. 591–604, 2015. DOI: 10.1287/trsc.2014.0534.

[2] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis, "Efficient models for timetable information in public transportation systems," *ACM Journal of Experimental Algorithmics*, vol. 12, 2.4:1–2.4:39, 2008. DOI: 10.1145/1227161.1227166.

[3] K. G. Zografos and K. N. Androutsopoulos, "Algorithms for itinerary planning in multimodal transportation networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 1, pp. 175–184, 2008. DOI: 10.1109/TITS.2008.915650.

[4] P. Hansen, "Bicriterion path problems," in *Multiple Criteria Decision Making Theory and Application*, ser. Lecture Notes in Economics and Mathematical Systems, G. Fandel and T. Gal, Eds., vol. 177, Berlin, Heidelberg: Springer, 1980, pp. 109–127. DOI: 10.1007/978-3-642-48782-8_9.

[5] E. Q. V. Martins, "On a multicriteria shortest path problem," *European Journal of Operational Research*, vol. 16, no. 2, pp. 236–245, 1984. DOI: 10.1016/0377-2217(84)90077-8.

[6] M. Müller-Hannemann and M. Schnee, "Finding all attractive train connections by multi-criteria Pareto search," in *Algorithmic Methods for Railway Optimization*, 2007, pp. 246–263. DOI: 10.1007/978-3-540-74247-0_13.

[7] H. Bast, D. Delling, A. Goldberg, *et al.*, "Route planning in transportation networks," in *Algorithm Engineering: Selected Results and Surveys*, ser. Lecture Notes in Computer Science, L. Kliemann and P. Sanders, Eds., vol. 9220, Cham: Springer, 2016, pp. 19–80. DOI: 10.1007/978-3-319-49487-6_2.

[8] D. Delling, T. Pajor, and R. F. Werneck, "Round-based public transit routing," in *Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2012, pp. 130–140. DOI: 10.1137/1.9781611972924.13.

[9] J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner, "Intriguingly simple and fast transit routing," in *Experimental Algorithms. 12th International Symposium, SEA 2013*, 2013, pp. 43–54. DOI: 10.1007/978-3-642-38527-8_6.

[10] J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner, "Connection scan algorithm," *ACM Journal of Experimental Algorithmics*, vol. 23, 1.7:1–1.7:56, 2018. DOI: 10.1145/3274661.

[11] H. Bast, E. Carlsson, A. Eigenwillig, *et al.*, "Fast routing in very large public transportation networks using transfer patterns," in *Algorithms – ESA 2010*, 2010, pp. 290–301. DOI: 10.1007/978-3-642-15775-2_25.

[12] S. Witt, "Trip-based public transit routing," in *Algorithms – ESA 2015. 23rd Annual European Symposium*, 2015, pp. 1025–1036. DOI: 10.1007/978-3-662-48350-3_85.

[13] D. Delling, J. Dibbelt, T. Pajor, and T. Zündorf, "Faster transit routing by hyper partitioning," in *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, 2017, 8:1–8:14. DOI: 10.4230/OASIcs.ATMOS.2017.8.

[14] M. Baum, V. Buchhold, J. Sauer, D. Wagner, and T. Zündorf, "UnLimited TRAnsfers for multi-modal route planning: An efficient solution," in *27th Annual European Symposium on Algorithms (ESA 2019)*, 2019, 14:1–14:16. DOI: 10.4230/LIPIcs.ESA.2019.14.

[15] M. Baum, V. Buchhold, J. Sauer, D. Wagner, and T. Zündorf, "ULTRA: Unlimited transfers for efficient multimodal journey planning," *Transportation Science*, vol. 57, no. 6, pp. 1536–1559, 2023. DOI: 10.1287/trsc.2022.0198.

[16] D. Wagner and T. Zündorf, "Public transit routing with unrestricted walking," in *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*, 2017, 7:1–7:14. DOI: 10.4230/OASIcs.ATMOS.2017.7.

[17] OpenTripPlanner project, *OpenTripPlanner*, version 2.10.0, Open-source multimodal trip planner; documentation at https://docs.opentripplanner.org/, 2026. [Online]. Available: https://github.com/opentripplanner/OpenTripPlanner/tree/v2.10.0 (visited on Nov. 21, 2026).

[18] C. H. Papadimitriou and M. Yannakakis, "On the approximability of trade-offs and optimal access of web sources," in *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS 2000)*, 2000, pp. 86–92. DOI: 10.1109/SFCS.2000.892068.

[19] G. Tsaggouris and C. Zaroliagis, "Multiobjective optimization: Improved FPTAS for shortest paths and nonlinear objectives with applications," *Theory of Computing Systems*, vol. 45, no. 1, pp. 162–186, 2009. DOI: 10.1007/s00224-007-9096-4.

[20] D. Delling, J. Dibbelt, T. Pajor, D. Wagner, and R. F. Werneck, "Computing multimodal journeys in practice," in *Experimental Algorithms. 12th International Symposium, SEA 2013*, 2013, pp. 260–271. DOI: 10.1007/978-3-642-38527-8_24.

[21] H. Bast, M. Brodesser, and S. Storandt, "Result diversity for multi-modal route planning," in *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2013)*, 2013, pp. 123–136. DOI: 10.4230/OASIcs.ATMOS.2013.123.

[22] P. Campigotto, C. Rudloff, M. Leodolter, and D. Bauer, "Personalized and situation-aware multimodal route recommendations: The FAVOUR algorithm," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 1, pp. 92–102, 2017. DOI: 10.1109/TITS.2016.2565643.

[23] Y. Disser, M. Müller-Hannemann, and M. Schnee, "Multi-criteria shortest paths in time-dependent train networks," in *Experimental Algorithms. 7th International Workshop, WEA 2008*, 2008, pp. 347–361. DOI: 10.1007/978-3-540-68552-4_26.

[24] A. Berger, M. Grimmer, and M. Müller-Hannemann, "Fully dynamic speed-up techniques for multi-criteria shortest path searches in time-dependent networks," in *Experimental Algorithms (SEA 2010)*, 2010, pp. 35–46. DOI: 10.1007/978-3-642-13193-6_4.

[25] D. Delling, J. Dibbelt, and T. Pajor, "Fast and exact public transit routing with restricted Pareto sets," in *Proceedings of the Twenty-First Workshop on Algorithm Engineering and Experiments (ALENEX)*, 2019, pp. 54–65. DOI: 10.1137/1.9781611975499.5.

[26] M. Potthoff and J. Sauer, "Fast multimodal journey planning for three criteria," in *Proceedings of the Symposium on Algorithm Engineering and Experiments*

(ALENEX 2022), 2022, pp. 145–157. DOI: 10.1137/1.9781611977042.12.

[27] MOTIS project, *MOTIS: Multimodal routing, geocoding, and map tiles*, Open-source software, MIT license, 2026. [Online]. Available: https://github.com/motis-project/motis (visited on Nov. 20, 2026).

[28] M. Müller-Hannemann and M. Schnee, "Efficient timetable information in the presence of delays," in *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*, 2009, pp. 249–272. DOI: 10.1007/978-3-642-05465-5_10.

[29] H. Bast, J. Sternisko, and S. Storandt, "Delay-robustness of transfer patterns in public transportation route planning," in *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2013)*, 2013, pp. 42–54. DOI: 10.4230/OASIcs.ATMOS.2013.42.

[30] D. Bez and J. Sauer, "Fast and delay-robust multimodal journey planning," in *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, 2024, pp. 105–117. DOI: 10.1137/1.9781611977929.8.

[31] K. Böhmová, M. Mihalák, T. Pröger, R. Srámek, and P. Widmayer, "Robust routing in urban public transportation: How to find reliable journeys based on past observations," in *13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2013)*, 2013, pp. 27–41. DOI: 10.4230/OASIcs.ATMOS.2013.27.

[32] M. H. Keyhani, M. Schnee, K. Weihe, and H.-P. Zorn, "Reliability and delay distributions of train connections," in *12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2012)*, 2012, pp. 35–46. DOI: 10.4230/OASIcs.ATMOS.2012.35.

[33] J. Dibbelt, B. Strasser, and D. Wagner, "Delay-robust journeys in timetable networks with minimum expected arrival time," in *14th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2014)*, 2014, pp. 1–14. DOI: 10.4230/OASIcs.ATMOS.2014.1.

[34] K. E. Watkins, B. Ferris, A. Borning, G. S. Rutherford, and D. Layton, "Where is my bus? Impact of mobile real-time information on the perceived and actual wait time of transit riders," *Transportation Research Part A: Policy and Practice*, vol. 45, no. 8, pp. 839–848, 2011. DOI: 10.1016/j.tra.2011.06.010.

[35] B. Ferris, K. Watkins, and A. Borning, "OneBusAway: Results from providing real-time arrival information for public transit," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI 2010)*, 2010, pp. 1807–1816. DOI: 10.1145/1753326.1753597.

[36] T. Ma, G. Motta, and K. Liu, "Delivering real-time information services on public transit: A framework," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 10, pp. 2642–2656, 2017. DOI: 10.1109/TITS.2017.2656387.

[37] L. S. C. Pun-Cheng, "An interactive web-based public transport enquiry system with real-time optimal route computation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 2, pp. 983–988, 2012. DOI: 10.1109/TITS.2011.2181501.

[38] Z. Guo and N. H. M. Wilson, "Assessing the cost of transfer inconvenience in public transport systems: A case study of the London Underground," *Transportation Research Part A: Policy and Practice*, vol. 45, no. 2, pp. 91–104, 2011. DOI: 10.1016/j.tra.2010.11.002.

[39] P. Colpaert, A. Llaves, R. Verborgh, O. Corcho, E. Mannens, and R. Van de Walle, "Intermodal public transit routing using Linked Connections," in *Proceedings of the ISWC 2015 Posters & Demonstrations Track co-located with the 14th International Semantic Web Conference (ISWC 2015)*, 2015. [Online]. Available: https://ceur-ws.org/Vol-1486/paper_28.pdf.

[40] P. Colpaert, R. Verborgh, and E. Mannens, "Public transit route planning through lightweight Linked Data interfaces," in *Web Engineering. 17th International Conference, ICWE 2017*, 2017, pp. 403–411. DOI: 10.1007/978-3-319-60131-1_26.

[41] J. A. Rojas, H. Delva, P. Colpaert, and R. Verborgh, "Publishing public transport data on the Web with the Linked Connections framework," *Semantic Web*, vol. 14, no. 4, pp. 659–693, 2023. DOI: 10.3233/SW-223116.

[42] S. Wandelt, X. Sun, and Y. Zhu, "Lossless compression of public transit schedules," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 11, pp. 3075–3086, 2016. DOI: 10.1109/TITS.2016.2542131.

[43] P. Sanders, D. Schultes, and C. Vetter, "Mobile route planning," in *Algorithms – ESA 2008. 16th Annual European Symposium*, 2008, pp. 732–743. DOI: 10.1007/978-3-540-87744-8_61.

[44] A. Haas, A. Rossberg, D. L. Schuff, *et al.*, "Bringing the Web up to speed with WebAssembly," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '17)*, 2017, pp. 185–200. DOI: 10.1145/3062341.3062363.

[45] A. Jangda, B. Powers, E. D. Berger, and A. Guha, "Not so fast: Analyzing the performance of WebAssembly vs. native code," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 107–120. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/jangda.

[46] MobilityData, *GTFS Schedule reference*, General Transit Feed Specification, Revised 27 April 2026, 2026. [Online]. Available: https://gtfs.org/documentation/schedule/reference/ (visited on Nov. 20, 2026).

[47] MobilityData, *GTFS Realtime reference*, General Transit Feed Specification, Feed specification version 2.0, 2026. [Online]. Available: https://gtfs.org/documentation/realtime/reference/ (visited on Nov. 20, 2026).