CIRRELT

CIRRELT-2025-30

Generalized Route Scheduling Problem with Time-dependent Travel Times on Road Networks

Bahman Bornay
Michel Gendreau
Bernard Gendron

September 2025

Bureau de Montréal
Université de Montréal
C.P. 6128, succ. Centre-Ville
Montréal (Québec) H3C 3J7
Tél : 1-514-343-7575
Télécopie : 1-514-343-7121

Bureau de Québec
Université Laval,
2325, rue de la Terrasse
Pavillon Palasis-Prince, local 2415
Québec (Québec) G1V 0A6
Tél : 1-418-656-2073
Télécopie : 1-418-656-2624

Generalized Route Scheduling Problem with Time-dependent Travel Times on Road Networks

Bahman Bornay[1,2*], Michel Gendreau[1,2], Bernard Gendron[1,3]

1. Interuniversity Research Centre on Enterprise Networks Logistics and Transportation (CIRRELT)
2. Department of Mathematics and Industrial Engineering, Polytechnique Montréal
3. Department of Computer Science and Operations Research, Université de Montréal

Abstract. We study the Generalized Route Scheduling Problem with Time-dependent Travel Times on Road Networks (GRSP-TDRN), where each request offers multiple pickup/dropoff candidates with time windows. A feasible solution must (i) choose a cluster sequence, (ii) select one node per cluster, and (iii) optimize the continuous depot departure time—decisions coupled through time-dependent travel. To the best of our knowledge, this is the first work to address a *generalized* route scheduling problem with time-dependent (TD) travel time functions defined over *real road networks*. We develop a continuous–time framework based on closed-form arrival/departure functions and lower–envelope propagation. First, we give two efficient solvers for TD quickest-path queries (TDQPP): TDQPFA and TDDA. Second, we present two exact dynamic-programming schemes—Linear Forward Propagation (LFP) and Post–Order Binary–Tree Propagation (PO–BTP, iterative and recursive)—prove their correctness and complexity and show that under uniform time cost they coincide with a Bellman–Ford value function. We also design a fast move evaluation module that performs feasibility screening, filters non-promising moves, and computes exact move costs from cached data, powered by a robust bidirectional search that retrieves minimal composition chains from binary trees. On 30 sparse and 30 dense networks, our TDQPP solvers outperform a state-of-the-art Bellman–Ford variant: TDDA achieves mean speedups of 70.7% (sparse) and 71.9% (dense), while TDQPFA achieves 43.0% (sparse) and 117.2% (dense); best–case gains exceed 8× and reach 16× against the state-of-the-art. Embedded in a sequential route–construction heuristic for up to 3,000 requests, PO–BTP is consistently fastest on larger instances. Precomputing cluster–to–cluster functions yields geometric–mean schedule speedups of 1.8–2.3× across all the proposed DP methods (wins in 30⁄30 cases per density). The framework applies broadly to VRPTW/PDPTW/DARP with TD travel on real networks and operates without time discretization.

Keywords: Time-dependent, routing, road network, timing problem, generalized, DARP, PDPTW, VRP, SPPRC

---

* Corresponding author: bahman.madadkar-bornay@etud.polymtl.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2025

1 Introduction

Vehicle routing problems (VRPs) are central to transportation and logistics, yet two features increasingly strain classical models: (i) *time-dependent (TD) travel times* that vary with departure time, and (ii) *service flexibility*, where each request provides candidate pickup and/or delivery locations. These features require continuous-time rescheduling on the road network after every move, whether the outer method is exact (e.g., decomposition) or heuristic (e.g., Adaptive Large Neighborhood Search (ALNS)).

Route scheduling is a recurring subproblem across exact and heuristic methods. After each insertion/removal or local move, the route must be rescheduled, often cast as a Shortest Path Problem with Resource Constraints (SPPRC) with tailored labeling, resource extension functions, and dominance rules (Desrochers and Soumis 1988, Desrosiers et al. 1995, Irnich and Desaulniers 2005, Desaulniers 2006, Irnich 2008). Yet customer-based (multi)graphs typically assume static shortest paths and ignore time dependency. Recent work mitigates this shortcoming: Gmira et al. (2021) develop dominant paths on discretized horizons; Vidal et al. (2021) give closed-form dominant paths that avoid discretization; and Visser and Spliet (2020) streamline move evaluations for Time-dependent Vehicle Routing Problem with Time Windows (TDVRPTW).

Gap in the literature. The recent survey by Adamo et al. (2024, Sec. 9.2) emphasizes the need to *combine* the closed-form TD preprocessing of Vidal et al. (2021) with the fast move-evaluation techniques of Visser and Spliet (2020) to obtain space- and time-efficient route rescheduling. Existing strands cover these pieces only partially: Vidal et al. (2021) derive closed-form TD *arrival-time* profiles in preprocessing, while Visser and Spliet (2020) accelerate move evaluations on *complete customer graphs* (arc-level TD), without network-level preprocessing or optimization. To our knowledge, no prior framework unifies these elements in a single continuous-time approach that simultaneously (i) models request-level flexibility—each request admits a *set* of candidate pickup locations and a *set* of candidate drop-off locations, each with its own time window; (ii) accounts for time-dependent travel times in continuous time (no discretization); and (iii) operates on the physical road network rather than a complete customer graph. This paper fills that gap with a three-phase pipeline (preprocessing, optimization, postprocessing) based on closed-form TD profiles and exact, provably correct DP-based route schedulers.

We consider TD road networks that satisfy FIFO (Ichoua, Gendreau, and Potvin 2003); the triangle inequality need not hold. We model TD functions (travel, arrival, departure) on subnetworks with possible parallel arcs. Routes represented as sequences of request-based *clusters* rather than single customer nodes. In our generalized setting, each request offers a set of candidate pickup locations and a set of candidate delivery locations; a feasible solution selects one node per cluster and schedules the route accordingly.

Within this framework we introduce the Generalized Route Scheduling Problem with Time-dependent Travel Times on Road Networks (GRSP-TD$_{\mathcal{RN}}$), a route-scheduling model that ties together three decisions: (i) the optimal sequence of clusters, (ii) the optimal node within each cluster, and (iii) the optimal start time from the depot, which connects to the Optimal Start Time Problem (OSTP).

Contributions. (i) **Generalized road-network model.** We formalize the GRSP-TD$_{\mathcal{RN}}$ where requests have candidate pickup/delivery clusters; logical arcs map to subnetworks (with parallel arcs) and are evaluated

in continuous time. (ii) **Closed-form TD arrival/departure functions.** We present two algorithms for the Time-Dependent Quickest Path Problem (TDQPP) that compute closed-form TD arrival-time functions on road subnetworks, supporting extension across the network and fast feasibility checks. Using point–line duality, we further give a tailored algorithm that constructs lower envelopes of continuous piecewise-linear (CPL) TD arrival and departure functions. On the set of instances we consider for experiments (sparse and dense networks), these TDQPP algorithms outperform the method of Vidal et al. (2021) in preprocessing runtime; see Section 7. (iii) **Exact scheduling via DP.** We design two exact dynamic-programming schemes—Linear Forward Propagation (LFP) and Post-Order Binary Tree Propagation (PO-BTP, recursive and iterative). (iv) **Fast move evaluation.** We design fast feasibility screening and lower-bounding scheme to filter out non-promising moves, together with an *offline exact route-cost* computation that reuses cached data (see Table 2), instead of rerunning the DP scheduler for every move. This is powered by the provably optimal tree search algorithms detailed in our companion work (Bornay, Gendreau, and Gendron 2025), which correct a critical flaw in a prior method (Visser and Spliet 2020) and guarantee the retrieval of valid, minimal composition chains. (v) **Empirical evaluation in two tracks.** *Track A (TDQPP benchmarking):* we compare our TD quickest-path algorithms with recent methods on sparse and dense road networks and report performance improvements. *Track B (DP schedulers in context):* we embed LFP and PO-BTP (R-PO-BTP, I-PO-BTP) in a sequential route construction heuristic for a generalized DARP with TD travel times on road networks, using instances with up to 3,000 requests. We evaluate two execution modes—*precomputed* (stop-to-stop closed-form departure-time functions computed once) and *on-the-fly*—and report runtimes and the speedups obtained by precomputation, which are especially marked in static-request settings. (vi) **Complexity results.** We establish $\mathcal{NP}$*-completeness in the strong sense* for feasibility and $\mathcal{NP}$*-hardness in the strong sense* for optimization of the continuous-time Bellman–Ford route-scheduling model (Section 2 and Appendix C).

We address generalized route scheduling problems where the objective is either: (1) to minimize total route duration, (2) to minimize the total travel and waiting times (which under certain assumptions coincides with route duration), or (3) to solve a multi-objective problem with lexicographic prioritization, where total route duration serves as the primary objective and additional service-level goals define secondary objectives.

The paper is organized as follows: Section 2 formalizes the GRSP-TD$_{\mathcal{RN}}$ and presents a Bellman–Ford–style recurrence for clustered routes. Section 3 develops TD-function modeling on road networks (subnetworks, envelopes, extensions). Section 5 details LFP and PO-BTP. Section 6 covers post-processing (optimal node selection and path recovery). Section 7 reports computational results. Appendix A lists all abbreviations used in the paper; Appendix B provides the shorthand notation used in Appendices C–F, which present the supplementary theoretical and algorithmic results (proofs and complexity analyses).

2 Problem Definition: Route Scheduling for GRSP-TD$_{\mathcal{RN}}$

We introduce the Generalized Route Scheduling Problem with Time-dependent Travel Times on Road Networks (GRSP-TD$_{\mathcal{RN}}$), which captures TD travel times on physical road networks and represents a route as a sequence of pickup/delivery *clusters* (one node per cluster is ultimately selected), starting and ending at a depot. This framework extends the classical SPPRC—a key subproblem in exact methods for

VRPTW/PDPTW/DARP—and includes those problems as special cases.

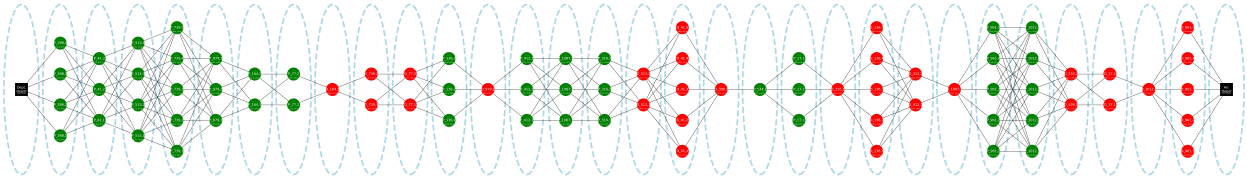

Figure 1: Layered route representation. Ellipses are clusters; black/green/red nodes denote depot/pickup/dropoff locations. $\sigma(r) = 32$, with 15 assigned requests. Zoom in to view the ID and time window of each stop.

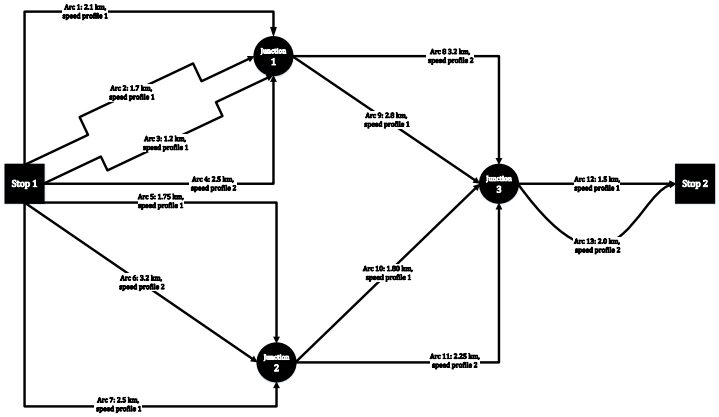

Figure 2: A subnetwork between two stops with three junctions and multiple road segments; parallel arcs may exist with distinct speed profiles.

Definition 1 (Route sequence and subnetwork representation) *Let $r \in \mathscr{R}$ denote a vehicle route. Define $\sigma(r)$ as the sequence of clusters in route r, beginning at the departure depot cluster $\nu(o)$, followed by pickup and drop-off clusters $\nu(i)$ for $i \in \{1, \ldots, 2L\}$, and ending at the arrival depot cluster $\nu(d)$. Thus, the full route sequence is:*

$$\sigma(r) := \{\nu(o), \nu(1), \nu(2), \ldots, \nu(2L), \nu(d)\}. \tag{1}$$

Here, L is the number of requests, and each request m has a pickup-cluster $\mathscr{P}_m$ and a delivery-cluster $\mathscr{D}_m$. Hence, there are $2L$ clusters in between the depot clusters, each cluster representing a set of stops in the underlying road network (see Figure 2). For all requests $m \in \mathscr{M}$ assigned to the route r, the precedence and coupling constraints must be satisfied:

$$\forall m \in \mathscr{M}, \quad \exists\, i : \nu(i) = \mathscr{P}_m, \quad \Longrightarrow \quad \exists\, j > i : \nu(j) = \mathscr{D}_m, \tag{2}$$

where $\mathscr{P}_m$ and $\mathscr{D}_m$ denote the pickup and dropoff clusters of request $m \in \mathscr{M}$, respectively, with $\mathscr{P}_m \cap \mathscr{D}_m = \emptyset$, and $\mathscr{P}_m, \mathscr{D}_m \subset \mathcal{N}$ in the road network $G_{\mathcal{RN}} := (\mathcal{N}, \mathcal{A})$. As illustrated in Figure 1, the structure of a route follows a layered graph representation, where each node in cluster $\nu(i)$ is connected to every node in the next

cluster $\nu(i+1)$. *Formally, the set of* logical arcs *in the route sequence is given by:*

$$\mathcal{A}_{\sigma(r)} \coloneqq \bigcup_{i=0}^{2L} \Big\{ (n, n') \mid n \in \nu(i),\ n' \in \nu(i+1) \Big\}. \tag{3}$$

The arcs in $\mathcal{A}_{\sigma(r)}$ *represent logical travel options at the route layer. However, at the physical road network level, each such arc corresponds to a subnetwork, as illustrated in Figure 2.*

The route scheduling problem involves three interdependent optimization layers: (i) *Optimal sequencing*—determining the order of visits subject to capacity, precedence, and time window constraints; (ii) *Optimal location selection*—choosing specific pickup and dropoff locations from each cluster (one per cluster) to minimize cost; and (iii) a third challenge arises from TD travel times, requiring an additional optimization known as the Optimal Start Time Problem (OSTP) (Hashimoto, Yagiura, and Ibaraki 2008), which computes the optimal departure time from the depot. Notably, the outcomes of both sequencing and location selection depend on the OSTP solution.

We formulate the GRSP-TD$_{\mathcal{RN}}$ as a DP model using a Bellman–Ford equation. For a route sequence as in Definition 1, define the system function

$$\omega_{i,j}(t) = \begin{cases} (\gamma_j \circ \phi_{ij})(t), & \text{if } (i,j) \in \mathcal{A}_{\sigma(r)}, \\ \infty, & \text{otherwise}, \end{cases} \quad \forall\, i \in \nu(h-1),\ j \in \nu(h),\ h \in \{1, 2, \ldots, 2L{+}1\}. \tag{4}$$

Here, $\phi_{ij}(t) = t + \theta_{ij}(t)$ is the TD arrival time at j when departing i at t, $\theta_{ij}(t)$ is the TD travel time, $\gamma_j(t) = \max\{a_j, t\}$ enforces the ready time at j with time window $[a_j, b_j]$, and $\omega_{ij}(t)$ is the TD departure from j (after any waiting). The TD waiting time is

$$\chi_j(t) = \max\{0,\ a_j - \phi_{ij}(t)\} \ = \ \omega_{ij}(t) - \phi_{ij}(t).$$

The Bellman–Ford recursion reads:

$$\overleftarrow{V}_{2L+1,\,j}(\tau) = 0, \quad \forall\, j \in \nu(d),\ \forall\, \tau \in [a_j, b_j] \tag{5}$$

$$\overleftarrow{V}_{h-1,\,i}(t) = \min_{j \in \nu(h)} \left\{ \begin{array}{l} \underbrace{\underbrace{\alpha\, c_{ij}\, \theta_{ij}(t)}_{\text{travel cost}} + \underbrace{\beta\, w_j\, \chi_j(t)}_{\text{waiting cost}}}_{\text{arc transition cost}} + \underbrace{\overleftarrow{V}_{h,\,j}(\omega_{ij}(t))}_{\text{cost-to-go}} \\ \text{subject to:} \\ t + \theta_{ij}(t) \le \omega_{ij}(t), \\ \chi_j(t) = \omega_{ij}(t) - t - \theta_{ij}(t), \\ \chi_j(t) \in \mathbb{R}_{\ge 0},\ \forall\, t \in [a_i, b_i], \\ \theta_{ij}(t) \in \mathbb{R}_{\ge 0},\ \forall\, t \in [a_i, b_i], \\ \omega_{ij}(t) \in [a_j, b_j],\ \forall\, t \in [a_i, b_i] \end{array} \right\}$$

$$\forall\, i \in \nu(h-1),\ \forall\, h \in \{1, 2, \ldots, 2L{+}1\}, \tag{6}$$

where c_{ij} and w_j are the per-unit costs of travel and waiting time, respectively, with weights $\alpha, \beta > 0$ and $\alpha + \beta = 1$. The optimal route cost is

$$\min_{\tau \in [a_o, b_o]} \overleftarrow{V}_{0,\,o}(\tau), \tag{7}$$

where $\tau^* = \arg\min_{\tau \in [a_o, b_o]} \overleftarrow{V}_{0,\,o}(\tau)$ solves the OSTP (itself $\mathcal{NP}$-hard).

Remark 1 (Reduction to route duration) *If travel and waiting are penalized at the same per-unit rate (i.e., no distinct waiting charge), normalize so that $\alpha c_{ij} = \beta w_j = 1$ for all (i, j). Then the arc cost reduces to $\theta_{ij}(t) + \chi_j(t)$. Consequently, the route objective reduces to route duration (arrival at $d \in \nu(d)$ minus start time).*

Proposition 1 (SPPRC-TD$_{\mathcal{RN}}$: Generalization of the SPPRC) *The recursion in Equations*(5)-(6) *generalizes standard resource-constrained shortest-path models on consecutive clusters; with uniform time cost as desribed in Remark 1, it yields TD-SPPTW-CC-PD and TD-ESPPTW-CC-PD; otherwise, it yields TD-SPPTW-CC-TC-PD, and TD-ESPPTW-CC-TC-PD variants; Here, TD = time-dependent, CC = capacity constraints, TC = waiting time penalties distinct from travel cost, PD = pickup-delivery precedence, and E = elementary paths.*

Route–subnetwork interplay and single–query cost. For consecutive clusters $\nu(k)$ and $\nu(k{+}1)$, each logical arc $i \to j$ induces a road-level subnetwork S_{ij} between stops i and j. For a fixed depot start time τ, layer k evaluates $\omega_{ij}(t_i) = \gamma_j(\phi_{ij}(t_i))$ with $\phi_{ij}(t_i) = t_i + \Theta_{ij}(t_i)$, where t_i is the departure produced at i by the previous layer and $\Theta_{ij}(t)$ is the TD quickest-path travel time across S_{ij} when departing at t. Inside S_{ij}, arrivals propagate through all incoming links; with parallel links, each junction takes the minimum over predecessors before continuing. This subnetwork evaluation is required at every layer even for a single τ. If τ changes later, the whole route-layer evaluation must be recomputed from the new start time. Hence global optimality over $[a_o, b_o]$ requires solving (7); sampling finitely many start times simply repeats the route–subnetwork evaluation per sample and remains approximate. In the sequel, we replace such sampling by closed-form extended departure-time functions (see §3.4: Subnetwork to Logical Arc Mapping) and optimize exactly over continuous interval $[a_o, b_o]$.

Since global optimality requires optimizing over a continuous horizon, we next establish that this scheduling layer is computationally intractable, even in highly simplified settings.

Theorem 1 (Feasibility is $\mathcal{NP}$-complete in the strong sense) *Even with the following restrictions: (i) every cluster is a singleton; (ii) travel times are static FIFO; (iii) the start time is fixed (τ=0); (iv) no capacity constraints; and (v) precedence is a chain of checkpoint clusters $M_1 \prec \cdots \prec M_m \prec d$ (each a singleton with point window $[jB, jB]$ and zero incoming travel time), deciding feasibility of the Bellman–Ford schedule is $\mathcal{NP}$-complete in the strong sense.*

Theorem 2 (Optimization is $\mathcal{NP}$-hard in the strong sense) *Under the same restrictions outlined in Theorem 1, minimizing route duration is $\mathcal{NP}$-hard in the strong sense.*

Proof sketches are given in Appendix C; full reductions and bookkeeping appear there. These hardness results justify replacing repeated single-time evaluations by closed-form extended departure-time functions in the sequel, and they hold *a fortiori* for time-dependent arcs.

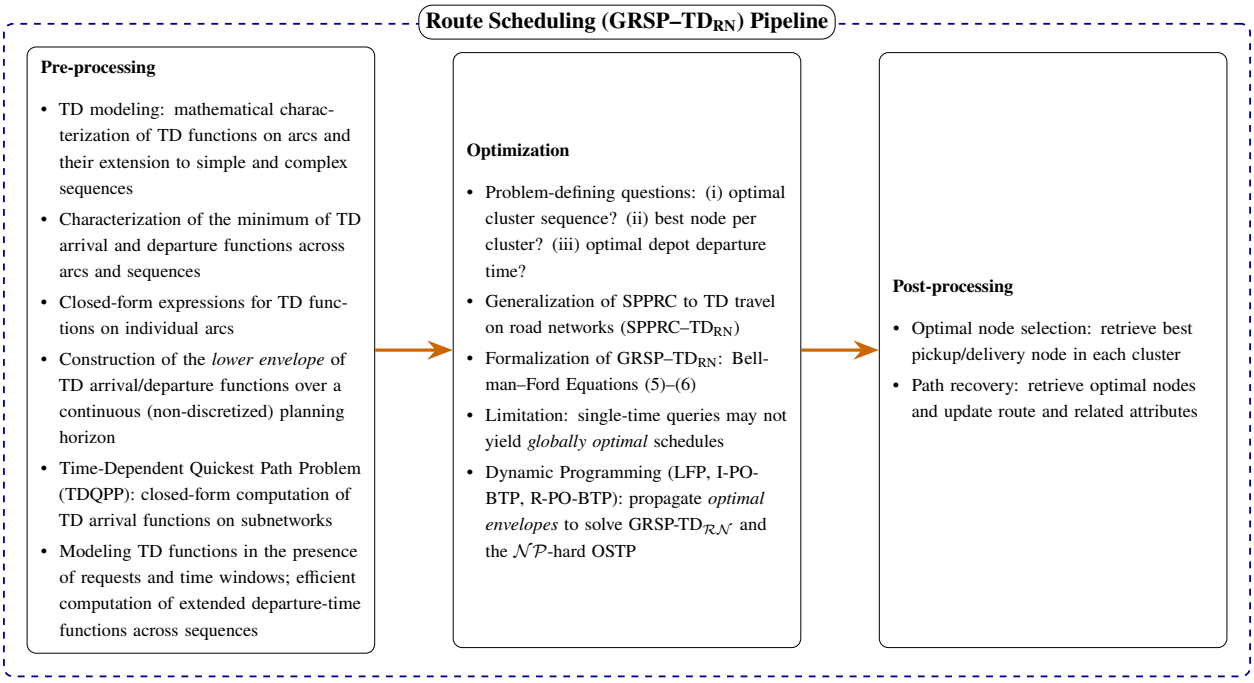

Figure 3: Overview of our three-phase route-scheduling framework.

Given the inherent complexity of the GRSP-TD$_{\mathcal{RN}}$, Figure 3 provides a schematic overview to guide the reader through our proposed three-phase solution method. This roadmap illustrates the logical progression from modeling TD functions, to formulating and solving the route scheduling problem, and finally to post-processing. It bridges the theoretical foundations with their algorithmic and practical applications, clarifying how each component fits into the overall framework.

3 Pre-processing Phase: Modeling TD Functions on Real Road Networks

The solution of the GRSP-TD$_{\mathcal{RN}}$ relies critically on the ability to model time-dependent functions on real road networks. This section develops the mathematical foundations required for these operations, which later enable the efficient dynamic programming algorithms presented in Section 5.

A road network, $G_{\mathcal{RN}}(\mathcal{N}, \mathcal{A})$, consists of stops $\mathcal{S}$ and junctions/intersections $\mathcal{J}$ (nodes) connected by links (arcs). Let $\mathcal{N} := \{\mathcal{S}, \mathcal{J}\}$ and $\mathcal{A}$ denote the sets of all unique nodes and arcs defining the road network, respectively. We define the time horizon $\mathscr{H} = [t_{\text{init}}, t_{\text{max}}]$ as the period over which operations on $G_{\mathcal{RN}}$ are analyzed.

The network $G_{\mathcal{RN}}$ is considered time-independent if $\nu_{ij} : \mathscr{H} \to C$ for all $(i, j) \in \mathcal{A}$, where ν_{ij} is the average speed on arc (i, j) and $C \in \mathbb{R}^+$ is a constant. This implies $\frac{d\nu_{ij}}{dt} = 0$ for all $t \in \mathscr{H}$ and $(i, j) \in \mathcal{A}$. Conversely, $G_{\mathcal{RN}}$ is said to be TD if one assumes that average speeds on arcs vary over time. This assumption is widely used, as seen in the model proposed by Ichoua, Gendreau, and Potvin (2003). While one might consider applying such a method to a customer-based graph, as discussed in Section 1, this

approach often leads to suboptimal solutions. At the *road network level*, each pair of customer locations is not directly connected by necessarily a single arc but rather by an *underlying subnetwork* $G'_{\mathcal{RN}} = (\mathcal{N}', \mathcal{A}')$, where: $\mathcal{N}' \subseteq \mathcal{N}$ and $\mathcal{A}' \subseteq \mathcal{A}$. Each arc in this subnetwork has specific attributes that reflect its unique properties. By partitioning the time horizon exhaustively, we can assume $\nu_{ij}(t) \coloneqq v_\tau$ for $t \in \delta t_\tau$, where $\mathscr{H} = \{\delta t_\tau\}_{\tau=1}^{n}$ with disjoint intervals $\delta t_\tau \cap \delta t_{\tau+1} = \emptyset$ for all $\tau \in \{1, 2, \ldots, n-1\}$. Let $\delta t_\tau = [\underline{t}_\tau, \bar{t}_\tau]$, so that for $(i,j) \in \mathcal{A}$, $\mathscr{B}_{\nu_{ij}} = \bigcup_{\tau=1}^{n-1}\{\bar{t}_\tau\}$ forms a sorted collection, in ascending order, of all breakpoints in the piecewise constant speed function $\nu_{ij}(t)$. Associated with each arc $(i,j) \in \mathcal{A}$ are its primary attributes: length d_{ij} and a speed profile index (SPI). The SPI bounds the speed on each arc, i.e., $\nu_{ij}^{\pi}(t) \in [\underline{v}^{\pi}, \overline{v}^{\pi}]$ for $\pi \in \Pi$, where Π is a finite set of natural numbers used to classify arcs according to a SPI-based labeling scheme. Specifically, $(i,j) \in \mathcal{A}^{\pi}$ indicates that arc (i,j) belongs to a class of arcs, $\mathcal{A}^{\pi} \subset \mathcal{A}$, such that $\mathcal{A} = \bigcup_{\pi=1}^{|\Pi|} \mathcal{A}^{\pi}$ and $\mathcal{A}^{\pi} \cap \mathcal{A}^{\pi+1} = \emptyset$. Under these conditions, for any pair of locations in the network, one can construct travel time profiles such that, for any departure time, there exists a unique mapping of arrival time to the destination node. This ensures that a later departure time results in a later arrival time, satisfying the FIFO (First-In-First-Out) property (Ichoua, Gendreau, and Potvin 2003), which holds throughout the road network (Ghiani and Guerriero 2014). Furthermore, we assume that $\nu_{ij}(t) > 0$ almost everywhere on $\mathscr{H}$ for all $(i,j) \in \mathcal{A}$, ensuring that arcs are traversable except possibly at isolated breakpoints.

Definition 2 (Time-dependent Travel Time Function on Road Networks $\theta_{ij}(.)$) *For arc $(i,j) \in \mathcal{A}$, and piecewise-constant speed function $\nu_{ij}(t)$, let us define*

$$d_{ij} = \int_{\alpha}^{\alpha+\theta_{ij}(\alpha)} \nu_{ij}(t)\, dt \tag{8}$$

where α is the departure time from node i, $\theta_{ij}(\alpha)$ is the TD travel time on arc (i,j) starting from time α, leading to the arrival time of $\phi_{ij}(\alpha) = \alpha + \theta_{ij}(\alpha)$, having finished the journey of length d_{ij} as we reach the end of the link, i.e., node j.

The TD arrival time function $\phi_{ij}(t)$, and its inverse $\phi_{ij}^{-1}(t)$ are defined in Equations (9) and (10).

$$\phi_{ij}(t) = \begin{cases} t + \theta_{ij}(t) & \text{if } t + \theta_{ij}(t) \in \mathscr{H} \\ \infty & \text{otherwise} \end{cases} \tag{9}$$

$$\phi_{ij}^{-1}(\beta) \coloneqq \sup\{t \in \mathscr{H} : \phi_{ij}(t) \leq \beta\} \tag{10}$$

The functions $\theta_{ij}(t)$, $\phi_{ij}(t)$, and $\phi_{ij}^{-1}(t)$ are continuous piecewise-linear (CPL), with $\phi_{ij}(t)$ being monotonically non-decreasing. For a single time query t, the function $\phi_{ij}(t)$ returns the arrival time at node j when departing from node i at time t. However, single-point evaluations are not our primary interest. Instead, we aim to retrieve the closed-form representation of these TD functions, thereby retaining their values for any valid time query. Figure 4 illustrates the speed ν_{ij} function, the travel time function θ_{ij}, and the arrival time function ϕ_{ij} for a representative arc (i,j) in the road network.

Definition 3 (Closed-form representation of a CPL function ξ) *Let $\mathscr{B}_\xi = \{t_0 < t_1 < \cdots < t_m\}$ be the sorted breakpoints of a given CPL function $\xi_{i,k}(t)$. Its closed-form representation—denoted in capital letters—is the set of linear segments:*

$$\Xi_{i,k}(t) = \bigcup_{j=0}^{m-1} \left\{ (t_j, \xi_{i,k}(t_j)), (t_{j+1}, \xi_{i,k}(t_{j+1})) \right\}, \tag{11}$$

where each pair $(t_j, \xi_{i,k}(t_j)), (t_{j+1}, \xi_{i,k}(t_{j+1}))$ defines the affine piece on $[t_j, t_{j+1}]$. This convention is used throughout the paper.

Let $\mathscr{B}_{\nu_{ij}} \coloneqq \{\bar{t}_\tau\}_{\tau=1}^{n-1}$, where $\delta t_\tau \coloneqq [\underline{t}_\tau, \bar{t}_\tau]$ is the sorted collection of all breakpoints of the piecewise-constant speed function $\nu_{ij}(t)$. Then, $\theta_{ij}(t)$ has at most $2\left|\mathscr{B}_{\nu_{ij}}\right|$ breakpoints, which coincide with those of $\phi_{ij}(t)$ and $\phi_{ij}^{-1}(t)$, as illustrated in Figure 4. On any given arc in the road network, Algorithm 1 computes the TD travel time $\theta_{ij}(t)$ for a single-time departure time query, while Algorithm 2 retrieves the breakpoints of θ_{ij} and constructs $\Phi_{ij}(t)$ — the closed-form representation of $\phi_{ij}(t)$.

We have so far characterized the TD functions on an individual arc. Our main goal is to study their properties and extend the TD arrival time functions to sequences of arcs.

Definition 4 (Sequence) *Let $\sigma_{ik}(n) \coloneqq \{n_j\}_{j=i}^{k}$ be an ordered sequence of nodes in $G_{\mathcal{RN}}(\mathcal{N}, \mathcal{A})$, where each consecutive pair (n_j, n_{j+1}) is connected by a nonempty set of arcs $\mathcal{A}_{j,j+1} = \{\, e \in \mathcal{A} : e = (n_j, n_{j+1}) \,\}$, and no arc in the opposite direction (n_{j+1}, n_j) for all $j \in \{i, i+1, \ldots, k-1\}$ is used in this sequence. The sequence σ_{ik} is called a* simple sequence *if $|\mathcal{A}_{j,j+1}| = 1$ for all $j \in \{i, i+1, \ldots, k-1\}$; otherwise, it is called a* complex sequence *if there exists at least one $j \in \{i, i+1, \ldots, k-1\}$ such that $|\mathcal{A}_{j,j+1}| > 1$. A simple sequence uniquely determines a path in the network, whereas a complex sequence allows for multiple arc choices between certain consecutive nodes.*

Let $\psi_{i,k}(t)$ denote the extended arrival time function at node n_k along the sequence $\sigma_{ik}(n)$ with $i < k$. Then, the Optimal Extended Arrival Time Function (OEATF) is defined as:

$$\psi_{i,k}^*(t) = \min_{e \in \mathcal{A}_{k-1,k}} \left\{ \phi_e \circ \psi_{i,k-1}^*(t) \right\}, \quad \forall i, k \in \sigma_{ik}(n) : i < k \,, \forall t \in \mathscr{H}. \tag{12}$$

In the case of a *simple* sequence, this reduces to:

$$\psi_{i,k}^*(t) = \phi_{i,i+1} \circ \phi_{i+1,i+2} \circ \cdots \circ \phi_{k-1,k}(t), \quad \forall i, k \in \sigma_{ik}(n) : i < k \,, \forall t \in \mathscr{H}, \tag{13}$$

where $\phi_{\alpha,\alpha+1}$ denotes the TD arrival time function associated with arc $(n_\alpha, n_{\alpha+1})$ for each $\alpha \in \{i, i+1, \ldots, k-1\}$.

While the above recursion defines $\psi_{i,k}^*(t)$ pointwise, our algorithms operate on *closed-form*, CPL representations. We therefore let $\Psi_{i,k}(t)$ denote the closed-form expression of $\psi_{i,k}(t)$, and use $\mathscr{L}\mathscr{E}$ to denote

Algorithm 1: Compute TD Travel Time $\theta_{ij}(t)$ on Arc (i, j)

Input: Piecewise constant speed function ν_{ij}; distance d_{ij}; time horizon $\mathscr{H} = [t_{\text{init}}, t_{\max}]$
Output: Travel time $\theta_{ij}(t)$
Initialization:
$t \leftarrow t_0$ // Departure time from node i
$d \leftarrow d_{ij}$ // Total distance between i and j
$\tau \leftarrow 0$ // Step 1: Find the time interval τ for departure time t
while $t \geq \bar{t}_\tau$ **do**
  $\tau \leftarrow \tau + 1$
// Step 2: Initialize travel distance and cumulative travel time
$\Delta d \leftarrow 0$ // Total distance traveled up to period τ
$\theta \leftarrow 0$ // Initialize cumulative travel time
$\delta d^\tau \leftarrow (\bar{t}_\tau - t) \times v_{ij}^\tau$ // Distance covered in period τ
// Step 3: Calculate travel time across intervals until destination is reached
while $\delta d^\tau < d_{ij}$ **do**
  $\Delta d \leftarrow \delta d^\tau$;
  $\theta \leftarrow \theta + (\bar{t}_\tau - t)$ // Add time spent in the interval
  $t \leftarrow \bar{t}_\tau$;
  $\tau \leftarrow \tau + 1$;
  $\delta d^\tau \leftarrow \Delta d + (\bar{t}_\tau - t) \times v_{ij}^\tau$;
// Step 4: Add remaining time to cover remaining distance
$\theta \leftarrow \theta + \left(\frac{d_{ij} - \Delta d}{v_{ij}^\tau}\right)$;
return $\theta_{ij}(t) \leftarrow \theta$

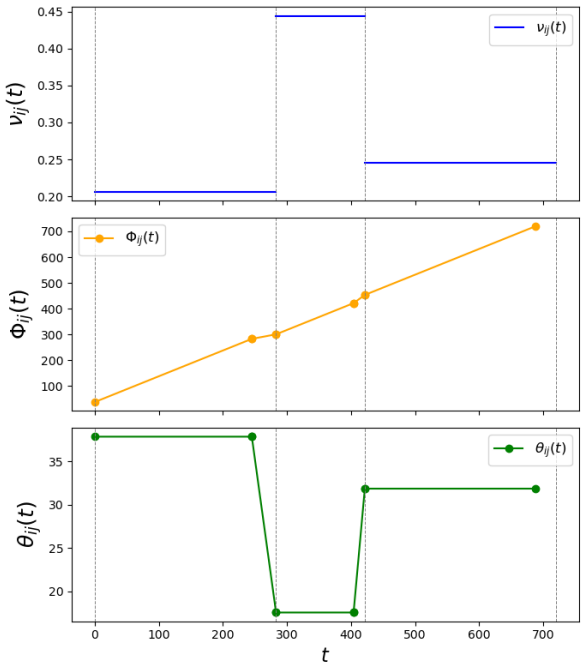

Figure 4: Speed, TD Travel Time, and TD Arrival Time

Algorithm 2: Find Breakpoints of $\theta_{ij}(t)$ and Construct Φ_{ij} on Arc (i, j)

Input: Piecewise constant speed function ν_{ij}; distance d_{ij}; time horizon $\mathscr{H} = [t_{\text{init}}, t_{\max}]$
Output: Closed-form of TD arrival function Φ_{ij} within the time horizon
Initialization:
$\mathscr{B}_1 \leftarrow \emptyset$ // Initialize set for breakpoints of ν_{ij}
$\mathscr{B}_2 \leftarrow \emptyset$ // Initialize set for breakpoints due to d_{ij} completion
$\mathscr{B}_3 \leftarrow \emptyset$ // Combine both sets
$\mathscr{B}_f \leftarrow [\,]$ // Initialize list for valid breakpoints in ascending order
$\Phi_{ij} \leftarrow \emptyset$ // Initialize closed-form arrival function $\Phi_{ij}(t)$
/* See Breakpoints of $\theta_{ij}(.)$, Prop. 3 */
Function MainFunction(*Find_Breakpoints*$(\nu_{ij}, d_{ij}, \mathscr{H})$):
  // Step 1: Identify breakpoints of $\nu_{ij}(t)$
  for *each interval* $\delta t_\tau := [\underline{t}_\tau, \bar{t}_\tau]$ *in* ν_{ij} **do**
    $\mathscr{B}_1 \leftarrow \mathscr{B}_1 \cup \{\bar{t}_\tau\}$;
  // Step 2: Breakpoints due to completion of d_{ij} within constant-speed intervals
  for *each interval* $\delta t_\tau := [\underline{t}_\tau, \bar{t}_\tau]$ *with constant speed* v_τ *in* ν_{ij} **do**
    $t_{\text{complete}} \leftarrow \underline{t}_\tau + \frac{d_{ij}}{v_\tau}$;
    if $t_{complete} \in \delta t_\tau$ **then**
      $\mathscr{B}_2 \leftarrow \mathscr{B}_2 \cup \{t_{\text{complete}}\}$;
  $\mathscr{B}_3 \leftarrow \mathscr{B}_1 \cup \mathscr{B}_2$ // Step 3: Combine and sort breakpoints
  Sort $\mathscr{B}_3$ in increasing order;
  // Step 4: Filter breakpoints based on the time horizon $\mathscr{H}$
  for *each* $t \in \mathscr{B}_3$ **do**
    Compute $\phi_{ij}(t) = t + \theta_{ij}(t)$ from Algorithm 1;
    if $t \in \mathscr{H}$ *and* $\phi_{ij}(t) \in \mathscr{H}$ **then**
      $\mathscr{B}_f \leftarrow \mathscr{B}_f \cup \{t\}$;
  // Step 5: Construct the closed-form arrival function $\Phi_{ij}(t)$
  for $i = 0$ *to* $|\mathscr{B}_f| - 2$ **do**
    $t_i \leftarrow \mathscr{B}_f[i]$;
    $t_{i+1} \leftarrow \mathscr{B}_f[i+1]$;
    $p_i \leftarrow \{t_i, \phi_{ij}(t_i)\}$;
    $q_i \leftarrow \{t_{i+1}, \phi_{ij}(t_{i+1})\}$;
    $s_i \leftarrow \{p_i, q_i\}$ // Define linear segment s_i
    $\Phi_{ij} \leftarrow \Phi_{ij} \cup \{s_i\}$;
  // Return the closed-form arrival function
  return $\Phi_{ij}(\cdot)$;

the lower-envelope operator. The closed-form recursion becomes:

$$\Psi^*_{i,k}(t) = \mathscr{LE}\left(\bigcup_{e\in\mathcal{A}_{k-1,k}}\left\{\Phi_e \circ \Psi^*_{i,k-1}(t)\right\}\right), \quad \forall i,k \in \sigma_{ik}(n) : i<k\,, \forall t \in \mathscr{H}. \tag{14}$$

For any origin $o \in \mathcal{N}$ and destination $d \in \mathcal{N}$ in the road network $G_{\mathcal{RN}} = (\mathcal{N}, \mathcal{A})$, the optimal extended arrival time function $\psi^*_{o,d}(t)$ satisfies:

$$\psi^*_{o,d}(t) = \min_{\pi\in\Pi(d)}\left\{\min_{e\in\mathcal{A}_{\pi,d}}\left(\phi_e \circ \psi^*_{o,\pi}(t)\right)\right\} = \min_{\substack{\pi\in\Pi(d)\\ e\in\mathcal{A}_{\pi,d}}} \phi_e\big(\psi^*_{o,\pi}(t)\big), \quad \forall t \in \mathscr{H}, \tag{15}$$

and its closed-form representation thus satisfies:

$$\Psi^*_{o,d}(t) = \mathscr{LE}\left(\bigcup_{\pi\in\Pi(d)}\left\{\bigcup_{e\in\mathcal{A}_{\pi,d}}\left\{\Phi_e \circ \Psi^*_{o,\pi}(t)\right\}\right\}\right), \tag{16}$$

where $\Pi(d) \subseteq \mathcal{N}$ is the set of all predecessor nodes of d, $\mathcal{A}_{\pi,d}$ is the set of arcs connecting π to d, and ϕ_e, and its closed-form Φ_e are the TD arrival time function associated with arc $e \in \mathcal{A}_{\pi,d}$.

3.1 Lower Envelope: An Algorithm Tailored to Time-dependent Functions

While existing algorithms—such as those proposed by Hershberger (1989)—can be used to compute lower envelopes, we develope a new algorithm specifically tailored to the structure of TD arrival/departure time functions. Leveraging the *point-line duality theorem* from computational geometry, our Algorithm 3 achieves the same asymptotic complexity of $\mathcal{O}(n \log n)$ while offering a significantly simpler and more practical implementation.

Algorithm 3: Compute Lower Envelope via Upper Convex Hull (LE-UCH)

Input: Two consecutive nodes $i, j \in \mathcal{N}$ on the road network $G_{\mathcal{RN}} = (\mathcal{N}, \mathcal{A})$; Φ_{ij}: a collection of closed-form arrival time functions, each represented as a set of line segments $s = \{p, q\}$ where p and q are the endpoints of segment s.
Output: The lower envelope $\mathscr{LE}(\Phi_{ij}(t))$ of all arrival time functions between i and j.
Initialization: $\mathcal{UH} \leftarrow \emptyset$; // Upper convex hull in the dual space
$\mathscr{B} \leftarrow \emptyset$; // Breakpoints of $\mathscr{LE}(\Phi_{ij}(t))$
$\mathscr{LE}(\Phi_{ij}(t)) \leftarrow \emptyset$; // Lower envelope
Priority queue $pQ \leftarrow \emptyset$; // Stores segments sorted by decreasing slope (dual x-coordinate)
foreach $s \in \Phi_{ij}$ **do**
  Insert s into pQ;
Add the first two segments from pQ to $\mathcal{UH}$;
while *pQ is not empty* **do**
  while $|\mathcal{UH}| \geq 2$ *and Orientation*$(\mathcal{UH}[-2], \mathcal{UH}[-1], pQ.front()) \neq CCW$ **do**
    Remove the last segment from $\mathcal{UH}$; // Maintain convexity
  Append pQ.front() to $\mathcal{UH}$; Remove pQ.front();
$\mathscr{B}$.append($\mathcal{UH}[0].p.x$) ; // First breakpoint
for $s = 1$ *to* $|\mathcal{UH}| - 1$ **do**
  Compute intersection point $p =$ Intersect$(\mathcal{UH}[s], \mathcal{UH}[s+1])$; $\mathscr{B}$.append$(p.x)$;
$\mathscr{B}$.append($\mathcal{UH}[-1].q.x$) ; // Last breakpoint
for $i = 1$ *to* $|\mathscr{B}| - 1$ **do**
  Append segment$(\mathscr{B}[i], \mathscr{B}[i+1])$ to $\mathscr{LE}(\Phi_{ij}(t))$;
return $\mathscr{LE}(\Phi_{ij}(t))$;

Theorem 3 (Lower envelope via upper convex hull) *Let $\Phi_{ij}(t)$ be the collection of closed-form TD arrival time functions associated with all arcs connecting node i to node j, represented as a collection of line segments. Then: (1) The lower envelope $\mathscr{LE}(\Phi_{ij}(t))$ is obtained by a counterclockwise traversal of the upper convex hull $\mathcal{UH}(\mathscr{P})$ of the corresponding dual points. (2) $\mathscr{LE}(\Phi_{ij}(t))$ is CPL, and its segments are ordered from left to right by increasing slope. (3) The lower envelope can be computed in $\mathcal{O}(n \log n)$ operations, where $n = |\Phi_{ij}|$ is the total number of input segments.*

By duality, a symmetric relationship holds for latest departure time functions, derived via upper envelopes of inverse arrival time functions. This formulation supports backward DP and feasibility checks. (See Lemmas 3-4), and Theorem 9 in Appendix.

3.2 Efficient Algorithms for the Time-Dependent Quickest Path Problem (TDQPP)

We now compute closed-form *extended arrival* functions from a source to every node of a road-level subnetwork (e.g., Figure 2). The Bellman–Ford-variant method of Vidal et al. (2021) builds these functions by repeated relaxations and lower-envelope operations; it performs $\mathcal{O}(|\mathcal{N}|\,|\mathcal{A}|)$ function updates overall. Both proposed algorithms - TDQPFA and TDDA - use the same CPL *operations* (composition and lower envelope), but reduce the number of relaxations dramatically. Throughout, each CPL update on a Ψ-profile with $\mathscr{B}$ breakpoints costs $\mathcal{O}(\log \mathscr{B})$ time. At the end of either algorithm we obtain the final optimal extended arrival functions $\{\Psi^*_{iy}\}_{y\in\mathcal{N}}$.

Algorithm 4: TDQPFA

Input: $G_{\mathcal{RN}} = (\mathcal{N}, \mathcal{A})$; source i; horizon $\mathscr{H} = [t_{\text{init}}, t_{\max}]$
Output: Closed-form arrival-time profiles $\{\Psi^*_{iy}\}_{y\in\mathcal{N}}$

foreach $j \in \mathcal{N}$ **do**
  $\Psi'_{ij} \leftarrow \Psi^*_{ij} \leftarrow \begin{cases} f_{\text{ID}}, & j = i \\ f_\infty, & j \neq i \end{cases}$
$\mathcal{Q} \leftarrow \{i\}$; $\text{inQ} \leftarrow \{i\}$
while $\mathcal{Q} \neq \emptyset$ **do**
  $x \leftarrow \text{pop}(\mathcal{Q})$; $\text{inQ} \leftarrow \text{inQ} \setminus \{x\}$
  foreach $(x, y) \in \mathcal{A}$ **do**
    if $x = i$ **then**
      $\Psi'_{iy} \leftarrow \Phi_{xy}$ `// Alg. 2`
    else
      $\Psi'_{iy} \leftarrow \Psi'_{iy} \cup \{\, \Phi_{xy} \circ \Psi^*_{ix} \,\}$ `// Alg. 6`
    $\Psi'_{iy} \leftarrow \mathscr{LE}(\Psi'_{iy})$ `// Alg. 3`
    if $\Psi'_{iy} \neq \Psi^*_{iy}$ **then**
      $\Psi^*_{iy} \leftarrow \Psi'_{iy}$
      if $y \notin \text{inQ}$ **then**
        $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{y\}$; $\text{inQ} \leftarrow \text{inQ} \cup \{y\}$
return $\{\Psi^*_{iy}\}_{y\in\mathcal{N}}$

Algorithm 5: TDDA

Input: $G_{\mathcal{RN}} = (\mathcal{N}, \mathcal{A})$; source i; horizon $\mathscr{H} = [t_{\text{init}}, t_{\max}]$
Output: Closed-form arrival-time profiles $\{\Psi^*_{iy}\}_{y\in\mathcal{N}}$

foreach $j \in \mathcal{N}$ **do**
  $\Psi'_{ij} \leftarrow \Psi^*_{ij} \leftarrow \begin{cases} f_{\text{ID}}, & j = i \\ f_\infty, & j \neq i \end{cases}$
$\mathcal{Q} \leftarrow$ priority queue; $\mathcal{Q}.\text{push}(\Psi^*_{ii}(t_{\text{init}}),\, i)$;
$\text{visited}[j] \leftarrow \text{false}\ \forall j$
while $\mathcal{Q} \neq \emptyset$ **do**
  remove (k, x) with smallest key k from $\mathcal{Q}$
  if $\text{visited}[x]$ **then**
    continue;
  $\text{visited}[x] \leftarrow \text{true}$
  foreach $(x, y) \in \mathcal{A}$ **do**
    if $x = i$ **then**
      $\Psi'_{iy} \leftarrow \Phi_{xy}$ `// Alg. 2`
    else
      $\Psi'_{iy} \leftarrow \Psi'_{iy} \cup \{\, \Phi_{xy} \circ \Psi^*_{ix} \,\}$ `// Alg. 6`
    $\Psi'_{iy} \leftarrow \mathscr{LE}(\Psi'_{iy})$ `// Alg. 3`
    if $\Psi'_{iy} \neq \Psi^*_{iy}$ **then**
      $\Psi^*_{iy} \leftarrow \Psi'_{iy}$
      $\text{newKey} \leftarrow \Psi^*_{iy}(t_{\text{init}})$;
      $\mathcal{Q}.\text{push}(\text{newKey},\, y)$
return $\{\Psi^*_{iy}\}_{y\in\mathcal{N}}$

3.2.1 TDQPFA: Adaptive label relaxation with queue management

Algorithm 4 adapts SPFA to TD networks, maintaining a queue $\mathcal{Q}$ of nodes whose labels can still improve. Let ρ denote the average number of successful decreases per node (typically small on road networks). Then, its time complexity is $\mathcal{O}(|\mathcal{A}|\,\rho) \cdot \mathcal{O}(\log \mathscr{B})$, with worst-case $\mathcal{O}(|\mathcal{N}|\,|\mathcal{A}|) \cdot \mathcal{O}(\log \mathscr{B})$, and $\mathcal{O}(|\mathcal{N}| + |\mathcal{A}|)$ space requirement.

3.2.2 TDDA: Priority-based relaxation and efficient path selection

Algorithm 5 augments TDQPFA with a priority queue and is label-setting under FIFO: once a node is popped, its arrival function is final. Its complexity is $\mathcal{O}\big((|\mathcal{N}| + |\mathcal{A}|) \log |\mathcal{N}|\big) \cdot \mathcal{O}(\log \mathscr{B})$, with space requirement of $\mathcal{O}(|\mathcal{N}| + |\mathcal{A}|)$.

	Sparse $\big(|\mathcal{A}| = \Theta(|\mathcal{N}|)\big)$	**Dense** $\big(|\mathcal{A}| = \Theta(|\mathcal{N}|^2)\big)$
BF variant (Vidal et al. 2021)	$\mathcal{O}\big(|\mathcal{N}|^2\big) \cdot \mathcal{O}(\log \mathscr{B})$	$\mathcal{O}\big(|\mathcal{N}|^3\big) \cdot \mathcal{O}(\log \mathscr{B})$
TDQPFA (ours)	$\mathcal{O}\big(|\mathcal{N}|\,\rho\big) \cdot \mathcal{O}(\log \mathscr{B})$	$\mathcal{O}\big(|\mathcal{N}|^2\,\rho\big) \cdot \mathcal{O}(\log \mathscr{B})$
TDDA (ours)	$\mathcal{O}\big(|\mathcal{N}| \log |\mathcal{N}|\big) \cdot \mathcal{O}(\log \mathscr{B})$	$\mathcal{O}\big(|\mathcal{N}|^2 \log |\mathcal{N}|\big) \cdot \mathcal{O}(\log \mathscr{B})$

Table 1: Time bounds for computing closed-form arrival-time profiles $\{\Psi^*_{iy}\}_{y\in\mathcal{N}}$ from a source i. ρ: average successful relaxations per node; $\mathscr{B}$: CPL breakpoints per profile. All use $\mathcal{O}(|\mathcal{N}|+|\mathcal{A}|)$ space.

Compared with the BF variant ($\mathcal{O}(|\mathcal{N}|\,|\mathcal{A}|) \cdot \mathcal{O}(\log \mathscr{B})$), TDDA improves to $\mathcal{O}((|\mathcal{N}|+|\mathcal{A}|) \log |\mathcal{N}|) \cdot \mathcal{O}(\log \mathscr{B})$, and TDQPFA to $\mathcal{O}(|\mathcal{A}|\,\rho) \cdot \mathcal{O}(\log \mathscr{B})$; on sparse (resp. dense) networks this yields an asymptotic factor of $\Omega(|\mathcal{N}|/\log|\mathcal{N}|)$ (resp. $\Omega(|\mathcal{N}|/\log|\mathcal{N}|)$), with larger gains in practice because $\rho \ll |\mathcal{N}|$, matching the empirical speedups in Section 7.

3.3 Requests and Presence of Time Windows

In the previous sections, we characterized the properties of TD functions over the time horizon $\mathscr{H}$. We now extend this analysis to consider the presence of time windows associated with customer locations. In particular, we investigate how time windows restrict the domains and ranges of these functions, which is crucial for feasibility checks during route planning. Throughout this section, $[a_i, b_i]$ denotes the time window at customer location i. We assume a service time of zero at all locations, allowing for waiting but prohibiting delay, i.e., the service must begin before b_i. Finally, our goal is to efficiently extend optimal envelope of the extended departure time functions—corresponding to that of the so-called *dominant path*.

3.3.1 Time Windows and Composite Function Properties

We now formalize the impact of time windows on the TD functions by introducing the time-window-constrained arrival and departure functions, analyzing their properties, and providing constructive formulations for feasibility analysis.

Define $\widetilde{\phi}_{ij} : [a_i, b_i] \to [a_j, b_j] \cup \{\infty\}$ as the TD arrival time function at node j on arc (i, j) under time window constraints. Its feasibility, domain, and range are specified as follows: if $i = j$, then $\widetilde{\phi}_{ij}(.)$ is

infeasible if and only if $a_i > b_j$. Otherwise:

$$\text{Range}(\widetilde{\phi}_{ij}) = \text{Dom}(\widetilde{\phi}_{ij}) = [a_i, \min\{b_i, b_j\}] \tag{17}$$

If $i \neq j$, then $\widetilde{\phi}_{ij}(.)$ is *infeasible* if and only if:

$$\phi_{ij}(a_i) > b_j \quad \text{or} \quad \phi_{ij}^{-1}(b_j) < a_i. \tag{18}$$

If *feasible*:

$$\begin{aligned} \text{Dom}(\widetilde{\phi}_{ij}) &= \left[a_i, \min\{b_i, \phi_{ij}^{-1}(b_j)\}\right], \\ \text{Range}(\widetilde{\phi}_{ij}) &= \left[\phi_{ij}(a_i), \min\{\phi_{ij}(b_i), b_j\}\right]. \end{aligned} \tag{19}$$

The optimal time-window-constrained arrival time $\widetilde{\phi}_{ij}^*$ on a given arc $e = (i, j)$ corresponds to the earliest possible arrival time at node j when departing from node i within its time window, and is given by:

$$\widetilde{\phi}_{ij}^* = \min_{t \in [a_i, b_i]} \widetilde{\phi}_{ij}(t). \tag{20}$$

The optimal value is uniquely attained at $t = a_i$; that is, $\widetilde{\phi}_{ij}^* = \phi_{ij}(a_i)$. If multiple parallel arcs exist between nodes i and j, then:

$$\widetilde{\phi}_{ij}^* = \min_{e \in \mathcal{A}_{i,j}} \widetilde{\phi}_e(a_i). \tag{21}$$

Let $\omega_{ij}(t) : [a_i, b_i] \to [a_j, b_j]$ denote the TD departure time function on arc (i, j). For a single departure time query $t \in [a_i, b_i]$, $\omega_{ij}(t)$ returns the corresponding feasible departure time at node j. Its domain and range are given by:

$$\text{Dom}\left(\omega_{ij}\right) = \text{Dom}\left(\widetilde{\phi}_{ij}\right) = \left[a_i, \min\left\{b_i, \phi_{ij}^{-1}(b_j)\right\}\right], \tag{22}$$

$$\text{Range}\left(\omega_{ij}\right) = \left[\widetilde{\phi}_{ij}^*, b_j\right]. \tag{23}$$

The CPL quasi-monotonic function $\omega_{ij}(t)$ is feasible if and only if $\widetilde{\phi}_{ij}(t)$ is feasible, and if feasible, then the optimal departure time at node j,

$$\omega_{ij}^* \coloneqq \min_{t \in \text{Dom}(\omega_{ij})} \omega_{ij}(t), \tag{24}$$

is unique. However, the minimizer may not be, i.e., multiple values of t may attain the minimum.

Let $\gamma_j : \mathscr{H} \to [t_{\text{init}}, b_j] \cup \{\infty\}$ define the time-window ready time function, which specifies the earliest feasible service completion time at node j given a tentative arrival time $t \in \mathscr{H}$:

$$\gamma_j(t) = \begin{cases} a_j, & t \leq a_j, \\ t, & a_j < t \leq b_j, \\ \infty, & t > b_j. \end{cases} \tag{25}$$

Then, the departure time function $\omega_{ij}(t)$ can be expressed by the time-window ready time function, and therefore, admits two equivalent representations:

$$\omega_{ij}(t) = \max\left\{a_j, \widetilde{\phi}_{ij}(t)\right\} = (\gamma_j \circ \phi_{ij})(t), \quad \forall t \in [a_i, b_i], \tag{26}$$

The first form models waiting explicitly, while the second expresses it via composition, which is particularly advantageous in algorithmic implementations.

Let $\Gamma_j : \mathscr{H} \to [t_{\text{init}}, b_j] \cup \{\infty\}$ denote the closed-form representation of $\gamma_j(t)$. The function Γ_j is CPL with exactly two segments:

$$\Gamma_j \coloneqq \left\{ \left\{ (t_{\text{init}}, a_j), (a_j, a_j) \right\}, \left\{ (a_j, a_j), (b_j, b_j) \right\} \right\}. \tag{27}$$

The closed-form representation of the departure time function on a single arc could hence be defined as:

$$\Omega_{ij}(t) = (\Gamma_j \circ \Phi_{ij})(t) = (\Gamma_j \circ \widetilde{\Phi}_{ij})(t), \quad \forall t \in [a_i, b_i] \tag{28}$$

We now characterize the extended departure time function and its closed-form representation, which form the cornerstone of our DP algorithms. Let $\sigma_{i,k}$ be a sequence of nodes from i to k, and let $f^*_{i,k-1}(t)\ ,\ \forall t \in [a_i, b_i]$ be the Optimal Extended Departure Time Function (OEDTF) at the predecessor node $k-1$. Then,

$$f^*_{i,k}(t) \ = \ \min_{e \in \mathcal{A}_{k-1,k}} \left(\gamma_k \circ \phi_e \circ f^*_{i,k-1}(t)\right) \ = \ \min_{e \in \mathcal{A}_{k-1,k}} \omega_e(f^*_{i,k-1}(t)). \tag{29}$$

Its closed-form representation is thus defined as:

$$\mathscr{F}^*_{i,k}(t) = \mathscr{L}\mathscr{E}\left(\bigcup_{e \in \mathcal{A}_{k-1,k}} \left\{\Gamma_k \circ \Phi_e \circ \mathscr{F}^*_{i,k-1}(t)\right\}\right) = \mathscr{L}\mathscr{E}\left(\bigcup_{e \in \mathcal{A}_{k-1,k}} \left\{\Omega_e \circ \mathscr{F}^*_{i,k-1}(t)\right\}\right), \tag{30}$$

Let $o, d \in \mathcal{N}$ in the time-dependent network $G_{\mathcal{RN}} = (\mathcal{N}, \mathcal{A})$. Suppose $f^*_{o,\pi}$ is known for $\pi \in \Pi(d)$. Then,

$$f^*_{o,d}(t) = \min_{\pi \in \Pi(d)} \left(\min_{e \in \mathcal{A}_{\pi,d}} \left(\gamma_d \circ \phi_e \circ f^*_{o,\pi}(t)\right)\right) = \min_{\pi \in \Pi(d)} \left(\min_{e \in \mathcal{A}_{\pi,d}} \left(\omega_e(f^*_{o,\pi}(t))\right)\right). \tag{31}$$

Its closed-form representation is:

$$\mathscr{F}^*_{o,d}(t) = \mathscr{L}\mathscr{E}\left(\bigcup_{\pi \in \Pi(d)} \left\{ \bigcup_{e \in \mathcal{A}_{\pi,d}} \left\{\Gamma_k \circ \Phi_e \circ \mathscr{F}^*_{o,\pi}(t)\right\}\right\}\right) = \mathscr{L}\mathscr{E}\left(\bigcup_{\pi \in \Pi(d)} \left\{ \bigcup_{e \in \mathcal{A}_{\pi,d}} \left\{\Omega_e \circ \mathscr{F}^*_{o,\pi}(t)\right\}\right\}\right), \tag{32}$$

where $\Pi(d)$ denotes the set of predecessor nodes of d, $\mathcal{A}_{\pi,d}$ the set of arcs from π into d, γ_d the ready-time function at d, ϕ_e the arrival-time function on arc e, $\omega_e = \gamma_d \circ \phi_e$ the corresponding departure-time function, and Ω_e its closed-form representation. Hence, the optimal departure time from node o is computed through

Algorithm 6: Feasibility Check and Composite Function Construction

Input: Two piecewise linear, non-decreasing functions represented as vectors of segments: $f_1 = \{S_1^{(1)}, \ldots, S_n^{(1)}\}$ (inner function) and $f_2 = \{S_1^{(2)}, \ldots, S_m^{(2)}\}$ (outer function).
Output: Feasibility status and composite function $f_2 \circ f_1$ (if feasible).
Initialization:
$\mathscr{B}_{f_{\text{composite}}} \leftarrow \emptyset$ // Initialize set for breakpoints of $f_2 \circ f_1$
$\mathcal{S}_{\text{composite}} \leftarrow \emptyset$ // closed-form expression of composition $f_2 \circ f_1$
INF_SEG_VEC $\leftarrow \{\{(T_{\text{init}}, T_{\max}), (T_{\max}, T_{\max})\}\}$ // Default infeasibility marker
isfeas_composite $\leftarrow$ (false, INF_SEG_VEC)
if f_1 *or* f_2 *is empty* **then**
  return *isfeas_composite*
if $f_1[n].q_y \leq f_2[0].p_x$ **then**
  return $(\textit{true}, \{\{(f_1[0].p_x, f_2[0].p_y), (f_1[n].q_x, f_2[0].p_y)\}\})$
// Extract breakpoints from both functions
$\mathscr{B}_{f_1} \leftarrow$ get_breakpoints_from_line_segments(f_1) // Breakpoints from f_1
$\mathscr{B}_{f_2} \leftarrow$ get_breakpoints_from_line_segments(f_2) // Breakpoints from f_2
// Determine domain of $f_2 \circ f_1$-See Lemma 5, and Propositions 9-10
$\min_x \leftarrow \max(f_1[0].p_y, f_2[0].p_x)$ // Lower bound of domain
$\max_x \leftarrow \min(f_1[n].q_y, f_2[m].q_x)$ // Upper bound of domain
// Compute breakpoints of $f_2 \circ f_1$
foreach $p \in \mathscr{B}_{f_1}$ **do**
  if $p_y \in [\min_x, \max_x]$ **then**
    $y \leftarrow$ get_y_coordinate(f_2, p_y) **if** $y \in [\min_y, \max_y]$ **then**
      $\mathscr{B}_{\text{composite}} \leftarrow \mathscr{B}_{\text{composite}} \cup (p_x, y)$
foreach $q \in \mathscr{B}_{f_2}$ **do**
  $x' \leftarrow$ get_x_coordinate(f_1, q_x) **if** $x' \in [f_1[0].p_x, f_1[n].q_x]$ **then**
    $\mathscr{B}_{\text{composite}} \leftarrow \mathscr{B}_{\text{composite}} \cup (x', q_y)$
if $\mathscr{B}_{composite} = \emptyset$ **then**
  return *isfeas_composite*
// Sort and deduplicate breakpoints
$\mathscr{B}_{\text{composite}} \leftarrow$ sort_and_remove_duplicates($\mathscr{B}_{\text{composite}}$)
// Construct composite function segments
for $i = 1$ ***to*** $|\mathscr{B}_{composite}| - 1$ **do**
  $\mathcal{S}_{\text{composite}} \leftarrow \mathcal{S}_{\text{composite}} \cup (\mathscr{B}_{\text{composite}}[i], \mathscr{B}_{\text{composite}}[i+1])$
// Merge adjacent segments if needed
$\mathcal{S}_{\text{composite}} \leftarrow$ remove_redundant_segments($\mathcal{S}_{\text{composite}}$)
return $(\textit{true}, \mathcal{S}_{composite})$

Equation (33), or alternatively, Equation (34):

$$t^* = \underset{t\in[a_o,b_o]}{\arg\min} \left(\min_{\pi\in\Pi(d)} \left(\min_{e\in\mathcal{A}_{\pi,d}} \left(\gamma_d \circ \phi_e \circ f^*_{o,\pi}(t) \right) \right) \right) = \underset{t\in[a_o,b_o]}{\arg\min} \left(\min_{\pi\in\Pi(d)} \left(\min_{e\in\mathcal{A}_{\pi,d}} \left(\omega_e\big(f^*_{o,\pi}(t)\big) \right) \right) \right) \quad (33)$$

$$t^* = \underset{t\in[a_o,b_o]}{\arg\min} \left(\mathscr{L}\mathscr{E} \left(\bigcup_{\pi\in\Pi(d)} \left\{ \bigcup_{e\in\mathcal{A}_{\pi,d}} \left\{ \Omega_e \circ \mathscr{F}^*_{o,\pi}(t) \right\} \right\} \right) \right). \quad (34)$$

Notice that computing the optimal point $(t^*, f^*_{o,d})$ involves minimization over a continuous time span, while computing this optimal in form of $(t^*, \mathscr{F}^*_{o,d})$, involves (i) extending optimal departure time envelopes, (ii) a linear/binary search over the breakpoints of the final envelope.

3.4 Subnetwork to Logical Arc Mapping

Recall that $\Omega_{ij}(\cdot)$ is the closed-form departure-time function on an individual arc, $\mathscr{F}_{o,d}(\cdot)$ its extension over sequences, and $\Psi^*_{o,d}(\cdot)$ the closed-form extended arrival-time function in Equation (16). Consider a subnetwork (Figure 2) connecting two stops. With time-windowed requests, each stop inherits windows, whereas interior junctions do not. We abstract each such subnetwork $\mathscr{G}_{i,j} \subseteq G_{\mathcal{RN}}$ by a *logical* arc (i,j) induced by the road network: for stop pairs (i,j) in $\mathcal{E}_{\log} := \{(i,j) \in \mathcal{S} \times \mathcal{S} : \text{ there exists a directed path } i \rightsquigarrow j \text{ in } G_{\mathcal{RN}}\}$, we set $\Omega_{i,j} := \Gamma_j \circ \Psi^*_{i,j}$. A route sequence (Definition 1) can therefore be viewed as a complex sequence (Definition 4) in a lifted graph where consecutive stops are linked by logical arcs. Using Algorithms 4–5, we compute $\Psi^*_{i,j}$, which enables precomputation of $\Omega_{i,j}$ for customer locations. The closed-form of the OD optimal extended departure time function then can be written as:

$$\mathscr{F}^*_{o,d}(t) = \mathscr{L}\mathscr{E}\left(\bigcup_{\pi \in \Pi(d):(\pi,d)\in\mathcal{E}_{\log}} \{\Omega_{\pi,d} \circ \mathscr{F}^*_{o,\pi}(t)\}\right), \tag{35}$$

which is the logical-arc version of Equation (32). This *precomputation* is further discussed in Section 5.2. For a formal statement and proof of the equivalence, see Proposition 7 (Equivalence of the Logical-Arc Abstraction) in the Appendix.

Computational benefit. This abstraction lets us (i) precompute $\Psi^*_{i,j}$ once per stop pair (network-only), then (ii) impose windows by $\Omega_{i,j} = \Gamma_j \circ \Psi^*_{i,j}$. For *static* requests, cache $\Omega_{i,j}$; for *dynamic/online* settings, keep $\Psi^*_{i,j}$ and update only Γ_j. Equation (32) can thus be evaluated over logical arcs with fewer compositions and effective reuse of cached functions.

In this section, we introduced a family of functions whose closed-form expressions arise from compositions of CPL (monotonically) nondecreasing functions. Algorithm 6 formalizes these compositions while ensuring feasibility through domain–range alignment (also, see Lemma 5, Propositions 9-10). We refer the interested reader to Appendix D, for further details about the properties of TD functions.

This foundation enables the transition to our Optimization Phase for the GRSP-TD$_{\mathcal{RN}}$, presented in the next section.

4 Optimization Phase

Building on the TD function modeling in Section 3, we now formulate the recursive computation of optimal extended departure time functions between any two clusters in a route. This formulation forms the cornerstone of our DP algorithms.

Theorem 4 (Optimal cluster-to-cluster departure time functions) *In a route sequence $\sigma(r)$ (Def. 1), let $\mathscr{F}^{l,s}_{\nu(i),\nu(k)}(.)$ denote the closed-form expression of the optimal departure time function from any node indexed l in the origin cluster $\nu(i)$ to any node indexed s in the destination cluster $\nu(k)$, where $i \leq k$. These functions can be recursively computed as:*

$$\mathscr{F}^{l,s}_{\nu(i),\nu(k)}(t) = \mathscr{L}\mathscr{E}\left(\bigcup_{p=0}^{|\nu(k-1)|-1} \left\{\mathscr{F}^{p,s}_{\nu(k-1),\nu(k)} \circ \mathscr{F}^{l,p}_{\nu(i),\nu(k-1)}(t)\right\}\right), \tag{36}$$

for all $l \in \{0, 1, \ldots, |\nu(i)| - 1\}$ and $s \in \{0, 1, \ldots, |\nu(k)| - 1\}$, where l and s are node indices in clusters $\nu(i)$ and $\nu(k)$, respectively, and $\mathscr{LE}(\cdot)$ denotes the lower envelope operator. Here, $\mathscr{F}^{p,s}_{\nu(k-1),\nu(k)} = \Omega_{p,s}$ by the abstraction (Def. 6 and Prop. 7).

5 Optimization Phase

Building on the TD function modeling in Section 3, we now formulate the recursive computation of optimal extended departure time functions between any two clusters in a route. This formulation forms the cornerstone of our DP algorithms.

Theorem 5 (Optimal cluster-to-cluster departure time functions) *In a route sequence $\sigma(r)$ (Def. 1), let $\mathscr{F}^{l,s}_{\nu(i),\nu(k)}(.)$ denote the closed-form expression of the optimal departure time function from any node indexed l in the origin cluster $\nu(i)$ to any node indexed s in the destination cluster $\nu(k)$, where $i \leq k$. These functions can be recursively computed as:*

$$\mathscr{F}^{l,s}_{\nu(i),\nu(k)}(t) = \mathscr{LE}\left(\bigcup_{p=0}^{|\nu(k-1)|-1} \left\{\mathscr{F}^{p,s}_{\nu(k-1),\nu(k)} \circ \mathscr{F}^{l,p}_{\nu(i),\nu(k-1)}(t)\right\}\right), \tag{37}$$

for all $l \in \{0, 1, \ldots, |\nu(i)| - 1\}$ and $s \in \{0, 1, \ldots, |\nu(k)| - 1\}$, where l and s are node indices in clusters $\nu(i)$ and $\nu(k)$, respectively, and $\mathscr{LE}(\cdot)$ denotes the lower envelope operator. Here, $\mathscr{F}^{p,s}_{\nu(k-1),\nu(k)} = \Omega_{p,s}$ by the abstraction (Def. 6 and Prop. 7).

This theorem states that the optimal departure time function between two clusters is obtained by composing the optimal functions of their respective predecessors and selecting the minimum over all possible connecting logical arcs—each representing a subnetwork (e.g., Figure 2)—implemented via the lower envelope.

The optimal route duration of the route sequence $\sigma(r)$ is thus defined as:

$$\Delta^*_{\sigma(r)} \coloneqq \min_{t \in \mathscr{H}} \left(\mathscr{F}^{0,0}_{\nu(o),\nu(d)}(t) - t\right) \tag{38}$$

The optimal start time—a solution to the $\mathcal{NP}$-hard OSTP—is hence derived from:

$$t^* \coloneqq \arg\min_{t \in \mathscr{H}} \Delta_{\sigma(r)}(t) \coloneqq \arg\min_{t \in \mathscr{H}} \left(\mathscr{F}^{0,0}_{\nu(o),\nu(d)}(t) - t\right) \tag{39}$$

The minimizer t^* could be found in a linear or binary search, and it might be *not* unique (see also Remark 2, for further elaborations).

The following theorem formally bridges our recursive formulation in Theorem 5 (Equation (37)) with the value function in our proposed generic Bellman-Ford DP formulation (Equations (5)–(6)).

Theorem 6 (Equivalence to Bellman-Ford DP under uniform time cost) *Suppose both travel and waiting times incur identical unit costs as outlined in Remark 1, then, t^* defined in Equation* (39) *also solves the DP model in Equations* (5)–(6)*.*

We next describe the components of the optimization framework.

5.1 Optimization Components

The optimization layer has two main components: (i) *scheduling* for a fixed route, and (ii) *solution improvement* that changes the route sequence(s).

Scheduling. Given a route r with sequence $\sigma(r)$ (Definition 1), the goal is to minimize the total route duration $\Delta^*_{\sigma(r)}$ in (38) by choosing the optimal depot start time t^* in (39). This is achieved by DP algorithms that propagate lower envelopes of departure-time functionsfrom depot to depot along the route, in accordance with the recursion (37) (Theorem 5). Once the depot-to-depot function is obtained, we read off t^*; optimal in-cluster node choices are then recovered in post-processing (Section 6).

Solution improvement. Solution improvement concerns *cluster sequencing*. In a single-vehicle setting it amounts to (i) selecting the subset of requests to serve and (ii) ordering their clusters to obtain a sequence $\sigma(r)$ with minimal duration. With a fleet, both *intra-route* and *inter-route* moves are used: e.g., request insertions, swaps between two routes, and concatenation/k-exchange of subsequences. Each move produces one or more modified route sequences whose costs must be evaluated to decide acceptance. Naïvely, re-running a full DP scheduler per candidate route is prohibitive—especially in the GRSP-TD$_{\mathcal{RN}}$, where each request has multiple pickup and delivery candidates and logical arcs correspond to road-level subnetworks.

To enable fast move evaluation, we cache and reuse computed cluster-to-cluster departure-time functions in a dedicated data structure (Table 2). This avoids redundant recomputation while preserving exactness; details are given in Section 5.4. Our experiments instantiate these ideas within a sequential construction heuristic, using insertion as the illustrative move (Section 7).

Finally, we introduce execution-mode and notation conventions used to present and benchmark the schedulers in Section 5.2.

5.2 Notation and Execution Modes for DP Algorithms

Following Definition 1, for a given route r, let its sequence be $\sigma(r) = \{\nu(0), \nu(1), \ldots, \nu(2L+1)\}$, so that $|\sigma(r)| = 2L+2$. Define $\mathscr{R} = |\sigma(r)|$, $n = \max_i|\nu(i)|$, $\mathscr{B} = \max_{e=(i,j)\in\mathcal{A}_{\sigma(r)}} \mathscr{B}_{\phi_{ij}}$, and $(j-i) =$ hops for a single $(\nu(i), \nu(j))$ query. $\mathscr{F}^{l,s}_{\nu(i),\nu(k)}$ represent optimal departure time function from node $l \in \nu(i)$ to node $s \in \nu(k)$, as defined in Equation (37).

Departure Time Matrix. Define the upper-triangular *departure time matrix* $\boldsymbol{\mathscr{F}}$, whose (i,j)-block stores a $|\nu(i)| \times |\nu(j)|$-matrix containing all node-to-node closed-form departure time functions between origin–destination clusters $\nu(i)$ and $\nu(j)$. *Formally*, for $0 \le i \le j \le \mathscr{R}-1$, set $\boldsymbol{\mathscr{F}}_{\nu(i),\nu(j)} = \bigcup_{p=0}^{|\nu(i)|-1} \bigcup_{s=0}^{|\nu(j)|-1} \{\mathscr{F}^{p,s}_{\nu(i),\nu(j)}\}$. All lower-triangular entries ($i > j$) are omitted. In particular, on the diagonal $\boldsymbol{\mathscr{F}}_{\nu(i),\nu(i)} = \boldsymbol{\mathscr{F}}_{ID}$ is simply the vector of identity functions (one per node). Table 2 illustrates how these blocks are laid out. Because only the upper-triangular blocks of $\boldsymbol{\mathscr{F}}$ are used, one can collapse the (i,j) index pair into a single index:

$$k = \text{F\_INDEX}(i, j, \mathscr{R}) = i \cdot \mathscr{R} + j - \tfrac{i\,(i+1)}{2},$$

$\boldsymbol{\mathscr{F}}$	$\nu(0)$	$\nu(1)$	$\nu(2)$	$\cdots$	$\nu(\mathscr{R}-1)$
$\nu(0)$	$\boldsymbol{\mathscr{F}_{ID}}$	$\boldsymbol{\mathscr{F}_{\nu(0),\nu(1)}}$	$\boldsymbol{\mathscr{F}_{\nu(0),\nu(2)}}$	$\cdots$	$\boldsymbol{\mathscr{F}_{\nu(0),\nu(\mathscr{R}-1)}}$
$\nu(1)$	-	$\boldsymbol{\mathscr{F}_{ID}}$	$\boldsymbol{\mathscr{F}_{\nu(1),\nu(2)}}$	$\cdots$	$\boldsymbol{\mathscr{F}_{\nu(1),\nu(\mathscr{R}-1)}}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$\nu(\mathscr{R}-1)$	-	-	-	$\cdots$	$\boldsymbol{\mathscr{F}_{ID}}$

Table 2: Matrix of Cluster-to-Cluster Optimal Departure Time Functions $\boldsymbol{\mathscr{F}} = \bigcup_{i=0}^{\mathscr{R}-1} \bigcup_{j=i}^{\mathscr{R}-1} \{\boldsymbol{\mathscr{F}}_{\nu(i),\nu(j)}\}$

thereby mapping $\boldsymbol{\mathscr{F}}_{\nu(i),\nu(j)} \mapsto \boldsymbol{\mathscr{F}}_k$. Since there are only $\frac{\mathscr{R}(\mathscr{R}+1)}{2}$ nonempty blocks instead of $\mathscr{R}^2$, this yields almost a 50% reduction in storage. If each cluster has at most n nodes, the naive storage cost would be $\mathcal{O}(\mathscr{R}^2\, n^2)$; after compression it becomes $\mathcal{O}(\frac{\mathscr{R}(\mathscr{R}+1)}{2}\, n^2)$. This is crucial when $\mathscr{R}$ and n are large.

Execution Modes. Let $i \in \nu(k-1)$ and $j \in \nu(k)$ index two nodes in consecutive clusters, connected via a logical arc (i, j) corresponding to a physical subnetwork (see Figure 2)—with i and j representing *Stop1* and *Stop2*, respectively. The node-to-node departure time function $\Omega_{ij} = \Gamma_j \circ \widetilde{\Phi}_{ij}$ captures the departure time envelope from i to reach j within time-window constraints, and is stored as $\mathscr{F}^{i,j}_{\nu(k-1),\nu(k)}$ in the matrix $\boldsymbol{\mathscr{F}}$ (Table 2). We distinguish two execution modes: (i) *Precomputed*: All functions Γ_j, $\widetilde{\Phi}_{ij}$, and Ω_{ij} are computed in a preprocessing phase (e.g., under static demand), enabling constant-time access during DP. Time-window feasibility checks are bypassed at runtime. Also, read Section 3.4. (ii) *On-the-fly*: In dynamic or online settings, none of these functions are available a priori. All components—including feasibility verification and functional composition—are computed on demand, resulting in significantly higher computational cost.

Classical VRPs as a Special Case. When each cluster $\nu(i)$ is a singleton node, the DP collapses to a composition chain $\mathscr{F}_{\nu(i),\nu(k)}(t) = \Omega_{\nu(k-1),\nu(k)} \circ \cdots \circ \Omega_{\nu(i),\nu(i+1)}(t)$, eliminating branching and envelope operations. Complexity bounds reduce accordingly, as shown in Corollaries 1, 2, 3, and 4.

5.3 Exact Route Scheduling Schemes for GRSP-TD$_{\mathcal{RN}}$

This section focuses on the scheduling component (Section 5.1). We present two exact DP schemes that propagate lower envelopes of departure-time functions along a given route sequence, and thereby minimize the total route duration $\Delta^*_{\sigma(r)}$ in (38) by choosing the optimal depot start time t^* in (39). The schemes differ in the *order* in which compositions are formed and propagated.

5.3.1 Linear Forward Propagation (LFP)

A natural approach is to compute the envelopes *forward*, starting at the departure depot, processing clusters sequentially, and finishing at the arrival depot; see Algorithm 7. Theorem 7 establishes correctness.

Theorem 7 (LFP solves the DP to optimality) *Algorithm 7 correctly and exactly solves the dynamic program and returns the complete matrix of departure-time functions $\boldsymbol{\mathscr{F}}$ whenever a feasible schedule exists for the route; otherwise it certifies infeasibility.*

Algorithm 7: Linear Forward Propagation (LFP)

Input: Route r with clusters and nodes.
Output: Feasibility of route r, updated route states and attributes, and updated $\mathscr{F}$ matrix.
Initialization:

$r' \leftarrow r$.clone() // Create a copy of the route
r'.initialize_DP() // Initialize states
Function *solveDP()*:
  if r' *is empty* **then**
    return ***false***
  foreach *origin_index* $\leftarrow 0$ **to** $|\sigma(r)| - 2$ **do**
    foreach $c \leftarrow$ *origin_index* **to** $|\sigma(r)| - 2$ **do**
      foreach $s \in \nu(c+1)$ **do**
        if s *has no time-window-feasible predecessors* **then**
          continue
        foreach $p \in \nu(c)$ **do**
          if p *is not TW-feasible for* s **then**
            continue
          // Compute or retrieve $\Omega_{p,s}$
          if $\mathscr{F}^{p,s}_{\nu(c),\nu(c+1)} = \emptyset$ **then**
            $\Psi \leftarrow$ Compute_Psi(p, s)
            $\Gamma \leftarrow$ Compute_Gamma(s)
            if *composite feasible* **then**
              $\Omega \leftarrow \Gamma \circ \Psi$ // Alg. 6
              $\mathscr{F}^{p,s}_{\nu(c),\nu(c+1)} \leftarrow \Omega$
            else
              continue
          if $c =$ *origin_index* **then**
            continue /* we have already set $\mathscr{F}^{p,s}_{\nu(c),\nu(c+1)} \leftarrow \Omega$ */
          else
            foreach $l \in \nu($*origin_index*$)$ **do**
              if *composite feasible between* Ω *and* $\mathscr{F}^{l,p}_{\nu(\textit{origin\_index}),\nu(c)}$ **then**
                $\mathscr{F}^{l,s}_{\nu(\text{origin\_index}),\nu(c+1)} \leftarrow \mathscr{F}^{l,s}_{\nu(\text{origin\_index}),\nu(c+1)} \cup \left\{ \Omega \circ \mathscr{F}^{l,p}_{\nu(\text{origin\_index}),\nu(c)} \right\}$ // Alg. 6
        // Take lower envelope if multiple predecessors
        if $c >$ *origin_index* $\wedge\ |\nu(c)| > 1$ **then**
          foreach $l \in \nu($*origin*$)$ **do**
            $\mathscr{F}^{l,s}_{\nu(\text{origin\_index}),\nu(c+1)} \leftarrow$ LowerEnvelope$(\mathscr{F}^{l,s}_{\nu(\text{origin_index}),\nu(c+1)})$ // Alg. 3
      // Early termination if entire matrix block is empty
      if $\mathscr{F}_{\nu(\textit{origin\_index}),\nu(c+1)}$ *block is empty* **then**
        return ***false***
  // Update the original route state
  r.update_state(r')
  r.set_is_DP_feas_True()
  opt_times $\leftarrow r$.retrieve_opt_start_end()
  r.set_optimal_start_time(opt_times.x)
  r.set_optimal_arr_at_depot_time(opt_times.y)
  // Optimal route duration $\Delta^*_{\sigma(r)}$
  r.set_POF(opt_times.y − opt_times.x)
  return ***true***

5.3.2 Post-Order Binary Tree Propagation (PO-BTP)

An alternative is a decomposition paradigm: the route sequence is *decomposed* into subsequences, extended departure-time functions are computed for the subsequences, and then *aggregated* to obtain the envelope for

the whole route. Unlike LFP, which propagates functions strictly left-to-right, PO-BTP builds a binary tree of compositions and evaluates it in *post-order*.

Algorithm 8: Post-Order Binary Tree Propagation (PO-BTP)

Input: Route r with clusters and nodes.
Output: Feasibility of route r, updated route states and attributes, and updated $\mathscr{F}$ matrix.
Initialization:
$r' \leftarrow r$.clone() `// Create a copy of the route`
r'.initialize_DP() `// Initialize the route states and attributes`
`// Initialize the DP tree`
if r'.*get_Tree*() $= \emptyset$ **then**
  r'.set_Tree() `// Set tree if it does not exist`
`// Set the root of the DP tree`
if r'.*Tree.get_root()* $= \emptyset$ **then**
  first_index $\leftarrow 0$
  last_index $\leftarrow |\sigma(r')| - 1$
  r'.Tree.set_root(TreeNode(first_index, last_index))
Function `MainFunction(solveDP())`**:**
  root $\leftarrow r'$.Tree.get_root()
  `// Post-order traversal & compositions through either Algorithm 9 or 10`
  dpfeas $\leftarrow r' \rightarrow$ `R-PO-BTP(r', root)` (**or, alternatively,** dpfeas $\leftarrow r' \rightarrow$ `I-PO-BTP(r', root)`)
  `// Check if computed solution is valid`
  if $|\sigma(r')| > 2$ **then**
    if $r'.\mathscr{F}_{0,|\sigma(r')|-1}[0,0] = \left\{(T_{init}, T_{\max}), (T_{\max}, T_{\max})\right\}$ **then**
      return ***false***
  `// Update route feasibility and attributes`
  if *dpfeas* $=$ *true* **then**
    `// Find nodes not in the tree and compute their envelopes; calls either Canonical_Decomposition_Search, or Bidirectional_Search, see Bornay, Gendreau, and Gendron (2025)`
    r.compute_or_retrieve_F()
    r.update_state(r')
    r.set_is_DP_feas_True()
    opt_times $\leftarrow r$.retrieve_opt_start_end()
    r.set_optimal_start_time(opt_times.x)
    r.set_optimal_arr_at_depot_time(opt_times.y)
    `// Optimal route duration` $\Delta^*_{\sigma(r)}$
    r.set_POF(opt_times.y − opt_times.x)
  else
    return ***false***
  return ***true***

In their seminal work, Visser and Spliet (2020) used a binary-tree representation for TD-VRPTW move evaluations. Their algorithm constructs the tree *bottom-up*: leaves correspond to pairwise compositions $\mathscr{F}_{\nu(i),\nu(i+1)} = \Omega_{\nu(i),\nu(i+1)}$, and higher levels are built by recursively composing children until the root $\mathscr{F}_{\nu(o),\nu(d)}$; function data are stored in an array-based tree.

We instead, perform a *post-order binary-tree traversal* that forms compositions dynamically. Leaves compute $\Omega_{\nu(i),\nu(i+1)}$; each internal node composes its two children to create the parent. This traversal (i) follows a clean bottom-up logic, (ii) enables *early termination* when a subtree is infeasible—avoiding wasted work—and (iii) decouples representation from storage: the tree is pointer-based, while all values $\mathscr{F}^{l,s}_{\nu(i),\nu(j)}$ are cached externally in the matrix of Table 2.

For a route sequence $\sigma(r) = \{0, 1, \ldots, 12\}$, its binary composition tree is shown in Figure 5, and Table 3 contrasts the construction orders for LFP, PO-BTP, and Visser and Spliet (2020). Although all

three perform the same 23 compositions (shown for a singleton-cluster route of length 13), they differ in *when* and *how* compositions are executed. Notably, PO-BTP combines the advantages of both: like the bottom-up scheme of Visser and Spliet (2020), it tends to compose the *smallest* (fewest-breakpoint) functions first—mitigating intermediate growth—while, like LFP, its post-order depth-first evaluation propagates infeasibility immediately, before the wasted work begins.

Order of Composition	LFP	PO-BTP	**Visser and Spliet (2020)**
1	$\Omega_{0,1}$	$\Omega_{0,1}$	$\Omega_{0,1}$
2	$\Omega_{1,2}$	$\Omega_{1,2}$	$\Omega_{1,2}$
3	$\mathscr{F}_{0,2} = \Omega_{1,2} \circ \Omega_{0,1}$	$\mathscr{F}_{0,2} = \Omega_{1,2} \circ \Omega_{0,1}$	$\Omega_{2,3}$
4	$\Omega_{2,3}$	$\Omega_{2,3}$	$\Omega_{3,4}$
5	$\mathscr{F}_{0,3} = \Omega_{2,3} \circ \mathscr{F}_{0,2}$	$\Omega_{3,4}$	$\Omega_{4,5}$
6	$\Omega_{3,4}$	$\mathscr{F}_{2,4} = \Omega_{3,4} \circ \Omega_{2,3}$	$\Omega_{5,6}$
7	$\mathscr{F}_{0,4} = \Omega_{3,4} \circ \mathscr{F}_{0,3}$	$\mathscr{F}_{0,4} = \mathscr{F}_{2,4} \circ \mathscr{F}_{0,2}$	$\Omega_{6,7}$
8	$\Omega_{4,5}$	$\Omega_{4,5}$	$\Omega_{7,8}$
9	$\mathscr{F}_{0,5} = \Omega_{4,5} \circ \mathscr{F}_{0,4}$	$\Omega_{5,6}$	$\Omega_{8,9}$
10	$\Omega_{5,6}$	$\mathscr{F}_{4,6} = \Omega_{5,6} \circ \Omega_{4,5}$	$\Omega_{9,10}$
11	$\mathscr{F}_{0,6} = \Omega_{5,6} \circ \mathscr{F}_{0,5}$	$\mathscr{F}_{0,6} = \mathscr{F}_{4,6} \circ \mathscr{F}_{0,4}$	$\Omega_{10,11}$
12	$\Omega_{6,7}$	$\Omega_{6,7}$	$\Omega_{11,12}$
13	$\mathscr{F}_{0,7} = \Omega_{6,7} \circ \mathscr{F}_{0,6}$	$\Omega_{7,8}$	$\mathscr{F}_{0,2} = \Omega_{1,2} \circ \Omega_{0,1}$
14	$\Omega_{7,8}$	$\mathscr{F}_{6,8} = \Omega_{7,8} \circ \Omega_{6,7}$	$\mathscr{F}_{2,4} = \Omega_{3,4} \circ \Omega_{2,3}$
15	$\mathscr{F}_{0,8} = \Omega_{7,8} \circ \mathscr{F}_{0,7}$	$\Omega_{8,9}$	$\mathscr{F}_{4,6} = \Omega_{5,6} \circ \Omega_{4,5}$
16	$\Omega_{8,9}$	$\Omega_{9,10}$	$\mathscr{F}_{6,8} = \Omega_{7,8} \circ \Omega_{6,7}$
17	$\mathscr{F}_{0,9} = \Omega_{8,9} \circ \mathscr{F}_{0,8}$	$\mathscr{F}_{8,10} = \Omega_{9,10} \circ \Omega_{8,9}$	$\mathscr{F}_{8,10} = \Omega_{9,10} \circ \Omega_{8,9}$
18	$\Omega_{9,10}$	$\mathscr{F}_{6,10} = \mathscr{F}_{8,10} \circ \mathscr{F}_{6,8}$	$\mathscr{F}_{10,12} = \Omega_{11,12} \circ \Omega_{10,11}$
19	$\mathscr{F}_{0,10} = \Omega_{9,10} \circ \mathscr{F}_{0,9}$	$\Omega_{10,11}$	$\mathscr{F}_{0,4} = \mathscr{F}_{2,4} \circ \mathscr{F}_{0,2}$
20	$\Omega_{10,11}$	$\Omega_{11,12}$	$\mathscr{F}_{6,10} = \mathscr{F}_{8,10} \circ \mathscr{F}_{6,8}$
21	$\mathscr{F}_{0,11} = \Omega_{10,11} \circ \mathscr{F}_{0,10}$	$\mathscr{F}_{10,12} = \Omega_{11,12} \circ \Omega_{10,11}$	$\mathscr{F}_{0,6} = \mathscr{F}_{4,6} \circ \mathscr{F}_{0,4}$
22	$\Omega_{11,12}$	$\mathscr{F}_{6,12} = \mathscr{F}_{10,12} \circ \mathscr{F}_{6,10}$	$\mathscr{F}_{6,12} = \mathscr{F}_{10,12} \circ \mathscr{F}_{6,10}$
23	$\mathscr{F}_{0,12} = \Omega_{11,12} \circ \mathscr{F}_{0,11}$	$\mathscr{F}_{0,12} = \mathscr{F}_{6,12} \circ \mathscr{F}_{0,6}$	$\mathscr{F}_{0,12} = \mathscr{F}_{6,12} \circ \mathscr{F}_{0,6}$

Table 3: Binary Composition Tree $\mathscr{F}_{0,12}$ for Singleton Route Sequence: this example illustrates an *incomplete* Tree

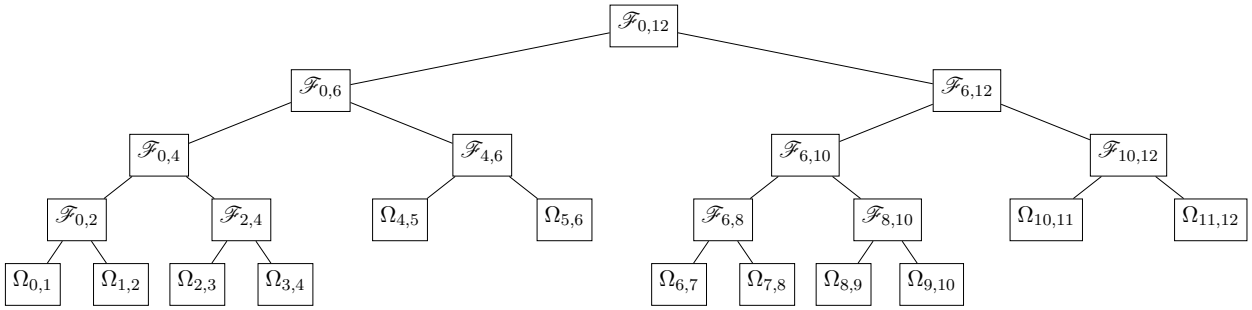

Figure 5: Binary Tree Representation of Composite Functions in Route Scheduling. The tree illustrates an *incomplete* binary tree: although both left and right subtrees rooted at the children of the root node are individually *complete*, the overall tree is not complete, as its lowest level is not fully filled from left to right.

PO-BTP admits two implementations: a stack-based *recursive* variant (R-PO-BTP; Algorithm 9) and a heap-based *iterative* variant (I-PO-BTP; Algorithm 10). Both compute the same depot-to-depot envelope.

Theorem 8 (PO-BTP solves the DP to optimality) *Algorithm 8, equipped with either the recursive post-order composition (Algorithm 9) or the iterative post-order composition (Algorithm 10), correctly and exactly*

Algorithm 9: Recursive Post-Order Binary Tree Propagation (R-PO-BTP)

Input: Route $\mathscr{R}$
Output: Feasibility of route $\mathscr{R}$, and updated $\boldsymbol{\mathscr{F}}$ matrix.
Initialization:
// Recursive Post-Order Traversal (RPOT)
Function *RPOT*($\mathscr{R}$, *current*)**:**
  if *current* $= \emptyset$ **then**
    return ***true*** // Base case: no infeasibility
  // Recursively solve left and right subtrees
  if $\neg$*RPOT*($\mathscr{R}$, *current.left*) ***or*** $\neg$*RPOT*($\mathscr{R}$, *current.right*) **then**
    return ***false*** // If any subtree is infeasible, return false
  // Process current node
  isFeas $\leftarrow$ processNode($\mathscr{R}$, current)
  return *isFeas*
// Check feasibility of composite functions
Function *processNode*($\mathscr{R}$, *node*)**:**
  if *node.is_leaf* **then**
    return $\mathscr{R}$*.if_feas_compute_omega*(*node*)
  // Compose functions if feasible-Alg. 6
  if *node.left* $\neq \emptyset$ ***and*** *node.right* $\neq \emptyset$ **then**
    if *node.left.is_leaf* ***and*** *node.right.is_leaf* **then**
      return $\mathscr{R}$*.if_feas_compute_composite_omega_omega*(*node*)
    else if $\neg$*node.left.is_leaf* ***and*** *node.right.is_leaf* **then**
      return $\mathscr{R}$*.if_feas_compute_composite_omega_F*(*node*)
    else if $\neg$*node.left.is_leaf* ***and*** $\neg$*node.right.is_leaf* **then**
      return $\mathscr{R}$*.if_feas_compute_composite_F_F*(*node*)
  return ***true***

Algorithm 10: Iterative Post-Order Binary Tree Propagation (I-PO-BTP)

Input: Route $\mathscr{R}$
Output: Feasibility of route $\mathscr{R}$, and updated $\boldsymbol{\mathscr{F}}$ matrix.
Initialization:
// Iterative Post-Order Traversal (IPOT)
Function *IPOT*($\mathscr{R}$, *current*)**:**
  if *current* $= \emptyset$ **then**
    return ***false*** // Invalid traversal
  // Initialize node stack
  nodes $\leftarrow \emptyset$ // Stack of nodes
  lastNodeVisited $\leftarrow \emptyset$
  while *current* $\neq \emptyset$ ***or*** *nodes* $\neq \emptyset$ **do**
    if *current* $\neq \emptyset$ **then**
      nodes.push(current)
      current $\leftarrow$ current.left_child
    else
      topNode $\leftarrow$ nodes.top()
      if *topNode.right_child* $\neq \emptyset$ ***and*** *lastNodeVisited* $\neq$ *topNode.right_child* **then**
        current $\leftarrow$ topNode.right_child
      else
        nodes.pop() // Process feasibility check for node
        if $\neg$*processNode*($\mathscr{R}$, *topNode*) **then**
          return ***false***
        lastNodeVisited $\leftarrow$ topNode
  return ***true***
// Check feasibility of composite functions
Function *processNode*($\mathscr{R}$, *node*)**:**
  // The same as the one shown in Algorithm 9

solves the dynamic programming problem: it returns the complete matrix $\boldsymbol{\mathscr{F}}$ if and only if a feasible schedule exists, and otherwise, detects and certifies infeasibility.

Table 4 presents the algorithmic time complexity of the two proposed dynamic programming algorithms—LFP and PO-BTP—under both *precomputed* and *on-the-fly* modes, for both the generalized setting (with multi-node clusters) and the classical case with singleton clusters. Two computational scopes are considered: (i) **Depot→Depot**, which refers to computing the departure time function from the origin depot to the destination depot—corresponding to a single origin-destination pair and a single row of the matrix $\boldsymbol{\mathscr{F}}$; and (ii) **All OD–pairs**, which involves computing optimal departure time functions for every origin-destination cluster pair in the route, i.e., the entire matrix $\boldsymbol{\mathscr{F}}$. In generalized routes (on-the-fly), PO-BTP reduces the degree in n by one relative to LFP: Depot→Depot $O(\mathscr{R}n^2(\mathscr{B}+n\log n))$ vs. $O(\mathscr{R}n^3\log n+\mathscr{R}n^2\mathscr{B})$; All OD–pairs $O(\mathscr{R}^2n^2(\mathscr{B}+n\log n)\log\mathscr{R})$ vs. $O(\mathscr{R}^2n^3\log n+\mathscr{R}^2n^2\mathscr{B})$.

For formal proofs of the reported bounds, refer to the propositions and corollaries in Appendix E.

		LFP		PO-BTP	
Mode	**Scope**	**Generalized**	**Singleton**	**Generalized**	**Singleton**
Precomputed	Depot→Depot	$\mathcal{O}(\mathscr{R}n^3 \log n)$	$\mathcal{O}(\mathscr{R})$	$\mathcal{O}(\mathscr{R}n^3 \log n)$	$\mathcal{O}(\mathscr{R})$
	All OD–pairs	$\mathcal{O}(\mathscr{R}^2 n^3 \log n)$	$\mathcal{O}(\mathscr{R}^2)$	$\mathcal{O}(\mathscr{R}^2 n^2 \log n \log \mathscr{R})$	$\mathcal{O}(\mathscr{R}^2)$
On-the-Fly	Depot→Depot	$\mathcal{O}(\mathscr{R}n^3 \log n + \mathscr{R}n^2\mathscr{B})$	$\mathcal{O}(\mathscr{R}\mathscr{B})$	$\mathcal{O}(\mathscr{R}n^2(\mathscr{B} + n \log n))$	$\mathcal{O}(\mathscr{R}(\mathscr{B} + \log \mathscr{R}))$
	All OD–pairs	$\mathcal{O}(\mathscr{R}^2 n^3 \log n + \mathscr{R}^2 n^2 \mathscr{B})$	$\mathcal{O}(\mathscr{R}^2 \mathscr{B})$	$\mathcal{O}(\mathscr{R}^2 n^2(\mathscr{B} + n \log n) \log \mathscr{R})$	$\mathcal{O}(\mathscr{R}^2 \log \mathscr{R})$

Table 4: Comparative Complexity of LFP vs. PO-BTP

We now describe the second component of the optimization layer: *solution improvement*.

5.4 Solution Improvement

A solution is improved by modifying one or more routes via intra-route or inter-route moves. The process typically (i) checks feasibility of the move, (ii) evaluates its cost against the chosen criterion (e.g., route-duration objective), (iii) applies the move, (iv) invokes one of our DP schedulers to re-schedule the modified route(s), and (v) updates all route attributes. We do not detail every local-search operator; instead, we briefly describe the procedure for insertion/removal, which we use when embedding our DP schedulers in a sequential route-construction heuristic. The same design readily applies—with minor adaptations—to other local-search operators.

5.4.1 Fast Move Evaluation: Screening, Lower Bounding

For each request pair, we precompute the *minimum travel time (MTT)* over the sub-network connecting them, using the fastest arc speeds across all arcs. This can be approximated efficiently in real time. Each cluster $\mathscr{C}$ is assigned a time window $[\underline{t}, \overline{t}]$, where $\underline{t} = \min\{a_n\}$, $\overline{t} = \max\{b_n\}$ for all $n \in \mathscr{C}$, and $[a_n, b_n]$ is the time window of location n. During screening, starting from the depot and the earliest departure time, we compute an approximate earliest arrival time (AEAT) using MTT, and propagate an approximate earliest departure time (AEDT) at each cluster, constrained by its time window. If at any index $i \in \sigma(r')$, AEDT violates the cluster's time window, the move is pruned early. The function `singleClusterFastFeas` performs: (i) index validation, (ii) vehicle capacity check (for pickups), (iii) approximate time-window feasibility using MTT, (iv) if passed, exact time-window feasibility between adjacent nodes. If both pickup and dropoff clusters of a request $m \in \mathscr{M}$ pass this test for given insertion indices, a lower bound on the route duration is computed based on the AEDT at the final depot. If this lower bound improves upon the incumbent exact cost (stored in a *Static Move Descriptor (SMD)*; see Zachariadis and Kiranoudis (2010)).

We compute exact route costs by reusing cached entries from Table 2, rather than rerunning the full DP.

5.4.2 Offline Exact Route Cost Evaluation

Consider evaluating the insertion of a request $m \in \mathscr{M}$ into a route r with $\sigma(r) = \{\nu(0), \nu(1), \ldots, \nu(10), \nu(11)\}$, where $\nu(0)$ and $\nu(11)$ are singleton departure/arrival depots and $\nu(1)$–$\nu(10)$ are the pickup/dropoff clusters of five assigned requests. To obtain the best insertion cost for m, we enumerate promising pickup and delivery positions (those that are feasible and pass the lower-bound screen in Section 5.4.1) and then perform *offline* exact cost

evaluation: we compose and propagate optimal departure-time envelopes on the subsequences created by the hypothetical insertion, without yet modifying the route. Suppose we test pickup position 3 and delivery position 9, yielding the candidate route r' with $\sigma(r') = \{\nu(0), \nu(1), \nu(2), \mathscr{P}_m, \nu(3), \ldots, \nu(8), \mathscr{D}_m, \nu(9), \nu(10), \nu(11)\}$. This query reuses precomputed entries for the subsequences $[\nu(0), \nu(2)]$, $[\nu(3), \nu(8)]$, and $[\nu(9), \nu(11)]$. Applying Eq. (37) (Theorem 5), and suppressing node indices and the explicit $\mathscr{LE}(\cdot)$ for brevity, we carry out: (1) $\mathscr{F}_{\nu(2),\,\mathscr{P}_m}$; (2) $\mathscr{F}_{\nu(0),\,\mathscr{P}_m}$ by composing (1) with $\mathscr{F}_{\nu(0),\,\nu(2)}$; (3) $\mathscr{F}_{\mathscr{P}_m,\,\nu(3)}$; (4) $\mathscr{F}_{\mathscr{P}_m,\,\nu(8)}$ by composing (3) with $\mathscr{F}_{\nu(3),\,\nu(8)}$ (5) $\mathscr{F}_{\nu(0),\,\nu(8)}$ by composing (4) with (2); (6) $\mathscr{F}_{\nu(8),\,\mathscr{D}_m}$; (7) $\mathscr{F}_{\mathscr{D}_m,\,\nu(9)}$; (8) $\mathscr{F}_{\mathscr{D}_m,\,\nu(11)}$ by composing (7) with $\mathscr{F}_{\nu(9),\,\nu(11)}$; (9) $\mathscr{F}_{\nu(0),\,\mathscr{D}_m}$ by composing (6) with (5); and finally, (10) $\mathscr{F}_{\nu(0),\,\nu(11)}$ by composing (8) with (9). on is then obtained by plugging (10) into Eq. (38).

This procedure avoids expensive end-to-end rescheduling. In the tree-based DP, this requires a robust tree search whenever a desired subsequence $\mathscr{F}_{\nu(a),\nu(b)}$ is not an explicit node. As we demonstrate in our companion work (Bornay, Gendreau, and Gendron 2025), the retrieval method proposed by Visser and Spliet (2020) is flawed and can return invalid or non-minimal chains. For this paper, we therefore employ the provably optimal Bidirectional Search algorithm detailed in Bornay, Gendreau, and Gendron (2025). For example, to retrieve the minimal chain for $\mathscr{F}_{\nu(3),\nu(8)}$ (see Figure 8), the algorithm correctly returns $\mathscr{F}_{\nu(6),\nu(8)} \circ \mathscr{F}_{\nu(4),\nu(6)} \circ \mathscr{F}_{\nu(3),\nu(4)}$.

We defer node selection and depot start-time decisions throughout the improvement phase; they are finalized in post-processing on the resulting scheduled routes (Section 6).

6 Post-processing

Once the full departure–time envelope $\mathscr{F}^{0,0}_{\nu(o),\nu(d)}(t)$ and its minimizer $t^* = \arg\min_{t\in\mathscr{H}}\big[\mathscr{F}^{0,0}_{\nu(o),\nu(d)}(t)\big]$ have been computed (by any of our DP schemes), there are two natural ways to extract a concrete feasible path (nodes *and* waiting times) consistent with this envelope: (i) A *per-cluster scan* that, for each intermediate cluster $\nu(j)$, $1 \le j \le 2L$, picks one optimal stop $s_j \in \nu(j)$ lying on the global optimum (Algorithm 11); (ii) A full *label-setting* pass through the cluster-layered DAG that recovers all arrival times, waits and predecessor pointers (Algorithm 12). Formal algorithmic time and space complexities (Table 5) are provided in Propositions 13 and 15 in Appendix F.

Remark 2 (Multiple "Optima" on a Flat Envelope) *As discussed in the previous section, the departure-time function—unlike the arrival-time functions—is not necessarily strictly injective. Consequently, the optimal envelope $\mathscr{F}^{0,0}_{\nu(0),\nu(2L+1)}(t)$ may contain a constant segment, for example: $\{(t_p, \mathscr{F}_{\nu(0),\nu(2L+1)}(t_p)), (t_q, \mathscr{F}_{\nu(0),\nu(2L+1)}(t_q))\}$, where $\mathscr{F}_{\nu(0),\nu(2L+1)}(t_p) = \mathscr{F}_{\nu(0),\nu(2L+1)}(t_q) = \mathscr{F}^*$. The interval $[t_p, t_q]$ defines a region of forward time slack (see Savelsbergh (1992)), during which any departure time results in the same final arrival time at the depot, without violating time-window constraints. In such cases, the choice of the optimal departure time t^* depends on the overarching optimization criterion. If minimizing total route duration is the goal, then the latest point $t^* = t_q$ yields $\Delta t = \mathscr{F}^* - t_q < \mathscr{F}^* - t_p$, and is therefore preferred. On the other hand, if the objective were to minimize the arrival time $\mathscr{F}(t)$ itself, then any $t \in [t_p, t_q]$ could be selected, including $t^* = t_p$. Accordingly, the selection of t^* within a flat segment should be guided by the decision maker's strategy or problem-specific objectives. Choosing t_p may improve robustness by buffering against downstream infeasibility (e.g., due to tight time windows), whereas selecting t_q reduces total waiting*

Algorithm 11: Optimal Node Selection

Input: DP matrix $\mathscr{F}$, envelope minimizer t^*
Output: Optimal node s_j^* in each cluster $\nu(j)$, $j = 1, \ldots, 2L$

for $j \leftarrow 1$ **to** $2L$ **do**
  foreach $s \in \nu(j)$ **do**
    $t_{j,s} \leftarrow \mathscr{F}^{0,s}_{\nu(o),\nu(j)}(t^*)$;
    $a_{j,s} \leftarrow \mathscr{F}^{s,0}_{\nu(j),\nu(d)}\big(t_{j,s}\big)$;
  Pick any
  $s_j^* \in \arg\min_{s\in\nu(j)} a_{j,s} = \arg\min_{s\in\nu(j)} \mathscr{F}^{s,0}_{\nu(j),\nu(d)}\big(\mathscr{F}^{0,s}_{\nu(o),\nu(j)}(t^*)\big)$.
return $\big(s_1^*, \ldots, s_{2L}^*\big)$;

Algorithm 12: Optimal-Path Recovery

Input: Cluster sequence $\nu(0), \ldots, \nu(2L+1)$, DP-matrix $\mathscr{F}$, optimal start t^*
Output: Complete path $(u_0, \ldots, u_{2L+1})$ with times

foreach *node* $u \in \nu(0)$ **do**
  $\mathrm{arr}_u \leftarrow t^*,\ \mathrm{dep}_u \leftarrow t^*,\ \mathrm{prev}_u \leftarrow \mathrm{nil}$
for $k \leftarrow 0$ **to** $2L$ **do**
  foreach *pred* $u \in \nu(k)$ *with finite* dep_u **do**
    foreach *suc* $v \in \nu(k+1)$ *with* $\mathscr{F}^{u,v}_{\nu(k),\nu(k+1)} \neq \emptyset$ **do**
      $\tau = \mathscr{F}^{u,v}_{\nu(k),\nu(k+1)}(\mathrm{dep}_u)$;
      if $\tau > b_v$ **then continue**;
      $\mathrm{wait} = \max\{0, a_v - \tau\},\ \mathrm{dep} = \tau + \mathrm{wait}$;
      if $\mathrm{dep} < \mathrm{dep}_v$ **then**
        update $\mathrm{arr}_v = \tau,\ \mathrm{dep}_v = \mathrm{dep},\ \mathrm{wait}_v = \mathrm{wait},\ \mathrm{prev}_v = u.$
Back-track from the unique sink in $\nu(2L+1)$ to recover the path.

times along the route. In practice, one may also choose an intermediate time based on driver preferences, synchronization requirements, or broader operational policies. Thus, in such cases, the choice of t^* is governed by the decision context.

Ref.	Algorithm	Time	Space
Prop. 13	Optimal node selection	$\mathcal{O}(L\,n \log \mathscr{B})$	$\mathcal{O}\Big(\sum_j \lvert\nu(j)\rvert\Big)$
Prop. 15	Path recovery	$\mathcal{O}\big(L\,n^2 \log \mathscr{B}\big)$	$\mathcal{O}(\sum_k \lvert\nu(k)\rvert)$

Table 5: Post-processing algorithmic costs.

7 Computational Experiments

All algorithms—including preprocessing, dynamic programming, and postprocessing—were implemented in `C++23`, compiled via `CMake 3.29` using Homebrew's `LLVM/Clang` with `-O3`, `-march=native`, and link-time optimization (`-flto`). Builds used `cmake -build . -parallel` with interprocedural optimization enabled. Experiments were executed single-threaded on a 2021 MacBook Pro (14 in., Apple M1 Pro, 10 cores, 16 GB unified memory) running macOS Sequoia 15.4.1. To ensure consistent CPU-time measurements, all runs were isolated via `tmux` and `caffeinate`, and timed using `std::chrono::high_resolution_clock`. The instances are generated to reflect realistic operational settings.

7.1 Network

A 16×16 square-kilometer area is considered, within which stops, intersections, and connecting arcs are randomly generated. The stop density is set to 2.0 stops per square kilometer. In *sparse* networks, each node is connected by 1 to 2 parallel arcs; in *dense* networks, this range increases to 4 to 6. The planning horizon is set to 720.0 minutes, i.e., $[t_{\text{init}} = 0.0,\ t_{\max} = 720.0]$, representing a 12-hour window from 8:00 AM to 8:00 PM. Each arc has a unique, piecewise-constant speed function defined over three non-overlapping

intervals. Arcs are assigned one of three speed profiles—*downtown*, *outskirts*, or *highway*—with speeds sampled independently within regulatory bounds. No two arcs share the same interval partition or speed vector. For both *sparse* and *dense* settings, 30 instances are generated and used in the experiments for *TD quickest path problems* (see Section 3.2) and for the *sequential route construction heuristic (RCH)* based on our proposed exact Dynamic Programming schemes: LFP, I-PO-BTP, and R-PO-BTP.

7.2 Requests

Transit requests for a GDARP-TD$_{\mathcal{RN}}$ are synthetically generated to reflect a realistic platform-based booking system. Each request corresponds to a client specifying origin and destination points. Based on these, a set of candidate pickup and dropoff clusters is proposed—each consisting of 1 to 5 nearby stops within 1.0 km walking distance. Individual time windows for each stop are computed based on an average walking speed of $5/60$ km/min. Each request serves 1 to 3 passengers. Time windows span between 30 and 60 minutes and account for a maximum ride-time constraint, which is enforced by requiring the in-vehicle travel time to be less than three times the quickest direct travel time using a dedicated vehicle.

7.3 Performance Comparison of Time-Dependent Quickest Path Algorithms

Table 6 presents results for both *sparse* and *dense* networks. For each pair of stops, a dedicated *subnetwork* is constructed, yielding a total of 512×511 directed subnetworks ($|\mathscr{G}_{\text{sub}}| = 261{,}632$). For every arc in each subnetwork, the closed-form arrival time function $\Psi_{i,j}$ is computed using our proposed quickest-path algorithms: the Time-dependent Quickest Path Faster Algorithm (TDQPFA) and the Time-Dependent Dijkstra's Algorithm (TDDA). These are benchmarked against the TD Bellman-Ford variant (BF) from Vidal et al. (2021). Each method is executed ten times per instance, and average values are reported. Table 7 reports percent speedups of our methods over the Bellman–Ford variant (BF) of Vidal et al. (2021), computed as $100\times\left(\frac{\text{BF}}{\text{Algorithm}}-1\right)$ and summarized over the instances where each method outperforms BF. TDDA achieves speedup in 28 out of 30 sparse instances and 29 out of 30 dense instances, with mean improvements of 70.7% and 71.9%, and maximum gains exceeding 1500% and 677%, respectively. TDQPFA outperforms BF in 29 sparse and 25 dense instances, with average gains of 43.0% and 117.2%, and maximum speedups exceeding 830% in both network types. These results clearly demonstrate the substantial computational advantage of the proposed methods in TD quickest-path queries.

Algorithm	**Sparse**			**Dense**		
	Min (%)	Mean (%)	Max (%)	Min (%)	Mean (%)	Max (%)
TDQPFA	9.97	42.99	834.09	4.45	117.22	829.96
TDDA	14.79	70.71	1502.08	5.03	71.90	677.19

Table 7: Speedups (%) over BF, computed as $100\times\left(\frac{\text{BF}}{\text{Algorithm}}-1\right)$.

7.4 Performance Comparisons of the Proposed Dynamic Programming (DP) Schemes

To test and compare the performance of the DP methods (LFP, I-PO-BTP, and R-PO-BTP) to solve the GRSP-TD$_{\mathcal{RN}}$ as proposed in Section 2, we consider solving a sequential route construction heuristic (RCH) of

Sparse							Dense						
Instance				TDQPP			Instance				TDQPP		
ID	$\|\mathcal{N}\|$	$\|\mathcal{A}\|$	density	TDQPFA	TDDA	BF	ID	$\|\mathcal{N}\|$	$\|\mathcal{A}\|$	density	TDQPFA	TDDA	BF
1	868376	957741	1.1030	0.082423	**0.079185**	0.091075	1	867196	2491337	2.8724	0.543135	**0.530368**	0.567897
2	862492	945599	1.0963	0.083396	**0.079785**	0.091845	2	865944	2489088	2.8746	2.146248	**1.683593**	2.561269
3	860856	943316	1.0956	0.084023	**0.080552**	0.092469	3	867872	2491908	2.8704	7.660515	**2.940260**	3.127591
4	866706	954907	1.1017	0.083343	**0.079797**	0.091811	4	862908	2481904	2.8761	**0.537617**	4.030596	4.999628
5	872212	964922	1.1066	0.083239	**0.079718**	0.091636	5	861026	2479087	2.8792	0.537314	**0.524610**	0.561597
6	862362	945773	1.0969	0.083584	**0.079960**	0.091919	6	870236	2496761	2.8685	0.537285	**0.524769**	0.561541
7	864442	950301	1.0993	0.083045	**0.079712**	0.091543	7	871888	2500578	2.8672	0.537792	**0.525195**	1.275808
8	862908	946778	1.0969	0.083437	**0.079943**	0.091829	8	868816	2493640	2.8697	0.563620	**0.551082**	0.589603
9	857700	936161	1.0913	2.475114	**0.080229**	0.092244	9	863422	2482042	2.8740	1.883258	**1.568241**	3.200522
10	865202	951138	1.0993	0.083005	**0.079722**	0.091535	10	867458	2490868	2.8711	**1.801935**	4.324030	4.012263
11	864478	949600	1.0985	0.083296	**0.079864**	0.091778	11	866030	2489087	2.8747	0.538533	**0.526209**	4.089627
12	865094	951106	1.0992	**0.083422**	0.094909	0.091911	12	860708	2476548	2.8774	0.759418	**0.742458**	0.793464
13	863128	947319	1.0978	0.083485	**0.079979**	0.091895	13	861122	2478506	2.8774	1.969604	**0.730385**	2.119803
14	864346	949932	1.0993	0.083182	**0.079576**	0.091561	14	862234	2480160	2.8765	0.542002	**0.529215**	0.566405
15	865202	950829	1.0987	0.083412	**0.079777**	0.091865	15	863530	2482936	2.8748	0.788811	**0.704068**	1.439284
16	866432	953936	1.1009	0.083367	**0.079901**	0.091715	16	869316	2496715	2.8710	0.639726	**0.624934**	0.668207
17	860080	940801	1.0946	0.083770	**0.080087**	0.092312	17	860812	2476787	2.8773	0.767959	**0.750306**	0.802161
18	862234	945544	1.0966	0.083317	**0.079741**	0.091835	18	863314	2482265	2.8751	**0.568966**	1.757033	0.991836
19	867678	956621	1.1024	0.082986	**0.079489**	0.091569	19	863976	2484344	2.8747	0.536407	**0.523815**	0.560467
20	867060	955534	1.1025	0.083511	**0.080037**	0.091967	20	862660	2480430	2.8752	0.536368	**0.523638**	0.669724
21	865312	951364	1.0993	0.083440	**0.080033**	0.091869	21	863640	2483977	2.8754	1.627444	**0.524992**	0.561875
22	858124	936821	1.0918	0.083753	**0.080257**	0.092354	22	856020	2467887	2.8825	**0.537847**	1.320482	3.912358
23	867248	955485	1.1018	0.137911	**0.080409**	1.288216	23	866612	2489563	2.8727	0.591105	**0.524584**	0.561271
24	868110	956716	1.1025	0.083055	**0.079585**	0.091521	24	866724	2489281	2.8718	5.041780	**3.384291**	3.554388
25	866226	953753	1.1009	0.082991	**0.079668**	0.091510	25	861448	2479397	2.8781	0.537793	**0.524924**	0.783059
26	861058	943228	1.0953	0.083718	**0.081658**	0.198097	26	866482	2489446	2.8726	0.537074	**0.524499**	0.563171
27	857982	937350	1.0922	0.083546	**0.080094**	0.092044	27	858096	2471790	2.8812	0.855890	**0.527749**	1.843240
28	861642	944870	1.0967	0.082999	**0.079628**	0.091448	28	858068	2471051	2.8798	0.538454	**0.525968**	0.562715
29	860014	941517	1.0949	0.083897	**0.080301**	0.093688	29	851338	2456231	2.8847	0.948136	**0.527339**	0.656935
30	863520	947494	1.0973	0.083074	**0.079537**	0.091495	30	862692	2479588	2.8731	0.537172	**0.524326**	0.882695

TDQPFA wins: 29/30 (sparse), 25/30 (dense); TDDA wins: 28/30, 29/30.

Table 6: Runtime (ms) to construct $\Psi_{i,j}$ in a 512×511 subnetwork (i.e., $|\mathscr{G}_{\text{sub}}| = 261{,}632$).

a GDARP-TD$_{\mathcal{RN}}$. The requests are first sorted based on the earliest end of time window of their dropoff at the original destination. Then, an empty route is initialized, and as many as feasible pickup and dropoff clusters of the requests are inserted in their best position in the partially constructed route, unless no more request can be assigned to the route, then the next route is initialized and this process is repeated until all requests are assigned. At each iteration, and also at the end of the algorithm, the route sequence is scheduled to optimality through a proposed DP scheme. The accelerated solution improvement is elaborated in Section 5.4.

Tables 8 and 9 present results for the GDARP-TD$_{\mathcal{RN}}$ solved using a sequential RCH, evaluated under three DP schemes: *LFP*, *I-PO-BTP*, and *R-PO-BTP*. Each method is run ten times per instance, and average values are reported. A heterogeneous fleet is used, consisting of: **(i) Bus** (capacity 50), **(ii) Minibus** (capacity 25), **(iii) Van** (capacity 15), and **(iv) Minivan** (capacity 7). Each row in the tables includes: $|\mathscr{M}|$ (number of requests), $|\mathscr{R}|$ (average number of constructed routes), **Vehicle Count (AVG)** (average number of vehicles used by type), **Obj.** (total route duration in minutes, averaged over all scheduled routes), and **Runtime (s)** (average runtime in seconds to schedule all routes via each DP method). All DP schemes return identical objective values (differences $< 10^{-6}$); we therefore focus on runtime. Table 8 reports results under the *on-the-fly* mode, where TD departure functions Ω_{ij} are computed during route scheduling. Table 9 reports

Figure 6: Geometric-mean percent speedups over BF. Error bars: 95% CIs from the log scale; runtime per subnetwork (10 runs).

results under the *precomputed* mode, where all Ω_{ij} functions are computed prior to execution. This strategy is particularly advantageous in *static-request* settings where all customer requests are known in advance.

7.4.1 On-the-Fly Comparison

Among the three DP schemes run *on-the-fly*, the tree-based variants (I-PO-BTP and R-PO-BTP) outperform LFP as instance size grows. R-PO-BTP is fastest in 23/30 sparse cases and 26/30 dense cases, while I-PO-BTP leads only on the very smallest problems. LFP rarely wins and falls further behind on larger $|\mathscr{M}|$.

Sparse											**Dense**									
Instance			Vehicle count (AVG)					Runtime (s)			Instance			Vehicle count (AVG)					Runtime (s)		
ID	$\lvert\mathscr{M}\rvert$	$\lvert\mathscr{R}\rvert$	(i)	(ii)	(iii)	(iv)	Obj. (min)	LFP	I-PO-BTP	R-PO-BTP	ID	$\lvert\mathscr{M}\rvert$	$\lvert\mathscr{R}\rvert$	(i)	(ii)	(iii)	(iv)	Obj. (min)	LFP	I-PO-BTP	R-PO-BTP
1	20	1.36	1.22	1.44	1.78	1.0	2297.51	**0.03**	0.04	0.04	1	20	1.39	1.0	1.78	1.44	1.33	2218.54	0.05	**0.03**	0.04
2	40	1.67	1.78	2.0	1.56	1.33	3230.93	0.5	0.45	**0.35**	2	40	1.58	1.33	1.89	1.56	1.56	2923.24	**0.27**	0.33	0.44
3	60	2.17	2.33	2.33	2.44	1.56	4159.6	0.66	0.63	**0.56**	3	60	2.08	2.11	1.89	1.89	2.44	4015.33	0.91	1.31	**0.67**
4	80	2.94	2.89	3.56	3.0	2.33	5701.12	1.31	0.83	**0.67**	4	80	2.89	3.0	3.44	1.89	3.22	5131.21	1.31	**1.21**	1.23
5	100	3.19	2.89	3.11	3.33	3.44	6614.62	**1.51**	1.67	1.83	5	100	3.0	4.78	2.11	2.56	2.56	6138.62	1.96	**1.57**	2.17
6	120	3.83	3.44	4.22	3.33	4.33	7735.22	2.34	**1.39**	1.4	6	120	4.06	4.67	3.89	3.22	4.44	7629.56	3.54	**2.58**	2.61
7	140	4.33	4.44	4.11	3.78	5.0	8971.11	2.97	**1.96**	2.12	7	140	4.64	5.22	3.89	4.56	4.89	8781.16	2.75	**1.97**	2.09
8	160	4.78	5.22	4.67	4.67	4.56	9909.27	3.28	**2.04**	2.51	8	160	4.86	5.11	5.44	4.78	4.11	9457.24	3.82	**3.32**	3.42
9	180	5.53	5.22	5.44	6.78	4.67	11189.58	5.39	3.78	**3.34**	9	180	5.19	6.0	4.67	5.33	4.78	10049.38	6.8	5.6	**4.83**
10	200	7.25	7.89	6.33	5.89	8.89	14049.91	2.6	2.3	**1.84**	10	200	6.78	6.78	6.78	6.56	7.0	12541.86	5.58	4.03	**3.92**
11	300	9.42	9.44	8.22	9.78	10.22	18483.38	7.19	**5.48**	5.65	11	300	8.36	9.67	7.11	8.89	7.78	16464.07	12.13	**8.94**	9.09
12	400	10.58	10.33	10.67	10.67	10.67	22106.04	14.14	**10.12**	13.33	12	400	9.81	9.33	10.33	9.56	10.0	19788.81	22.48	13.41	**13.27**
13	500	13.42	12.33	11.78	14.44	15.11	28034.83	20.08	**12.2**	13.61	13	500	13.58	12.33	15.11	13.78	13.11	26437.24	25.66	19.34	**13.87**
14	600	16.06	17.78	15.0	15.78	15.67	32953.99	22.14	16.64	**16.57**	14	600	14.64	13.56	12.78	15.11	17.11	29493.55	30.74	28.24	**26.2**
15	700	17.64	18.78	17.0	16.78	18.0	36967.19	30.05	**19.74**	22.27	15	700	17.31	17.33	18.67	16.33	16.89	34571.07	36.16	**27.58**	31.64
16	800	20.61	20.33	20.78	21.56	19.78	42912.37	33.98	**26.75**	28.33	16	800	19.36	19.11	18.22	19.67	20.44	38986.37	40.83	34.59	**33.34**
17	900	22.22	23.89	21.33	22.11	21.56	45115.84	54.68	**34.35**	34.73	17	900	20.64	19.89	22.0	19.44	21.22	42232.05	58.15	**47.34**	48.96
18	1000	24.31	22.44	24.44	24.78	25.56	50440.84	45.12	67.8	**39.22**	18	1000	22.19	21.0	21.89	22.89	23.0	45488.27	83.18	61.41	**55.93**
19	1100	25.33	24.33	25.11	25.44	26.44	53491.48	66.4	958.7	**49.4**	19	1100	23.44	22.22	22.78	25.89	22.89	48401.07	82.68	75.69	**60.91**
20	1200	29.0	27.33	31.33	28.44	28.89	59581.56	63.04	52.41	**51.96**	20	1200	23.56	23.33	24.0	24.78	22.11	50086.35	94.06	95.79	**91.03**
21	1300	30.31	30.89	29.78	30.44	30.11	62950.73	76.63	63.89	**57.91**	21	1300	28.25	26.0	28.33	27.78	30.89	57632.76	128.03	80.31	**79.76**
22	1400	30.31	30.33	28.11	32.33	30.44	63696.96	109.03	80.32	**69.8**	22	1400	27.58	26.11	30.33	26.78	27.11	58317.82	116.85	**100.67**	108.47
23	1500	32.08	30.44	33.22	32.33	32.33	68785.18	133.71	90.31	**76.86**	23	1500	30.61	31.33	30.67	29.44	31.0	63917.32	149.68	113.52	**103.88**
24	1600	35.53	34.56	36.67	35.78	35.11	73774.26	97.7	84.42	**81.09**	24	1600	32.64	32.44	30.67	30.22	37.22	68011.36	135.45	106.68	**105.08**
25	1700	36.89	35.56	37.44	39.22	35.33	78626.39	125.85	98.24	**81.42**	25	1700	34.11	31.33	34.33	34.44	36.33	70366.64	156.3	132.44	**126.99**
26	1800	41.36	42.33	40.0	42.44	40.67	86332.85	116.42	116.52	**88.49**	26	1800	35.11	37.44	34.33	34.44	34.22	73678.28	189.75	**134.18**	137.26
27	1900	41.81	41.22	42.89	42.56	40.56	86231.85	148.34	154.03	**96.93**	27	1900	36.39	34.56	37.33	37.89	35.78	76053.43	258.19	180.1	**167.24**
28	2000	41.83	41.33	44.44	40.78	40.78	87263.58	144.43	206.4	**127.22**	28	2000	38.03	37.22	38.33	39.44	37.11	79811.85	243.52	**171.01**	187.35
29	2500	52.36	52.44	50.11	52.78	54.11	109762.44	229.81	165.7	**149.67**	29	2500	47.83	48.0	49.33	47.11	46.89	98176.14	283.53	194.79	**182.04**
30	3000	60.14	60.22	59.0	58.33	63.0	126841.19	284.56	197.37	**193.43**	30	3000	56.81	57.33	57.22	57.11	55.56	116931.53	347.42	**266.27**	269.3

Table 8: Sequential RCH Experimental Results for Sparse and Dense Instances, Heterogeneous Fleet: 'On-the-fly'

7.4.2 Precomputed Comparison

With precomputation, R-PO-BTP still dominates on larger problems (24/30 sparse, 25/30 dense), but I-PO-BTP narrows the gap—winning in 6 sparse and 5 dense small instances. LFP's relative performance improves slightly versus its on-the-fly times, yet remains the slowest overall.

Sparse											Dense									
Instance			Vehicle count (AVG)					Runtime (s)			Instance			Vehicle count (AVG)					Runtime (s)	
ID	$\lvert\mathscr{M}\rvert$	$\lvert\mathscr{R}\rvert$	(i)	(ii)	(iii)	(iv)	Obj. (min)	LFP	I-PO-BTP	R-PO-BTP	ID	$\lvert\mathscr{M}\rvert$	$\lvert\mathscr{R}\rvert$	(i)	(ii)	(iii)	(iv)	Obj. (min)	LFP	I-PO-BTP	R-PO-BTP
1	20	1.42	1.22	1.78	1.44	1.22	2326.76	0.02	**0.01**	0.02	1	20	1.39	2.22	0.78	1.11	1.44	2212.42	0.03	**0.02**	**0.02**
2	40	1.69	2.44	0.56	1.89	1.89	3193.14	0.29	**0.2**	0.22	2	40	1.58	1.44	1.44	1.78	1.67	2912.68	**0.12**	0.22	0.16
3	60	2.19	1.44	2.56	2.11	2.67	4239.45	0.33	**0.26**	0.28	3	60	1.94	1.78	1.89	2.22	1.89	3853.68	0.56	**0.34**	0.63
4	80	3.0	3.67	2.33	3.67	2.33	5778.7	0.59	**0.47**	0.51	4	80	2.86	2.67	2.89	2.78	3.11	5284.3	0.63	**0.57**	0.71
5	100	3.19	2.44	4.22	3.22	2.89	6695.94	0.83	**0.58**	0.83	5	100	2.86	3.22	2.78	2.67	2.78	5857.31	1.35	0.76	**0.69**
6	120	3.67	3.56	3.56	3.67	3.89	7682.22	1.08	**0.74**	0.79	6	120	3.97	4.22	3.56	4.11	4.0	7479.2	1.55	**0.8**	0.96
7	140	4.36	4.67	4.11	4.0	4.67	8882.26	1.94	**1.13**	1.2	7	140	4.5	4.89	4.22	5.22	3.67	8581.5	1.74	**1.09**	1.41
8	160	4.83	4.44	5.11	4.22	5.56	10025.08	1.54	**1.14**	1.38	8	160	5.14	4.44	5.56	5.56	5.0	9886.0	1.42	1.58	**1.32**
9	180	5.58	6.78	5.89	4.89	4.78	11269.09	2.7	2.51	**2.21**	9	180	5.31	5.67	4.44	5.78	5.33	10353.1	3.68	**2.56**	2.91
10	200	7.31	6.67	7.33	8.22	7.0	14165.68	1.15	1.14	**1.1**	10	200	6.67	6.0	6.89	6.33	7.44	12515.12	2.25	**1.69**	2.47
11	300	8.89	9.11	8.78	9.33	8.33	17692.97	4.09	**2.67**	2.91	11	300	8.25	10.22	7.22	6.89	8.67	16403.34	5.86	4.48	**4.31**
12	400	10.78	10.78	11.22	10.22	10.89	22142.58	7.22	6.52	**5.5**	12	400	9.69	11.78	8.33	9.78	8.89	19674.37	10.29	8.36	**7.35**
13	500	13.0	13.89	10.78	13.11	14.22	27185.8	10.76	8.53	**7.51**	13	500	13.86	13.33	13.0	14.56	14.56	26853.0	9.42	7.22	**7.1**
14	600	16.06	16.56	15.89	15.22	16.56	32940.73	11.22	**7.52**	9.3	14	600	14.22	13.89	14.44	14.67	13.89	29174.54	16.45	12.19	**11.96**
15	700	17.25	18.67	18.78	17.11	14.44	36781.22	17.65	11.46	**10.94**	15	700	17.25	16.33	18.11	18.0	16.56	34763.69	16.87	15.73	**12.53**
16	800	20.33	20.67	22.67	18.89	19.11	42501.58	20.35	**13.82**	13.83	16	800	19.36	19.67	19.44	19.67	18.67	38987.47	21.94	17.33	**15.71**
17	900	22.28	22.67	21.33	19.89	25.22	44865.96	29.34	21.54	**21.51**	17	900	20.69	20.78	20.0	20.89	21.11	42004.96	29.94	24.1	**22.64**
18	1000	23.58	22.56	24.22	24.22	23.33	49336.22	24.0	**20.18**	22.44	18	1000	21.89	20.89	22.44	22.33	21.89	44886.89	43.34	32.8	**26.84**
19	1100	25.31	27.22	25.44	23.11	25.44	53226.24	35.31	**21.96**	23.71	19	1100	23.72	20.0	24.78	25.89	24.22	48349.98	44.38	34.37	**34.29**
20	1200	29.14	28.11	30.22	27.56	30.67	60232.75	34.77	26.19	**26.13**	20	1200	23.81	24.44	24.0	23.44	23.33	50724.33	65.02	**40.05**	47.4
21	1300	29.83	26.89	29.56	31.33	31.56	62735.78	38.2	29.9	**29.6**	21	1300	27.67	28.33	29.78	26.0	26.56	56886.95	54.88	43.88	**39.96**
22	1400	30.17	31.22	27.33	33.78	28.33	63587.52	54.66	**36.19**	41.32	22	1400	27.83	28.11	29.0	24.33	29.89	59306.88	66.8	**40.26**	46.35
23	1500	32.11	31.44	32.22	29.78	35.0	68961.72	63.29	45.0	**40.71**	23	1500	30.39	30.89	30.33	30.78	29.56	62934.4	77.07	56.0	**51.87**
24	1600	35.64	36.11	34.33	37.22	34.89	73605.85	55.77	**44.79**	47.62	24	1600	32.08	31.0	35.11	31.44	30.78	66932.27	75.92	58.0	**53.04**
25	1700	36.67	34.0	36.89	39.56	36.22	78145.72	56.7	43.22	**42.67**	25	1700	34.22	34.22	35.44	33.22	34.0	70227.8	99.16	**67.1**	71.82
26	1800	41.89	41.44	41.33	41.22	43.56	86892.51	58.48	49.42	**46.93**	26	1800	35.06	32.67	37.56	31.89	38.11	73840.95	134.24	75.55	**72.12**
27	1900	42.06	43.78	42.89	38.78	42.78	86064.07	91.33	**57.88**	60.4	27	1900	35.89	35.11	35.44	37.0	36.0	75746.48	114.63	95.94	**93.41**
28	2000	41.69	42.0	43.78	38.89	42.11	86897.71	93.08	**66.19**	72.92	28	2000	38.36	37.0	39.22	39.11	38.11	80604.94	131.93	100.15	**94.66**
29	2500	52.03	52.56	48.11	51.33	56.11	109495.72	137.99	**102.11**	110.43	29	2500	47.5	48.22	48.33	46.78	46.67	98096.39	159.47	120.69	**115.34**
30	3000	58.83	61.44	60.33	58.0	55.56	125654.71	190.92	150.35	**145.8**	30	3000	57.31	54.44	56.11	58.44	60.22	117074.41	192.38	**166.73**	170.7

Table 9: Sequential RCH Experimental Results for Sparse and Dense Instances, Heterogeneous Fleet: 'Precomputed'

7.4.3 Speedup from Precomputation

To quantify the gain from precomputing Ω_{ij}, Table 10 reports min/mean/max/95th-percentile speedups—computed as Speedup (%) $= 100 \times \left(\frac{\text{Runtime}_{\text{on-the-fly}}}{\text{Runtime}_{\text{precomputed}}} - 1\right)$— for each scheme over all 60 instances.

	Sparse			Dense		
Statistic	LFP	I-PO-BTP	R-PO-BTP	LFP	I-PO-BTP	R-PO-BTP
Min Speedup (%)	49.05	31.27	31.37	41.35	50.00	6.35
Mean Speedup (%)	87.07	251.19	80.64	94.86	107.55	100.99
Max Speedup (%)	126.09	4265.66	142.36	172.40	285.29	214.49
95th-percentile (%)	122.00	271.19	115.02	159.56	197.92	173.59

Table 10: Speedup (%) of Precomputed over On-the-fly Mode (60 instances)

Precomputing all Ω_{ij} functions before route scheduling delivers consistent and substantial speedups across every DP scheme and instance type. Even the smallest gains exceed 6% (R-PO-BTP on dense), while average improvements range from 81% (R-PO-BTP on sparse) to 251% (I-PO-BTP on sparse). I-PO-BTP benefits the most, with mean speedups of 251% on sparse and 108% on dense instances and a 95th-percentile gain above

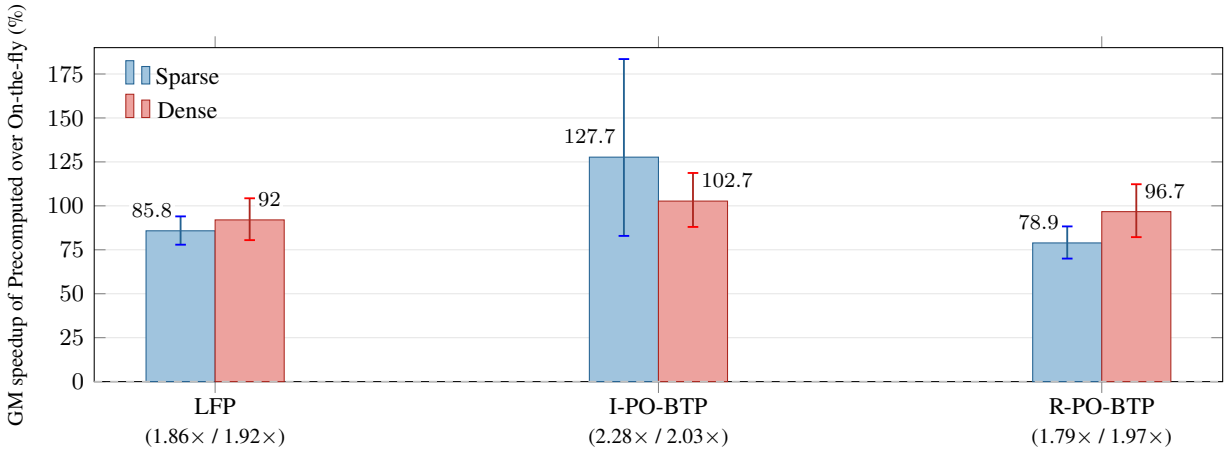

Figure 7: Geometric-mean percent speedup of *precomputed* vs *on-the-fly* schedule runtime (30 instances per density; 10 runs per instance). Error bars: 95% CIs from the log scale. Precomputation wins in 30/30 cases for every method and density.

270% in sparse settings. R-PO-BTP and LFP also see significant reductions—mean speedups of 80–95% and 95th-percentile gains of 115–160%—with all schemes achieving larger peaks on sparse problems. These results confirm that precomputation is universally advantageous, halving runtimes on average and delivering extreme improvements where on-the-fly becomes a bottleneck. The gain entails a memory trade-off—caching cluster-to-cluster profiles in Table 2—but the footprint is modest even on a laptop and is dwarfed by the $\approx 2\times$ average speedup, making precomputation essential for scaling.

8 Conclusion

This paper introduces the Generalized Route Scheduling Problem with Time-dependent Travel Times on Road Networks (GRSP-TD$_{\mathcal{RN}}$), a challenging routing–scheduling problem motivated by modern mobility and logistics platforms. To the best of our knowledge, it is the first study to address a *generalized* route scheduling problem on real road networks with time-dependent (TD) travel times and flexible service locations: each request offers multiple pickup/dropoff candidates with time windows; routes start and end at a depot, visit clustered pickup and delivery locations, and must respect precedence and capacity. Nodes within clusters are connected by *logical arcs*, each representing a road-level *subnetwork* that may include parallel arcs between junctions or stops.

We formulate a generic Bellman–Ford dynamic program for GRSP-TD$_{\mathcal{RN}}$ and show that it *strictly generalizes* four families of classical VRP subproblems, typically treated as SPPRC. We prove strong $\mathcal{NP}$-completeness of feasibility and strong $\mathcal{NP}$-hardness of optimization, and we identify a key limitation of single-time queries: they cannot guarantee global optimality over a continuous horizon. To overcome this, we develop two exact DP schemes—Linear Forward Propagation (LFP) and Post-Order Binary Tree Propagation (PO-BTP)—that propagate *optimal departure-time envelopes* across subnetworks along the route. We further prove that their optimal solutions coincide with those of the Bellman–Ford equations. The PO-BTP scheme is provided in two implementations—recursive (stack-based, R-PO-BTP) and iterative (heap-based, I-PO-BTP)—both returning the same optimal solution.

Because of the theoretical and algorithmic complexity, we propose a three-phase framework: *Preprocessing*, *Optimization*, and *Post-processing*.

Preprocessing. (i) We design a lower-envelope construction algorithm grounded in point–line duality for time-dependent piecewise-continuous linear (PCL) functions. (ii) We introduce two efficient Time-Dependent Quickest Path Problem (TDQPP) algorithms—TDQPFA and TDDA—that compute closed-form *extended arrival* functions on subnetworks and empirically outperform the Bellman–Ford variant of Vidal et al. (2021).

(iii) We model *extended departure-time* functions at road-network level and show a key reduction to avoid recomputing these functions within subnetworks while preserving exactness, and it enables both an *on-the-fly* and a highly effective *precomputed* execution mode.

Optimization. We provide three DP variants in practice—LFP, I-PO-BTP, and R-PO-BTP—and develop (a) fast feasibility screening and lower-bounding, and (b) an *offline exact route-cost evaluation* that uses cached $\mathscr{F}$-values to compute the exact cost of tentative moves *without* rerunning depot-to-depot scheduling. These mechanisms accelerate search, reject infeasible moves early, and avoid redundant recomputation. We embed the DPs within a sequential route-construction heuristic and demonstrate performance up to 3000 requests. We also identify and correct a critical flaw in a state-of-the-art binary-tree search used within PO-BTP. We propose an *optimal, robust bidirectional* search that returns a valid, minimal composition chain for any query segment and prove its correctness.

Post-processing. Given an optimally scheduled route, we present exact algorithms for (i) optimal node selection within each cluster and (ii) path recovery, with stated time–space guarantees.

Since GRSP-TD$_{\mathcal{RN}}$ generalizes subproblems of many classical models, the proposed framework—preprocessing on subnetworks, exact DP scheduling on routes, robust tree search, fast move evaluation, and exact offline route cost computation—can be used as a drop-in component within both exact methods and metaheuristics.

References

Adamo T, Gendreau M, Ghiani G, Guerriero E, 2024 *A review of recent advances in time-dependent vehicle routing*. *European Journal of Operational Research* .

Bornay B, Gendreau M, Gendron B, 2025 *Minimal-chain retrieval in binary composition trees for time-dependent route scheduling*. Working Paper CIRRELT-2025-31, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), URL `https://cirrelt.ca/documentstravail/cirrelt-2025-31.pdf`.

De Berg M, Cheong O, Van Kreveld M, Overmars M, 2008 *Computational geometry: algorithms and applications* (Springer).

Delling D, Wagner D, 2009 *Time-dependent route planning*. *Robust and Online Large-Scale Optimization: Models and Techniques for Transportation Systems*, 207–230 (Springer).

Desaulniers G, 2006 *Shortest path problems with resource constraints*. *Column generation* 5:33.

Desaulniers G, Desrosiers J, Erdmann A, Solomon MM, Soumis F, 2005 *The vehicle routing problem with time windows*. *Column generation*, 145–184 (Springer).

Desrochers M, Soumis F, 1988 *A generalized permanent labelling algorithm for the shortest path problem with time windows*. *INFOR: Information Systems and Operational Research* 26(3):191–212.

Desrosiers J, Dumas Y, Solomon MM, Soumis F, 1995 *Time constrained routing and scheduling*. *Handbooks in operations research and management science* 8:35–139.

Feillet D, Dejax P, Gendreau M, Gueguen C, 2004 *An exact algorithm for the elementary shortest path problem with resource constraints*. *Operations Research* 52(5):745–755.

Foschini L, Hershberger J, Suri S, 2011 *On the complexity of time-dependent shortest paths*. *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete algorithms*, 327–341 (SIAM).

Ghiani G, Guerriero E, 2014 *A note on the ichoua, gendreau, and potvin (2003) travel time model*. *Transportation Science* 48(3):458–462.

Gmira M, Gendreau M, Lodi A, Potvin JY, 2021 *Tabu search for the time-dependent vehicle routing problem with time windows on a road network*. *European Journal of Operational Research* 288(1):129–140.

Har-Peled S, 2011 *Geometric approximation algorithms*. Number 173 in Cambridge Texts in Applied Mathematics (American Mathematical Soc.).

Hashimoto H, Yagiura M, Ibaraki T, 2008 *An iterated local search algorithm for the time-dependent vehicle routing problem with time windows*. *Discrete Optimization* 5(2):434–456.

Hershberger J, 1989 *Finding the upper envelope of n line segments in o (n log n) time*. *Information Processing Letters* 33(4):169–174.

Ichoua S, Gendreau M, Potvin JY, 2003 *Vehicle dispatching with time-dependent travel times*. *European Journal of Operational Research* URL `http://dx.doi.org/10.1016/S0377-2217(02)00147-9`.

Irnich S, 2008 *Resource extension functions: Properties, inversion, and generalization to segments*. *OR Spectrum* 30(1):113–148.

Irnich S, Desaulniers G, 2005 *Shortest path problems with resource constraints*. *Column generation*, 33–65 (Springer).

Savelsbergh MW, 1992 *The vehicle routing problem with time windows: Minimizing route duration*. *ORSA journal on computing* 4(2):146–154.

Vidal T, Martinelli R, Pham TA, Hà MH, 2021 *Arc routing with time-dependent travel times and paths*. *Transportation Science* 55(3):706–724.

Visser TR, Spliet R, 2020 *Efficient move evaluations for time-dependent vehicle routing problems*. *Transportation science* 54(4):1091–1112.

Zachariadis EE, Kiranoudis CT, 2010 *A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem*. *Computers & operations research* 37(12):2089–2105.

Appendices

These appendices collect the abbreviations and notation used throughout the paper and present supplementary theoretical and algorithmic results. Appendix A lists the abbreviations. Appendix B provides the notation used in the subsequent appendices. Appendices C–F present the theoretical and algorithmic results.

A Abbreviations

Abbreviations

GRSP-TD$_{\mathcal{RN}}$	Generalized Route Scheduling Problem with Time-dependent Travel Times on Road Networks
SPPRC-TD$_{\mathcal{RN}}$	Shortest Path Problem with Resource Constraints with Time-dependent Travel Times on Road Networks
ALNS	Adaptive Large Neighborhood Search
CPL	continuous piecewise-linear
DARP	Dial-A-Ride Problem
EDTF	Extended Departure Time Function
ESPPRC	Elementary Shortest Path Problem with Resource Constraints
GDARP-TD$_{\mathcal{RN}}$	Generalized Dial-A-Ride Problem with Time-dependent Travel Times on Road Networks
I-PO-BTP	Iterative Post-Order Binary Tree Propagation
LFP	Linear Forward Propagation
OD	Origin-destination
OEATF	Optimal Extended Arrival Time Function
OEDTF	Optimal Extended Departure Time Function
OSTP	Optimal Start Time Problem
PDPTW	Pickup and Delivery Problem with Time Windows
PO-BTP	Post-Order Binary Tree Propagation
R-PO-BTP	Recursive Post-Order Binary Tree Propagation
SPPRC	Shortest Path Problem with Resource Constraints
SPPTW	Shortest Path Problem with Time Window
SPPTW-CC	Shortest Path Problem with Time Windows and Capacity Constraints
TD	time-dependent
TD-ESPPTW-CC-PD	Time-Dependent ESPPRC with Pickup-Delivery
TD-ESPPTW-CC-TC-PD	Time-Dependent ESPPRC with Time Costs and Pickup-Delivery
TD-SPPTW-CC-PD	Time-Dependent SPPTW with Capacity Constraints and Pickup-Delivery
TD-SPPTW-CC-TC-PD	Time-Dependent SPPTW with Capacity Constraints, Time Costs, and Pickup-Delivery
TDDA	Time-Dependent Dijkstra's Algorithm
TDQPFA	Time-dependent Quickest Path Faster Algorithm
TDQPP	Time-Dependent Quickest Path Problem
TDVRPTW	Time-dependent Vehicle Routing Problem with Time Windows
VRP	Vehicle Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows

B Notation Legend

- Time window of node i: $[a_i, b_i]$.
- Time horizon: $\mathscr{H} = [t_{\min}, t_{\max}]$. Breakpoints of g: $\mathscr{B}_g$. We use $|\mathscr{B}_\phi|$, $|\mathscr{B}_{\Phi_{ij}}|$, and $|\mathscr{B}_{\Omega_e}|$ as dictated by context.

- Arc/stop functions: θ_{ij} travel time; ϕ_{ij} arrival time; Φ_{ij} closed form of ϕ_{ij}; $\widetilde{\phi}_{ij}$ time-window–constrained arrival; $\omega_{ij} = \gamma_j \circ \phi_{ij}$ (departure time); Ω_{ij} closed form of ω_{ij}; Γ_j ready-time function (closed form); γ_j pointwise ready-time function; $\mathscr{F}^{l,s}_{\nu(i),\nu(j)}$ closed-form optimal departure-time function at node $s \in \nu(j)$ given a feasible departure from $l \in \nu(i)$ in route sequence $\sigma(r)$.
- Envelopes and hulls: $\mathscr{LE}$ / $\mathscr{UE}$ lower/upper envelope; $\mathcal{LH}$ / $\mathcal{UH}$ lower/upper hull; LCH/LH lower convex hull (point–line duality is used in the stated lemmas).
- Graph/sequence notation: σ sequence (Def. 4), $|\sigma|$ its length; $\Pi(d)$ predecessors of d; $\mathcal{A}_{u,v}$ arcs $u \to v$; $|\mathcal{A}_{\max}| = \max_{\pi \in \Pi(d)} |\mathcal{A}_{\pi,d}|$.
- Complexity shorthands: n number of segments sent to an envelope; $A_{\text{tot}} = \sum_j |\mathcal{A}_{j,j+1}|$; $|\mathscr{B}_\Omega| = \max_e |\mathscr{B}_{\Omega_e}|$.

Notation. Unless explicitly stated otherwise, the above shorthand notation applies throughout the appendices.

C Mathematical Contents of the Problem Definition Section

Proof of Proposition 1 (Generalization of the SPPRC-TD$_{\mathcal{RN}}$). (i) TD-SPPTW-CC-TC-PD extends SPPTW (Desrochers and Soumis 1988) by incorporating CC, TD, TC (Desaulniers 2006), and PD (Desaulniers et al. 2005). The elementary variant, TD-ESPPTW-CC-TC-PD, adds the no-revisit condition as in Feillet et al. (2004). (ii) TD-SPPTW-CC-PD generalizes SPPTW-CC with TD and PD, while TD-ESPPTW-CC-PD extends ESPPRC (Feillet et al. 2004) by adding time-dependency and PD constraints. Thus, Equation (6) encompasses these models.□

Definition 5 (Checkpoint cluster) *A* checkpoint cluster M_j *is a singleton node (zero service time) with a* point *time window* $[T_j, T_j]$ *and* zero *travel time on every arc entering* M_j*. We impose the precedence chain* $M_1 \prec M_2 \prec \cdots \prec M_m \prec d$ *and set* $T_j := jB$ *(with* $T_0 := 0$*). Because waiting is allowed, feasibility at* M_j *means the schedule reaches* M_j *no later than* T_j *and, if early, waits until* T_j*; arriving after* T_j *is infeasible.*

Proof of Theorem 1 (Feasibility cheching of the GRSP-TD$_{\mathcal{RN}}$ is $\mathcal{NP}$-complete in the strong sense). *Membership in* $\mathcal{NP}$. Given a route order and start time, we can forward–evaluate arrivals with waiting and check all windows and precedence in time polynomial in the input size.

Hardness (strong). Reduce from 3-Partition (strongly $\mathcal{NP}$-complete). The instance has integers $A = \{a_1, \ldots, a_{3m}\}$ with $\sum_{k=1}^{3m} a_k = mB$ and $B/4 < a_k < B/2$ for all k.

Construction. Create depots o, d, element clusters $E_1, \ldots, E_{3m}$ (each a singleton, zero service), and checkpoint clusters $M_1, \ldots, M_m$ as in Definition 5. Set the start window to $[a_o, b_o] = [0, 0]$ and each E_k's window to $[0, \infty)$. For any logical arc *entering* E_k set the (static FIFO) travel time to a_k; for any arc entering M_j or d, set it to 0. All necessary arcs between consecutive clusters are present with these times; service times are zero.

Correctness. Start at $\tau = 0$. Time increases only when entering an element cluster, by exactly a_k; entering any M_j or d adds 0. Let S_j be the total of a_k contributed by the elements scheduled strictly between M_{j-1} and M_j (with $M_0 := o$). Feasibility at M_j requires arrival by jB; since waiting is allowed, this is equivalent to $S_j \le B$. But $\sum_{j=1}^{m} S_j = \sum_{k=1}^{3m} a_k = mB$, hence every S_j must in fact equal B. Because each $a_k \in (B/4, B/2)$, $S_j = B$ can be achieved only by *exactly three* elements. Therefore there is a feasible schedule iff A admits a 3-partition into m triples each summing to B. The reduction is polynomial and uses only static FIFO times with polynomially encoded integers, so hardness holds in the strong sense. □

Proof of Theorem 2 (Optimization of the GRSP-TD$_{\mathcal{RN}}$ is $\mathcal{NP}$-complete in the strong sense). Use the instance built in the proof of Theorem 1 with start time fixed at 0 and the sink d having point window $[mB, mB]$. Every feasible schedule (if any) must arrive at d exactly at mB, so the optimal duration equals mB when the 3-PARTITION instance is a yes-instance. If no 3-partition exists, some checkpoint or d window is necessarily missed and the instance is infeasible (equivalently, the decision variant "duration $\leq mB$?" is false). Thus, an algorithm that computes the optimal duration (or decides whether the minimum duration is $\leq mB$) would decide a strongly $\mathcal{NP}$-complete problem. Therefore, computing the optimum is $\mathcal{NP}$-hard in the strong sense. □

D Mathematical Contents of the Pre-processing Section

Proposition 2 (Properties of travel time function) *Travel time function has the following properties:*

1. *$\theta_{ij}(t)$ is a CPL function.*
2. *θ_{ij} is a total function, i.e., $\theta_{ij} : \mathscr{H} \to \mathscr{H} \cup \{\infty\}$.*
3. *Let $\mathscr{B}_{\nu_{ij}} \coloneqq \{\bar{t}_\tau\}_{\tau=1}^{n-1}$ where $\delta t_\tau \coloneqq [\underline{t}_\tau, \bar{t}_\tau]$ is a sorted collection, in ascending order, of all breakpoints of the piecewise-constant speed $\nu_{ij}(t)$. Then, $\theta_{ij}(t)$ has at most $2\left|\mathscr{B}_{\nu_{ij}}\right|$ breakpoints.*

Proof. Proofs of the Properties 1-3 ar provided as follows:

1. From $d_{ij} = \int_\alpha^{\alpha+\theta_{ij}(\alpha)} \nu_{ij}(t)\,dt$ and $\eta = t - \alpha$, $\theta_{ij}(\alpha) = \int_0^{d_{ij}} \nu_{ij}^{-1}(\alpha+\eta)\,d\eta$. Since ν_{ij} is piecewise constant, ν_{ij}^{-1} is piecewise constant (namely $1/v_\tau$ on the image of δt_τ), whose integral is CPL.
2. It directly follows from Proposition 1. With $\nu_{ij}(t) > 0$ on each interval of $\mathscr{H}$, θ_{ij} exists for every $t \in \mathscr{H}$; if $d_{ij} > \int_t^{t_{\max}} \nu_{ij}$, set $\theta_{ij}(t) = \infty$.
3. To determine the total number of breakpoints in $\theta_{ij}(t)$, we consider two types of breakpoints:

   Type 1 Breakpoints of $\theta_{ij}(t)$ inherited from $\nu_{ij}(t)$: Since $\nu_{ij}(t)$ is piecewise constant, it has breakpoints at times $\mathscr{B}_{\nu_{ij}} \coloneqq \{\bar{t}_\tau\}_{\tau=1}^{n-1}$. For each breakpoint $\bar{t}_\tau \in \mathscr{B}_{\nu_{ij}}$, there is a change in speed from ν_τ (on interval $\delta t_\tau = [\underline{t}_\tau, \bar{t}_\tau]$) to $\nu_{\tau+1}$ (on interval $\delta t_{\tau+1} = [\bar{t}_\tau, \bar{t}_{\tau+1}]$). This change in speed affects the travel time $\theta_{ij}(t)$, introducing a corresponding breakpoint in $\theta_{ij}(t)$ at $t = \bar{t}_\tau$. Therefore, Type 1 contributes exactly $\left|\mathscr{B}_{\nu_{ij}}\right| = n - 1$ breakpoints.

   Type 2 Breakpoints due to arc completion within constant-speed intervals: In addition, $\theta_{ij}(t)$ may exhibit breakpoints resulting from the completion of the traversal of d_{ij} entirely within some δt_τ. Specifically, for a departure time $\alpha \in \delta t_\tau$, the traversal completes at time: $t_{\text{complete}} = \alpha + \theta_{ij}(\alpha) = \alpha + \frac{d_{ij}}{v_\tau}$. If $t_{\text{complete}} \in \delta t_\tau$, then $\theta_{ij}(t)$ has a breakpoint at $t = t_{\text{complete}}$ since the arc is fully traversed within δt_τ. Since such a completion can occur at most once within each interval δt_τ, this introduces at most $n - 1$ additional breakpoints. Hence, summing both contributions, the total number of breakpoints is bounded by: Total breakpoints $\leq \left|\mathscr{B}_{\nu_{ij}}\right| + \left|\mathscr{B}_{\nu_{ij}}\right| = 2\left|\mathscr{B}_{\nu_{ij}}\right|$. Thus, $\theta_{ij}(t)$ has at most $2\left|\mathscr{B}_{\nu_{ij}}\right|$ breakpoints. □

Proposition 3 (Properties of arrival time functions) *Arrival time functions have the following properties:*

1. *$\phi_{ij}\colon \mathscr{H} \to \mathscr{H} \cup \{\infty\}$ with $\phi_{ij}(t) = t + \theta_{ij}(t)$ if in $\mathscr{H}$, and ∞ otherwise.*
2. *$\phi_{ij}(t)$ is CPL, monotonically nondecreasing.*
3. *$\phi_{ij}^{-1}\colon \mathscr{H} \to \mathscr{H} \cup \{-\infty\}$ given by $\phi_{ij}^{-1}(\beta) = \sup\{t \in \mathscr{H} : \phi_{ij}(t) \le \beta\}$ is well-defined.*
4. *The breakpoints of ϕ_{ij} and ϕ_{ij}^{-1} correspond one-to-one to those of θ_{ij}.*
5. *Algorithm 2 retrieves the breakpoints of θ_{ij} and constructs $\Phi_{ij}(t)$—the closed-form of $\phi_{ij}(t)$—in $\mathcal{O}(n \log n)$ time.*
6. *Let $\psi_{i,k}(t)$ denote the extended arrival time function at node n_k along the sequence $\sigma_{i,k}(n)$ with $i < k$. Then, the otpimal extended arrival time function (OEATF) is: $\psi^*_{i,k}(t) = \min_{e \in \mathcal{A}_{k-1,k}} \left\{ \phi_e \circ \psi^*_{i,k-1}(t) \right\},\ \forall t \in \mathscr{H}$.*
7. *Computing the OEATF $\psi^*_{i,k}(t)$ costs $\mathcal{O}\big(|\mathcal{A}_{k-1,k}| \cdot \log |\mathscr{B}_\phi|\big)$.*
8. *Extended arrival time for simple sequences: Let $\sigma_{i,k}(n)$ be a simple sequence. For any $t \in \mathscr{H}$, $\psi_{i,k}(t)$ is infeasible iff $\exists\, \alpha \in \{i, \dots, k-1\}$ with $\psi^*_{i,\alpha}(t) \notin \mathscr{H}$ or $\phi_{\alpha,\alpha+1}(\psi^*_{i,\alpha}(t)) \notin \mathscr{H}$. Otherwise, $\psi^*_{i,k}(t) = \phi_{i,i+1} \circ \phi_{i+1,i+2} \circ \cdots \circ \phi_{k-1,k}(t)$.*
9. *Cost of evaluating $\psi^*_{i,k}$ (simple sequence): For a simple $\sigma_{i,k}(n)$ of length $|\sigma|$, any single value $\psi^*_{i,k}(t)$ can be computed in $\mathcal{O}\big(|\sigma| \cdot \log |\mathscr{B}_\phi|\big)$.*
10. *Uniqueness of the OEATF: The optimal extended arrival time function $\psi^*_{i,k}(t)$ is uniquely defined for all $t \in \mathscr{H}$, taking a finite value in $\mathscr{H}$ when feasible and ∞ otherwise.*
11. *OD optimal extended arrival time function: For any origin o and destination d in $G_{\mathcal{RN}}$, the OEATF satisfies $\psi^*_{o,d}(t) = \min_{\pi \in \Pi(d)} \min_{e \in \mathcal{A}_{\pi,d}} (\phi_e \circ \psi^*_{o,\pi})(t),\ \forall t \in \mathscr{H}$.*
12. *Cost of evaluating $\psi^*_{o,d}$: For any o, d, a single value $\psi^*_{o,d}(t)$ can be computed in $\mathcal{O}\big(|\Pi(d)| \cdot |\mathcal{A}_{\max}| \cdot \log |\mathscr{B}_\phi|\big)$ time.*
13. *Closed-form expression of $\psi^*_{i,k}$ via lower envelope: Let $\Psi_{i,k}$ be the closed form of $\psi_{i,k}$. Then, $\Psi^*_{i,k}(t) = \mathscr{LE}\Big\{\ \Phi_e \circ \Psi^*_{i,k-1}(t)\ :\ e \in \mathcal{A}_{k-1,k}\Big\}$.*
14. *Complexity of constructing $\Psi^*_{o,d}$: The closed form $\Psi^*_{o,d}$ can be built in $\mathcal{O}\big(|\Sigma_{o,d}| \cdot |\sigma| \cdot |\mathscr{B}_\phi|\ +\ n \log n\big)$, where $n \le |\Sigma_{o,d}| \cdot |\mathscr{B}_\phi|$ is the total number of segments sent to the lower envelope.*
15. *OD closed form of $\psi^*_{o,d}$ via lower envelope: Let $\Psi^*_{o,d}$ be the closed form of $\psi^*_{o,d}$. Then, $\Psi^*_{o,d}(t) = \mathscr{LE}\Big(\bigcup_{\pi \in \Pi(d)} \{\ \Phi_e \circ \Psi^*_{o,\pi}(t)\ :\ e \in \mathcal{A}_{\pi,d}\}\Big)$.*
16. *Complexity of constructing $\Psi^*_{o,d}$ The closed form $\Psi^*_{o,d}$ can be built in $\mathcal{O}\big(|\Sigma_{o,d}| \cdot |\sigma| \cdot |\mathscr{B}_\phi|\ +\ n \log n\big)$, where $n \le |\Sigma_{o,d}| \cdot |\mathscr{B}_\phi|$ is the total number of segments sent to the lower envelope.*
17. *Time-window-constrained arrival time function $\widetilde{\phi}_{ij}$: $\widetilde{\phi}_{ij}(t)$ is CPL and monotonically nondecreasing.*

18. *Uniqueness of $\widetilde{\phi}^*_{ij}$: The optimal time-window-constrained arrival time $\widetilde{\phi}^*_{ij}$—as shown in Equation* (20)*—is uniquely attained at $t = a_i$; that is, $\widetilde{\phi}^*_{ij} = \phi_{ij}(a_i)$.*

19. *Breakpoints of $\widetilde{\Phi}_{ij}$: If $\widetilde{\Phi}_{ij}$ is feasible, its breakpoints can be retrieved in $\mathcal{O}(\log |\mathscr{B}_{\Phi_{ij}}| + k)$, where $|\mathscr{B}_{\Phi_{ij}}|$ is the number of breakpoints of Φ_{ij} and k is the number retained in $\widetilde{\Phi}_{ij}$.*

Proof.

1. Immediate from Proposition 2.

2. $\phi_{ij} = t + \theta_{ij}$ is the sum of linear and CPL maps; since $\theta_{ij} \geq 0$, FIFO holds.

3. ϕ_{ij} is continuous and monotone; sublevel sets are closed (nonempty iff $\beta \geq \phi_{ij}(t_{\min})$). The supremum lies in $\mathscr{H}$ or we assign $-\infty$.

   Remark 3 *Although θ_{ij} need not be monotone, $\phi_{ij}(t) = t+\theta_{ij}(t)$ is CPL, monotonically nondecreasing (Proposition 2); hence the generalized inverse ϕ_{ij}^{-1} is well-defined without requiring θ_{ij} to be invertible.*

4. If t_b is a breakpoint of θ_{ij}, then $\phi_{ij}(t) = t + \theta_{ij}(t)$ changes slope at t_b; hence t_b is a breakpoint of ϕ_{ij}. Since ϕ_{ij} is CPL, monotonically nondecreasing, ϕ_{ij}^{-1} is CPL on $\mathrm{Im}(\phi_{ij})$; its breakpoints occur at $\beta_b = \phi_{ij}(t_b)$, giving a one-to-one correspondence with the breakpoints of ϕ_{ij} (and thus of θ_{ij}).

5. **Correctness.** By Proposition 3 and Proposition 4, all breakpoints of ϕ_{ij} (hence of Φ_{ij}) arise from: (i) speed changes of ν_{ij} (exactly $n-1$), and (ii) possible arc-completion times within constant-speed intervals (at most $n-1$). Steps 1–2, therefore collect a set $\mathscr{B}_3$ with $|\mathscr{B}_3| \leq 2(n-1)$. Step 3 sorts $\mathscr{B}_3$; Step 4 discards times outside $\mathscr{H}$; Step 5 builds segments per Definition 3. Thus, the algorithm returns the correct CPL representation Φ_{ij}. **Complexity.** Sorting $\mathscr{B}_3$ with $m \leq 2n$ elements costs $\mathcal{O}(n \log n)$; the remaining scans and constant-time computations are $\mathcal{O}(n)$. Hence, the total is $\mathcal{O}(n \log n)$.

6. Decompose $\sigma_{i,k}(n)$ into $\sigma_{i,k-1}(n)$ and $\sigma_{k-1,k}(n)$. Let $\psi^*_{i,k-1}(t)$ be the optimal arrival at n_{k-1}. For any $e \in \mathcal{A}_{k-1,k}$, $\psi_{i,k}(t) = \phi_e(\psi^*_{i,k-1}(t))$. Taking the minimum over all such arcs yields the stated recursion. If no feasible arc exists, set $\psi^*_{i,k}(t) = \infty$.

7. Assume $\psi^*_{i,k-1}(t)$ is available. For each $e \in \mathcal{A}_{k-1,k}$, evaluating the CPL function ϕ_e at $\psi^*_{i,k-1}(t)$ takes $\mathcal{O}(\log |\mathscr{B}_\phi|)$ via binary search on its breakpoints. A final minimum over $|\mathcal{A}_{k-1,k}|$ values costs $\mathcal{O}(|\mathcal{A}_{k-1,k}|)$. Thus the total is $\mathcal{O}(|\mathcal{A}_{k-1,k}| \cdot \log |\mathscr{B}_\phi|)$.

   Remark 4 (Precomputation of previous DP values) *At stage k, we assume $\psi^*_{i,k-1}(t)$ has already been computed and stored by the recursion. Hence, the work is exactly $|\mathcal{A}_{k-1,k}|$ binary searches (one per arc) plus a $|\mathcal{A}_{k-1,k}|$-way minimum; there is no extra cost to obtain $\psi^*_{i,k-1}(t)$.*

8. Simple sequence means each consecutive pair has exactly one arc (Definition 4), so Property 6 reduces to straight composition. Infeasibility occurs exactly when an intermediate value leaves $\mathscr{H}$; otherwise repeated composition yields the stated form.

Remark 5 (Reduction of OEATF) *In simple sequences, OEATF is pure composition (no minimization), unlike complex sequences with parallel arcs.*

9. By Proposition 8, evaluation is a chain of $|\sigma|$ calls to CPL ϕ's; each costs $\mathcal{O}(\log |\mathscr{B}_\phi|)$ by binary search on breakpoints.

10. Each ϕ_{ij} is CPL, monotonically nondecreasing (Proposition 2); these properties are preserved under composition. The pointwise minimum over a finite family of such functions is again CPL, monotonically nondecreasing, hence a well-defined function on $\mathscr{H}$. For any fixed t, the minimum of finitely many scalars is a single scalar; if no feasible arc applies, we set $\psi^*_{i,k}(t) = \infty$ by convention.

11. Apply Proposition 6 at node d over all predecessors $\pi \in \Pi(d)$ and incoming arcs $e \in \mathcal{A}_{\pi,d}$, noting that $\psi^*_{o,\pi}$ is already available by recursion. Uniqueness follows from Proposition 10.

12. By Proposition 9, for fixed π the inner minimum over $e \in \mathcal{A}_{\pi,d}$ costs $\mathcal{O}(|\mathcal{A}_{\pi,d}| \log |\mathscr{B}_\phi|)$. Summing over all $\pi \in \Pi(d)$ and using $|\mathcal{A}_{\pi,d}| \leq |\mathcal{A}_{\max}|$ yields the stated bound.

13. From Proposition 6, $\psi^*_{i,k}(t) = \min_{e \in \mathcal{A}_{k-1,k}} \phi_e(\psi^*_{i,k-1}(t))$. Replacing each function by its closed form yields a pointwise minimum over $\{\Phi_e \circ \Psi^*_{i,k-1}\}$, i.e., the lower envelope. Compositions and lower envelopes of CPL, monotonically nondecreasing functions remain CPL.

14. Each composition $\Phi_e \circ \Psi^*_{i,k-1}$ produces $\mathcal{O}(|\mathscr{B}_\phi|)$ segments; over $|\mathcal{A}_{k-1,k}|$ arcs this yields $n \leq |\mathcal{A}_{k-1,k}|\,|\mathscr{B}_\phi|$ segments. The lower envelope then costs $\mathcal{O}(n \log n)$ (Algorithm 3).

15. From Proposition 11, $\psi^*_{o,d}(t) = \min_{\pi \in \Pi(d)} \min_{e \in \mathcal{A}_{\pi,d}} \phi_e(\psi^*_{o,\pi}(t))$. Substitute closed forms and identify the pointwise minimum with $\mathscr{L}\mathscr{E}$.

16. For each sequence $\sigma \in \Sigma_{o,d}$ (length $|\sigma|$), composing along the arcs contributes $\mathcal{O}(|\sigma| \cdot |\mathscr{B}_\phi|)$ segments. Summing over $|\Sigma_{o,d}|$ sequences gives $\mathcal{O}(|\Sigma_{o,d}| \cdot |\sigma| \cdot |\mathscr{B}_\phi|)$ segments total, followed by a $\mathcal{O}(n \log n)$ lower envelope (Algorithm 3).

Remark 6 (Breakpoint explosion vs. cancellation) *In worst cases, composing CPL functions can create up to $\Theta(|\mathscr{B}_\phi|^2)$ segments per composition (Foschini, Hershberger, and Suri 2011, Delling and Wagner 2009), and across a sequence the number of candidate segments before the envelope is at most $\mathcal{O}(|\Sigma_{o,d}| \cdot |\sigma| \cdot |\mathscr{B}_\phi|)$. In real road networks, many segments are dominated and never appear in the final lower envelope ("breakpoint cancellation" / "geometric redundancy"), thus the empirical count n of segments passed to the envelope is typically far below these bounds, yielding markedly faster construction times.*

17. Immediate from Proposition 2.

18. Since $\widetilde{\phi}_{ij}$ is continuous and non-decreasing by Proposition 17, its minimum is attained at the left endpoint of its domain, namely $t = a_i$. Hence, $\widetilde{\phi}^*_{ij} = \phi_{ij}(a_i)$.

Remark 7 *With parallel arcs between i and j, then, $\widetilde{\phi}^*_{ij} = \min_{e\in\mathcal{A}_{i,j}} \widetilde{\phi}_e(a_i)$.*

19. Endpoints are a_i and $\min\{\phi_{ij}(b_i),\, b_j\}$ (each by $\mathcal{O}(1)$ interpolation on Φ_{ij}). Interior breakpoints are those of Φ_{ij} strictly between them. Locate the first by a binary search after a_i ($\mathcal{O}(\log|\mathscr{B}_{\Phi_{ij}}|)$), then scan forward collecting the k that lie before $\min\{\phi_{ij}(b_i), b_j\}$ ($\mathcal{O}(k)$). □

Proof of Theorem 3 (Lower envelope via upper convex hull). We prepare the proof by establishing Lemmas 1-2, as follows:

Lemma 1 (Lower envelope and upper convex hull duality) *Let $\mathscr{S}$ be a collection of line segments representing a CPL function in the primal plane. Let $\mathscr{P}$ be the set of dual points obtained by applying point-line duality to the supporting lines of these segments. Then, the lower envelope of $\mathscr{S}$ coincides with the upper convex hull of $\mathscr{P}$.*

Proof. Under the point–line duality $y = ax - b \leftrightarrow (a, b)$, incidence and sidedness are preserved. The supporting line of any segment in the primal corresponds to a point in the dual; the lower envelope of the primal segments is therefore in one-to-one correspondence with the upper convex hull $\mathcal{UH}(\mathscr{P})$ of their dual points. Each edge of $\mathcal{UH}(\mathscr{P})$ corresponds to exactly one segment of the lower envelope, and conversely (De Berg et al. 2008, Har-Peled 2011). □

Lemma 2 (Envelope orientation) *A counterclockwise traversal of $\mathcal{UH}(\mathscr{P})$ yields the segments of $\mathscr{LE}(\mathscr{S})$ ordered from left to right.*

Proof. A segment supported by $y = ax - b$ has slope a, which is the x-coordinate of its dual point (a, b). A counterclockwise traversal of $\mathcal{UH}(\mathscr{P})$ visits points in nondecreasing x, hence in nondecreasing slope, which corresponds to the left-to-right order of segments on the primal lower envelope. □

We are now ready to prove the theorem. Items (1)–(2) follow from Lemmas 1–2. For (3), standard convex-hull algorithms (e.g., Graham scan, monotone chain) build $\mathcal{UH}(\mathscr{P})$ in $\mathcal{O}(n \log n)$ time (De Berg et al. 2008). Extracting the lower envelope from the hull is linear, so the total is $\mathcal{O}(n \log n)$. □

Theorem 9 (Upper Envelope via Lower Convex Hull) *Let Φ_{ij}^{-1} be the collection of inverse arrival-time functions. Then: (1) $\mathscr{UE}(\Phi_{ij}^{-1})$ is obtained by a counterclockwise traversal of $\mathcal{LH}(\mathscr{P})$; (2) it is CPL, with segments ordered left-to-right by increasing slope; and (3) it can be computed in $\mathcal{O}(n \log n)$, where n is the number of segments.*

Proof. We first prove two auxiliary lemmas, Lemmas 3–4.

Lemma 3 (Lower envelope and upper envelope duality via inverse functions) *Let Φ_{ij} be closed-form of a CPL, monotonically nondecreasing arrival-time function and Φ_{ij}^{-1} its inverse. Then the segments of $\mathscr{LE}(\Phi_{ij})$ correspond one-to-one to the segments of $\mathscr{UE}(\Phi_{ij}^{-1})$.*

Proof. For a nondecreasing CPL function, inversion preserves piecewise linearity and breakpoint order. A segment that is active on the lower envelope of Φ_{ij} in the primal becomes an active segment on the upper envelope of Φ_{ij}^{-1} after swapping axes; the correspondence is bijective and preserves adjacency. □

Lemma 4 (Lower convex hull and upper envelope duality) *Let $\mathscr{P}$ be the set of dual points of the supporting lines of Φ_{ij}^{-1}. Then $\mathscr{UE}(\Phi_{ij}^{-1})$ corresponds to the lower convex hull $\mathcal{LH}(\mathscr{P})$ in the dual plane.*

Proof. Under $y = ax - b \leftrightarrow (a, b)$, the upper envelope of a family of lines in the primal corresponds to the lower convex hull of their dual points. Applying this to the supporting lines of the segments of Φ_{ij}^{-1} yields the claim (De Berg et al. 2008, Har-Peled 2011). □
With these lemmas in hand, we now prove the theorem. (1)–(2) follow from Lemmas 3–4. For (3), compute $\mathcal{LH}(\mathscr{P})$ in $\mathcal{O}(n \log n)$; reading off the envelope is linear, so the overall complexity is $\mathcal{O}(n \log n)$. □

Proposition 4 (Constant time construction of $\Gamma_j(.)$) *The function $\Gamma_j(.)$ can be constructed in $\mathcal{O}(1)$.*

Proof. On the finite portion $[t_{\text{init}}, b_j] \subseteq \mathscr{H}$, Γ_j consists of: (i) a constant segment from $(t_{\min}, a_j)$ to (a_j, a_j) (waiting until a_j); (ii) an identity segment from (a_j, a_j) to (b_j, b_j). Since only these two segments are needed, constructing Γ_j takes $\mathcal{O}(1)$ time. □

Lemma 5 (Feasibility of composing two functions) *Let $f_1 \colon \mathscr{H} \to \mathbb{R}$ and $f_2 \colon \mathscr{H} \to \mathbb{R}$ be two CPL nondecreasing functions. Then, the composition $f_2 \circ f_1$ is feasible, i.e., $\mathrm{Dom}(f_2 \circ f_1) \neq \emptyset$, iff $\mathrm{Range}(f_1) \cap \mathrm{Dom}(f_2) \neq \emptyset$. Moreover, $\mathrm{Dom}(f_2 \circ f_1) = f_1^{-1}(\mathrm{Range}(f_1) \cap \mathrm{Dom}(f_2))$.*

Proof. Since $f_2(f_1(t))$ is defined if and only if $f_1(t) \in \mathrm{Dom}(f_2)$, the domain of $f_2 \circ f_1$ is: $\{t \in \mathrm{Dom}(f_1) \mid f_1(t) \in \mathrm{Dom}(f_2)\}$. This is nonempty if and only if $\mathrm{Range}(f_1) \cap \mathrm{Dom}(f_2) \neq \emptyset$, proving feasibility. Since f_1 is non-decreasing and continuous, the pre-image of the intersection $\mathrm{Range}(f_1) \cap \mathrm{Dom}(f_2)$ is an interval.□

Proposition 5 (Complexity of composing CPL functions) *Let f_1, f_2 be CPL nondecreasing with n and m segments, and let $f = f_2 \circ f_1$. Algorithm 6 constructs f in $\mathcal{O}((n+m)\log(n+m))$ time.*

Proof. Extract breakpoints in $\mathcal{O}(n+m)$; map breakpoints of each function through the other via binary searches in $\mathcal{O}(n \log m + m \log n)$; sort/deduplicate in $\mathcal{O}((n+m)\log(n+m))$; then build segments in $\mathcal{O}(n+m)$. Sorting dominates. □

Proposition 6 (Properties of departure time functions) *Departure time functions have the follwing properties:*

1. *Feasibility of $\omega_{ij}(t)$: The function $\omega_{ij}(t)$ is feasible if and only if $\widetilde{\phi}_{ij}(t)$ is feasible.*
2. *$\omega_{ij}(t)$ is CPL, and quasi-monotonic.*
3. *Uniqueness of ω_{ij}^*: If ω_{ij} is feasible, the optimal value $\omega_{ij}^* := \min_{t \in \mathrm{Dom}(\omega_{ij})} \omega_{ij}(t)$ is unique, though the minimizer need not be.*
4. *ω_{ij} via composition: For any arc (i, j), the departure time function satisfies: $\omega_{ij}(t) = (\gamma_j \circ \phi_{ij})(t)$.*
5. *Breakpoints of ω_{ij} equals: $|\mathscr{B}_{\omega_{ij}}| = |\mathscr{B}_{\widetilde{\phi}_{ij}}| + 1$ if $\widetilde{\phi}_{ij}^* \leq a_j$; otherwise $|\mathscr{B}_{\omega_{ij}}| = |\mathscr{B}_{\widetilde{\phi}_{ij}}|$.*

6. *Cost of computing Ω_{ij}: The function $\Omega_{ij} : [a_i, b_i] \to [a_j, b_j]$ —closed-form representation of $\omega_{ij}(t)$—is CPL and obtained by: $\Omega_{ij}(t) = (\Gamma_j \circ \Phi_{ij})(t),\ \forall t \in [a_i, b_i]$ and can be computed in $\mathcal{O}(|\mathscr{B}_{\Phi_{ij}}|)$ time, where $|\mathscr{B}_{\Phi_{ij}}|$ is the number of breakpoints in Φ_{ij}.*

7. *Optimal extended departure time $f^*_{i,k}$: Let $f^*_{i,k-1}(t)\ ,\ t \in [a_i, b_i]$ be the OEDTF at the predecessor node $k-1$, on sequence $\sigma_{i,k}$. Then the single-value optimal departure time at node k is: $f^*_{i,k} = \min_{t\in[a_i,b_i]}\Big(\min_{e\in\mathcal{A}_{k-1,k}}\big(\gamma_k \circ \phi_e \circ f^*_{i,k-1}(t)\big)\Big) = \min\limits_{\substack{t\in[a_i,b_i]\\ e\in\mathcal{A}_{k-1,k}}} \omega_e(f^*_{i,k-1}(t))$.*

8. *Computational complexity of $f^*_{i,k}(t)$ for a single query: The computation of $f^*_{i,k}(t)$ for a single query $t \in [a_i, b_i]$ requires: $\mathcal{O}(|\mathcal{A}_{k-1,k}| \cdot \log|\mathscr{B}_\phi|)$ operations.*

9. *Domain and range of the EDTF for a simple sequence $\sigma_{i,i+2}(n)$ of size two: Let $f_{i,i+2}(t) = \omega_{i+1,i+2}\big(\omega_{i,i+1}(t)\big),\ t \in [a_i, b_i]$, be the EDTF along the simple sequence $\sigma_{i,i+2}(n)$. Define*

$$\mathscr{L} = \max\Big\{\min_{t\in[a_{i+1},b_{i+1}]}\mathrm{Dom}\big(\omega_{i+1,i+2}\big),\ \min_{t\in[a_i,b_i]}\mathrm{Range}\big(\omega_{i,i+1}\big)\Big\},$$

$$\mathscr{U} = \min\Big\{\max_{t\in[a_{i+1},b_{i+1}]}\mathrm{Dom}\big(\omega_{i+1,i+2}\big),\ \max_{t\in[a_i,b_i]}\mathrm{Range}\big(\omega_{i,i+1}\big)\Big\}.$$

Then,

$$\mathrm{Dom}\big(f_{i,i+2}\big) = \big[\omega^{-1}_{i,i+1}(\mathscr{L}),\ \omega^{-1}_{i,i+1}(\mathscr{U})\big],$$

$$\mathrm{Range}\big(f_{i,i+2}\big) = \Big[\min_{t\in\mathrm{Dom}(f_{i,i+2})} f_{i,i+2}(t),\ \max_{t\in\mathrm{Dom}(f_{i,i+2})} f_{i,i+2}(t)\Big].$$

10. *Domain and range of the EDTF for a simple sequence $\sigma_{i,i+L}(n)$: Let $L > 2$ and $f_{i,i+L} = \omega_{i+L-1,i+L} \circ f_{i,i+L-1}$. Then, $\mathrm{Dom}(f_{i,i+L}) = f^{-1}_{i,i+L-1}\Big(\mathrm{Range}(f_{i,i+L-1}) \cap \mathrm{Dom}(\omega_{i+L-1,i+L})\Big)$, i.e.,*

$$\mathrm{Dom}(f_{i,i+L}) = \Bigg[\max\Big(\min_{t\in\mathrm{Dom}(f_{i,i+L-1})}\mathrm{Range}(f_{i,i+L-1}),\ \min_{t\in[a_{i+L-1},b_{i+L-1}]}\mathrm{Dom}(\omega_{i+L-1,i+L})\Big),$$
$$\min\Big(\max_{t\in\mathrm{Dom}(f_{i,i+L-1})}\mathrm{Range}(f_{i,i+L-1}),\ \max_{t\in[a_{i+L-1},b_{i+L-1}]}\mathrm{Dom}(\omega_{i+L-1,i+L})\Big)\Bigg].$$

and

$$\mathrm{Range}(f_{i,i+L}) = \Big[f^*_{i,i+L},\ \max_{t\in\mathrm{Dom}(f_{i,i+L})} f_{i,i+L}(t)\Big], \qquad f^*_{i,i+L} = \min_{t\in\mathrm{Dom}(f_{i,i+L})} f_{i,i+L}(t).$$

11. *OD Optimal Extended Departure Time Function (OD OEDTF): Let $o, d \in \mathcal{N}$. If $f^*_{o,\pi}$ is known for $\pi \in \Pi(d)$, then,*

$$f^*_{o,d} = \min_{t\in[a_o,b_o]}\Bigg(\min_{\pi\in\Pi(d)}\Big(\min_{e\in\mathcal{A}_{\pi,d}}\big(\gamma_d \circ \phi_e \circ f^*_{o,\pi}(t)\big)\Big)\Bigg) = \min_{t\in[a_o,b_o]}\Bigg(\min_{\pi\in\Pi(d)}\Big(\min_{e\in\mathcal{A}_{\pi,d}}\big(\omega_e(f^*_{o,\pi}(t))\big)\Big)\Bigg).$$

12. *Computational complexity of $f^*_{o,d}$: For fixed t, computing $f^*_{o,d}(t)$ costs $\mathcal{O}(\sum_{\pi\in\Pi(d)} |\mathcal{A}_{\pi,d}| \cdot \log|\mathscr{B}_\phi|)$, where $|\mathscr{B}_\phi|$ bounds breakpoints of any ϕ_e.*

13. *Closed-form expression of $f^*_{i,k}$ via lower envelope: Let $\mathscr{F}^*_{i,k}$ be the closed form of the OEDTF at node k. Then: $\mathscr{F}^*_{i,k}(t) = \mathscr{LE}\left(\bigcup_{e\in\mathcal{A}_{k-1,k}} \left\{\Omega_e \circ \mathscr{F}^*_{i,k-1}(t)\right\}\right)$.*

14. *Computational complexity of constructing $\mathscr{F}^*_{i,k}$ For sequence $\sigma_{\alpha,\beta}(n)$, $\mathscr{F}^*_{i,k}(t)$ can be constructed in $\mathcal{O}(|\mathcal{A}_{k-1,k}| \cdot |\mathscr{B}_\Omega| + n\log n)$, where $|\mathscr{B}_\Omega| = \max_{e\in\mathcal{A}_{k-1,k}} |\mathscr{B}_{\Omega_e}|$ and $n \le \sum_{e\in\mathcal{A}_{k-1,k}} |\mathscr{B}_{\Omega_e}|$ is the number of segments sent to the envelope.*

15. *Closed form of $f^*_{o,d}$: Let $\mathscr{F}^*_{o,d}$ be the closed form of $f^*_{o,d}$. Then,*

$$\mathscr{F}^*_{o,d}(t) = \mathscr{LE}\Big(\ \{\ \Omega_e \circ \mathscr{F}^*_{o,\pi}(t)\ :\ \pi \in \Pi(d),\ e \in \mathcal{A}_{\pi,d}\ \}\Big).$$

16. *Cost of constructing $\mathscr{F}^*_{o,d}$: The closed form $\mathscr{F}^*_{o,d}(t)$ can be constructed in $\mathcal{O}\big(|\Sigma_{o,d}|\cdot|\mathscr{B}_\omega|\cdot|\sigma|+n\log n\big)$, where $|\Sigma_{o,d}|$ is the number of valid sequences from o to d, $|\sigma|$ is the maximum sequence length, $|\mathscr{B}_\omega|$ bounds breakpoints per closed-form of departure time function Ω_e, and n is the total number of segments sent to the envelope.*

17. *End-to-end cost of constructing $\mathscr{F}^*_{i,i+L}(t)$ across simple sequences: Let $\sigma_{i,i+L}(n)$ be a simple sequence of L arcs and each Ω_e have at most $|\mathscr{B}_\omega|$ breakpoints. Then $\mathscr{F}^*_{i,i+L}(t)$ can be built in $\mathcal{O}\big(L\cdot|\mathscr{B}_\omega|\cdot\log|\mathscr{B}_\omega|\big)$ time.*

18. *End-to-end cost of constructing $\mathscr{F}^*_{i,i+L}(t)$ across arbitrary sequences: Let $\sigma_{i,k}(n)$ be any sequence in $G_{\mathcal{RN}} = (\mathcal{N},\mathcal{A})$. For each $j \in \{i,\dots,k-1\}$, let $\mathcal{A}_{j,j+1}$ be the set of parallel arcs from n_j to n_{j+1}, and let $A_{\text{tot}} = \sum_{j=i}^{k-1}|\mathcal{A}_{j,j+1}|$, and $|\mathscr{B}_\Omega| = \max_{e\in\mathcal{A}_{j,j+1}}|\mathscr{B}_{\Omega_e}|$, $\forall j \in \{i,\dots,k-1\}$ If $n \le A_{\text{tot}}\cdot|\mathscr{B}_\Omega|$ denotes the total number of linear segments passed to the final lower-envelope, then $\mathscr{F}^*_{i,k}(t)$ can be constructed in $\mathcal{O}\big(A_{\text{tot}}\cdot|\mathscr{B}_\Omega| + n\cdot\log n\big)$ operations.*

Proof.

1. A departure is feasible if and only if the corresponding arrival at node j is feasible.

2. If $\widetilde{\phi}_{ij}(t) \le a_j$, waiting enforces a constant segment (multiple t share the same departure). For $\widetilde{\phi}_{ij}(t) \ge a_j$, $\omega_{ij}(t) = \widetilde{\phi}_{ij}(t)$ and is strictly increasing. In both regions, ω_{ij} inherits continuity and piecewise linearity from $\widetilde{\phi}_{ij}$. Therefore, ω_{ij} is nondecreasing—flat only on intervals where $\widetilde{\phi}_{ij}(t) \le a_j$ and strictly increasing otherwise—i.e., quasi-monotone.

3. By Proposition 2, ω_{ij} is monotonically nondecreasing; hence its minimum value is unique. If a waiting plateau occurs, multiple t can attain that value.

4. The function $\phi_{ij}(t)$ gives the arrival time at node j when departing from node i at time t. Applying

$\gamma_j(\cdot)$ enforces node j's time window constraint. Explicitly: $\omega_{ij}(t) = \begin{cases} a_j, & \phi_{ij}(t) \le a_j, \\ \phi_{ij}(t), & a_j < \phi_{ij}(t) \le b_j, \\ \infty, & \phi_{ij}(t) > b_j. \end{cases}$

This ensures that $\omega_{ij}(t)$ is well-defined and respects j's time window.

5. If the earliest arrival precedes a_j, the waiting plateau introduces one extra breakpoint at a_j; otherwise the breakpoint set is unchanged.

6. Since $\Phi_{ij}(t)$ is piecewise linear with $|\mathscr{B}_{\Phi_{ij}}|$ breakpoints, its evaluation at any $t \in [a_i, b_i]$ takes $\mathcal{O}(\log |\mathscr{B}_{\Phi_{ij}}|)$ time via binary search. Applying Γ_j involves at most two simple cases (waiting or immediate service) and thus takes $\mathcal{O}(1)$ time (see Proposition 4). Therefore, constructing Ω_{ij} requires $\mathcal{O}(|\mathscr{B}_{\Phi_{ij}}|)$ operations.

7. Decompose $\sigma_{i,k}$ into $\sigma_{i,k-1}$ and the parallel arcs $\mathcal{A}_{k-1,k}$. Departing $k-1$ at time $f^*_{i,k-1}(t)$ and traversing arc e yields $f_{i,k}(t) = \gamma_k \circ \phi_e \circ f^*_{i,k-1}(t) = \omega_e\big(f^*_{i,k-1}(t)\big)$. Minimizing over feasible t and e gives the stated single best departure time.

8. For each $e \in \mathcal{A}_{k-1,k}$, evaluating $\phi_e(t)$ via binary search among its $\mathcal{O}(|\mathscr{B}_\phi|)$ breakpoints takes $\mathcal{O}(\log |\mathscr{B}_\phi|)$. The ready-time adjustment Γ_k is $\mathcal{O}(1)$, and finally we take a minimum over the $\mathcal{O}(|\mathcal{A}_{k-1,k}|)$ resulting values in $\mathcal{O}(|\mathcal{A}_{k-1,k}|)$. Altogether, $\mathcal{O}\big(|\mathcal{A}_{k-1,k}| \cdot \log |\mathscr{B}_\phi|\big)$.

9. By Lemma 5 with $f_1 = \omega_{i,i+1}$, $f_2 = \omega_{i+1,i+2}$, feasibility requires $\mathrm{Range}(\omega_{i,i+1}) \cap \mathrm{Dom}(\omega_{i+1,i+2}) \neq \emptyset$, which is the interval $[\mathscr{L}, \mathscr{U}]$ by the endpoint definitions above. Hence, $\mathrm{Dom}(f_{i,i+2}) = \{t : \omega_{i,i+1}(t) \in [\mathscr{L}, \mathscr{U}]\} = [\omega^{-1}_{i,i+1}(\mathscr{L}), \omega^{-1}_{i,i+1}(\mathscr{U})]$. Continuity/monotonicity gives the range as the image of the domain endpoints.

10. Apply Lemma 5 recursively at each extension. Endpoints follow from intersecting consecutive range/domain intervals and mapping back through $f^{-1}_{i,i+L-1}$. The range is the image of the domain under the continuous nondecreasing $f_{i,i+L}$.

11. Any path ends with some $\pi \to d$ via $e \in \mathcal{A}_{\pi,d}$. Depart at o at t, reach π optimally at $f^*_{o,\pi}(t)$, traverse e and enforce γ_d. Minimizing over arcs, predecessors, and t yields the claim.

12. For each $\pi \in \Pi(d)$ and $e \in \mathcal{A}_{\pi,d}$, evaluate $\phi_e\big(f^*_{o,\pi}(t)\big)$ by binary search in $\mathcal{O}(\log |\mathscr{B}_\phi|)$, apply γ_d in $\mathcal{O}(1)$, then take a minimum. Summing the per-arc costs gives the bound.

13. From Proposition 7, $f^*_{i,k}(t) = \min_{e \in \mathcal{A}_{k-1,k}} (\omega_e \circ f^*_{i,k-1})(t)$. Replacing each function by its closed form yields a pointwise minimum, i.e., a lower envelope of CPL functions.

14. Compose each Ω_e with $\mathscr{F}^*_{i,k-1}$ in $\mathcal{O}(|\mathscr{B}_{\Omega_e}|)$ and accumulate $n \le \sum_e |\mathscr{B}_{\Omega_e}|$ segments. Build the lower envelope in $\mathcal{O}(n \log n)$. Summing gives the bound.

15. Directly from Proposition 11 by substituting closed forms and taking the pointwise minimum.

16. Per sequence σ, composing along its $|\sigma|$ arcs costs $\mathcal{O}(|\mathscr{B}_\omega| \cdot |\sigma|)$. Summed over $|\Sigma_{o,d}|$ sequences gives $\mathcal{O}(|\Sigma_{o,d}| \cdot |\mathscr{B}_\omega| \cdot |\sigma|)$. The final envelope over n segments costs $\mathcal{O}(n \log n)$.

17. By Proposition 5, composing two CPL functions with $\leq |\mathscr{B}_\omega|$ segments each costs $\mathcal{O}(|\mathscr{B}_\omega| \log |\mathscr{B}_\omega|)$. A chain of L arcs requires $L-1$ compositions, yielding the stated bound.

18. We build $\mathscr{F}^*_{i,k}$ by extending step by step from n_i to n_k. At each stage $j \to j+1$: (i) *Segment collection:* For each arc $e \in \mathcal{A}_{j,j+1}$, compose Ω_e with the current partial envelope. Each composition yields at most $|\mathscr{B}_\Omega|$ segments, so gathering all $|\mathcal{A}_{j,j+1}|$ of them costs $\mathcal{O}(|\mathcal{A}_{j,j+1}| \cdot |\mathscr{B}_\Omega|)$. (ii) *Lower-envelope:* We then compute the lower envelope of the resulting $n_j \leq |\mathcal{A}_{j,j+1}|\ B$ segments in $\mathcal{O}(n_j \cdot \log n_j)$. Summing over $j = i, \ldots, k-1$ gives $\sum_{j=i}^{k-1} \Big(\mathcal{O}(|\mathcal{A}_{j,j+1}| \cdot |\mathscr{B}_\Omega|) + \mathcal{O}(n_j \cdot \log n_j)\Big) = \underbrace{\mathcal{O}(A_{\text{tot}} \cdot |\mathscr{B}_\Omega|)}_{\text{all compositions}} + \underbrace{\mathcal{O}\Big(\sum_j n_j \cdot \log n_j\Big)}_{\leq \mathcal{O}(n \cdot \log n)}$. Since $\sum_j n_j \leq n$ and $\sum_j n_j \log n_j \leq n \log n$, the total cost is $\mathcal{O}(A_{\text{tot}} \cdot |\mathscr{B}_\Omega| + n \cdot \log n)$, as claimed. □

Definition 6 (Logical-arc abstraction) *Let $G_{\mathcal{RN}} = (\mathcal{N}, \mathcal{A})$ be the underlying TD road network of our problem, with $\mathcal{N} = \mathcal{S} \cup \mathcal{J}$ (stops $\mathcal{S}$, junctions $\mathcal{J}$). Define the set of* road-induced *logical-arc candidates*

$$\mathcal{E}_{\log} \subseteq \mathcal{S} \times \mathcal{S}, \qquad \mathcal{E}_{\log} := \{(i,j) : \textit{there exists a directed path } i \rightsquigarrow j \textit{ in } G_{\mathcal{RN}}\}.$$

For $(i,j) \in \mathcal{E}_{\log}$, let $\mathscr{G}_{i,j} \subseteq G_{\mathcal{RN}}$ be the subnetwork from i to j. Interior intersections have no time windows ($\Gamma_v = \mathrm{ID}_{\mathscr{H}}$ for $v \in \mathcal{J}$, i.e., $\Gamma_v(t) = t$ for all $t \in \mathscr{H}$); stops inherit request windows. Let $\Psi^*_{i,j}$ be the optimal extended arrival-time function on $\mathscr{G}_{i,j}$. Let $\mathcal{L}$ be the logical-arc abstraction operator; then,

$$\mathcal{L} : \mathcal{G}_{\mathrm{RN}} \to \mathcal{E}_{\log}, \qquad \mathcal{L}(\mathscr{G}_{i,j}) := \Omega_{i,j} \equiv \Gamma_j \circ \Psi^*_{i,j}.$$

We write $\mathscr{G}_{i,j} \rightsquigarrow (i,j)$ to indicate that the subnetwork induces the logical arc (i,j) with transfer map $\Omega_{i,j}$.

Proposition 7 (Equivalence of the Logical-Arc Abstraction) *Assume FIFO holds and interior junctions have no time windows ($\Gamma_v = \mathrm{ID}_{\mathscr{H}}$, $\forall v \in \mathcal{J}$). Let $\mathcal{L}$ be the logical-arc abstraction of Definition 6, so that $\Omega_{\pi,d} = \mathcal{L}(\mathscr{G}_{\pi,d}) = \Gamma_d \circ \Psi^*_{\pi,d}$, for any $(\pi,d) \in \mathcal{E}_{\log}$. Then, for any destination d,*

$$\mathscr{F}^*_{o,d}(t) = \mathscr{L}\mathscr{E}\left(\bigcup_{\pi \in \Pi(d):(\pi,d) \in \mathcal{E}_{\log}} \{\Omega_{\pi,d} \circ \mathscr{F}^*_{o,\pi}(t)\}\right) = \mathscr{L}\mathscr{E}\left(\bigcup_{\pi \in \Pi(d):(\pi,d) \in \mathcal{E}_{\log}} \bigcup_{e \in \mathcal{A}_{\pi,d}} \{\Gamma_d \circ \Phi_e \circ \mathscr{F}^*_{o,\pi}(t)\}\right),$$

and the right-hand side coincides with Equation (32)*.*

Proof. By Definition 6 and $\Gamma_v = \mathrm{ID}_{\mathscr{H}}$ on interior nodes, the subnetwork response between stops i and j is the lower envelope of arc-level functions:

$$\Psi^*_{i,j} = \mathscr{L}\mathscr{E}(\{\Phi_e : e \in \mathcal{A}_{i,j}\}).$$

Since FIFO implies Φ_e and Γ_j are nondecreasing, composition distributes over lower envelopes; hence,

$$\Omega_{i,j} = \Gamma_j \circ \Psi^*_{i,j} = \Gamma_j \circ \mathscr{LE}\{\Phi_e\} = \mathscr{LE}\{\Gamma_j \circ \Phi_e : e \in \mathcal{A}_{i,j}\}. \tag{40}$$

Let $g_\pi = \mathscr{F}^*_{o,\pi}$. Applying (40) with $j = d$ and using the envelope–composition exchange with a nondecreasing inner map g_π yields, for any $(\pi, d) \in \mathcal{E}_{\log}$,

$$\Omega_{\pi,d} \circ g_\pi = \left(\mathscr{LE}\{\Gamma_d \circ \Phi_e : e \in \mathcal{A}_{\pi,d}\}\right) \circ g_\pi = \mathscr{LE}\{\Gamma_d \circ \Phi_e \circ g_\pi : e \in \mathcal{A}_{\pi,d}\}. \tag{41}$$

Finally, taking the lower envelope over all admissible predecessors $\pi \in \Pi(d)$ with $(\pi, d) \in \mathcal{E}_{\log}$ and using (41) gives:

$$\mathscr{LE}\left(\bigcup_{\pi}\{\Omega_{\pi,d} \circ g_\pi\}\right) = \mathscr{LE}\left(\bigcup_{\pi}\bigcup_{e\in\mathcal{A}_{\pi,d}}\{\Gamma_d \circ \Phi_e \circ g_\pi\}\right),$$

which is exactly Equation (32), with subnetworks collapsed to logical-arc maps on the left and physical arcs on the right. □

E Mathematical Contents of the Optimization Phase Section

Proof of Theorem 5 (Optimal cluster-to-cluster departure time functions). Consider the TD road network $G_{\mathcal{RN}} = (\mathcal{N}, \mathcal{A})$. For any OD pair (o, d), Proposition 15 gives:

$$\mathscr{F}^*_{o,d}(t) = \mathscr{LE}\left(\bigcup_{\pi\in\Pi(d)}\bigcup_{e\in\mathcal{A}_{\pi,d}}\{\Gamma_d \circ \Phi_e \circ \mathscr{F}^*_{o,\pi}(t)\}\right) = (32).$$

By the logical-arc abstraction (Def. 6) and its equivalence (Prop. 7), each subnetwork between nodes in consecutive clusters in a route sequence $\sigma(r)$ collapses to its logical arc map Ω: for any nodes $p \in \nu(k-1)$, $s \in \nu(k)$, $\mathscr{F}^{p,s}_{\nu(k-1),\nu(k)} = \Omega_{p,s} = \Gamma_s \circ \Psi^*_{p,s}$. Substituting $\Omega_{p,s}$ for the corresponding arc set in (32) and taking the envelope over all $p \in \nu(k-1)$ yields $\mathscr{F}^{l,s}_{\nu(i),\nu(k)}(t) = \mathscr{LE}\left(\bigcup_{p=0}^{|\nu(k-1)|-1}\left\{\mathscr{F}^{p,s}_{\nu(k-1),\nu(k)} \circ \mathscr{F}^{l,p}_{\nu(i),\nu(k-1)}(t)\right\}\right)$ fora all $l \in \{0, 1, \ldots, |\nu(i)| - 1\}$ and $s \in \{0, 1, \ldots, |\nu(k)| - 1\}$, which is (37). □

Proof of Theorem 6 (Equivalence to Bellman-Ford DP under uniform time cost). In this case, the arc transition cost $\alpha\ c_{ij}\ \theta_{ij}(t) + \beta\ w_j\ \chi_j(t)$ reduces to $\theta_{ij}(t) + \chi_j(t)$, i.e., total journey time. Thus, both models yield the same minimizer t^*. □

Proof of Theorem 7 (Optimality and correctness of the LFP algorithm). The algorithm builds $\mathscr{F}^{l,s}_{\nu(i),\nu(j)}$ by the recursion of Theorem 5.

(1) Base case. For the first service cluster $\nu(1)$, the predecessor is the depot only. Thus, for each $s \in \nu(1)$, $\mathscr{F}^{0,s}_{\nu(\text{origin\_index}),\nu(1)}(t) = \Omega_{0,s}(t) = \Gamma_s \circ \Psi^*_{0,s}(t)$; no envelope is needed.

(2) Inductive step. For $k > 1$ and each $s \in \nu(k)$: retrieve $\Psi^*_{p,s}$ and Γ_s, set $\Omega_{p,s} = \Gamma_s \circ \Psi^*_{p,s}$, compose with $\mathscr{F}^{l,p}_{\nu(\text{origin\_index}),\nu(k-1)}$ for all $p \in \nu(k-1)$, and take the lower envelope:

$$\mathscr{F}^{l,s}_{\nu(\text{origin\_index}),\nu(k)}(t) = \mathscr{LE}\Big\{\Omega_{p,s} \circ \mathscr{F}^{l,p}_{\nu(\text{origin\_index}),\nu(k-1)}(t) \ : \ p \in \nu(k-1)\Big\}.$$

(3) Termination. If all entries for some $\nu(k)$ are infeasible/empty, no feasible schedule exists and the algorithm returns **false**; otherwise the matrix $\boldsymbol{\mathscr{F}}$ is filled.

(4) Conclusion. By construction this matches the Bellman–Ford recursion on the layered route network with waiting and envelopes. Hence, the algorithm is correct, and propagates optimal extended departure time envelopes. □

Proposition 8 (Cost of solving DP via LFP: full $\boldsymbol{\mathscr{F}}$ matrix) *The LFP algorithm fills $\boldsymbol{\mathscr{F}}$ for all $0 \le i \le j \le \mathscr{R} - 1$. Worst-case times:* ***Precomputed:*** $\mathcal{O}(\mathscr{R}^2\, n^3 \log n)$*;* ***On-the-fly:*** $\mathcal{O}(\mathscr{R}^2\, n^3 \log n + \mathscr{R}^2\, n^2\, \mathscr{B})$.

Proof. There are $\Theta(\mathscr{R}^2)$ cluster pairs. For one pair: up to n^2 (p, s) blocks; each block builds the envelope of up to n CPL candidates in $\mathcal{O}(n \log n)$, giving $\mathcal{O}(n^3 \log n)$ per pair. Summing yields $\mathcal{O}(\mathscr{R}^2 n^3 \log n)$. In on-the-fly mode, each $\Omega_{p,s}$ is constructed once at cost $\mathcal{O}(\mathscr{B})$ across $\Theta(\mathscr{R}^2 n^2)$ pairs, adding $\mathcal{O}(\mathscr{R}^2 n^2 \mathscr{B})$.□

Remark 8 (Amortized $\mathcal{O}(1)$ Lookup in On-the-fly Mode) *Each $\Omega_{p,s}$ is built once (first use) and cached; later uses are $\mathcal{O}(1)$. The $\mathcal{O}(\mathscr{R}^2 n^2 \mathscr{B})$ term counts each build once.*

Proposition 9 (Complexity via LFP: single OD pair) *For $(\nu(i), \nu(j))$ with $j > i$ and $(j - i)$ layers:* ***Precomputed:*** $\mathcal{O}((j-i)\, n^3 \log n)$*;* ***On-the-fly:*** $\mathcal{O}((j-i)\, n^3 \log n + (j-i)\, n^2\, \mathscr{B})$.

Proof. Per layer: $\mathcal{O}(n^2)$ blocks $\times \mathcal{O}(n \log n) = \mathcal{O}(n^3 \log n)$. On-the-fly adds $\mathcal{O}(n^2 \mathscr{B})$ builds per layer. □

Corollary 1 (LFP complexity, singleton clusters: full matrix) *If each cluster has one node ($n = 1$):* ***Precomputed:*** $\mathcal{O}(\mathscr{R}^2)$*;* ***On-the-fly:*** $\mathcal{O}(\mathscr{R}^2\, \mathscr{B})$.

Corollary 2 (LFP complexity, singleton clusters: single pair) *For a single $(\nu(i), \nu(j))$ with $j > i$ and $n = 1$:* ***Precomputed:*** $\mathcal{O}(j - i)$*;* ***On-the-fly:*** $\mathcal{O}((j-i)\, \mathscr{B})$.

Proof of Theorem 8 (Optimality and correctness of the PO-BTP algorithm). **Order.** Post-order visits each node after its children. Leaves $[i, i+1]$ compute/store Ω_e. A parent $[i, i+2]$ composes its children in the correct order to obtain $\mathscr{F}_{\nu(i),\nu(i+2)} = \mathscr{F}_{\nu(i+1),\nu(i+2)} \circ \mathscr{F}_{\nu(i),\nu(i+1)}$. By induction this yields at the root $\mathscr{F}_{\nu(o),\nu(d)} = \mathscr{F}_{\nu(\texttt{mid}),\nu(d)} \circ \mathscr{F}_{\nu(o),\nu(\texttt{mid})}$. **Envelopes.** `processNode(...)` dispatches: (i) `if_feas_compute_omega(...)` at leaves (store Ω_e); (ii) `if_feas_compute_composite_omega_omega(...)` at parents of two leaves (Theorem 5, Eq. 37); (iii) `if_feas_compute_composite_omega_F(...)` when left child is internal, right is leaf (applies e.g. for $\mathscr{F}_{\nu(6),\nu(9)}$ in Fig. 8); (iv) `if_feas_compute_composite_F_F(...)` when both children are internal. Indices are aligned in each case and lower envelopes are applied with feasibility/dominance. Therefore both chaining and envelope propagation are correct, and propagating optimal departure time envelopes. □

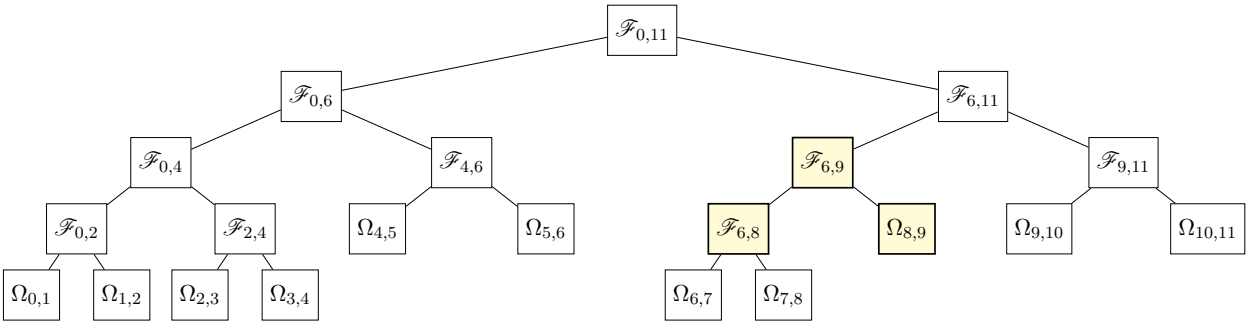

Figure 8: PO-BTP invokes `if_feas_compute_composite_omega_F(...)` to compute $\mathscr{F}_{\nu(6),\nu(9)}$.

Proposition 10 (Cost of PO-BTP) *For a route $\sigma(r)$ and caching each Ω_{ij} once, computing the depot-to-depot function $\mathscr{F}_{\nu(0),\nu(\mathscr{R}-1)}$ costs:* ***Precomputed:*** $\mathcal{O}(\mathscr{R}\, n^3 \log n)$*;* ***On-the-fly:*** $\mathcal{O}(\mathscr{R}\, n^2(\mathscr{B} + n \log n))$.

Proof. Each internal tree node spans a cluster-pair $(\nu(i), \nu(k))$. To build $\mathscr{F}_{\nu(i),\nu(k)}$ we consider all n^2 origin–successor pairs (l, s). For each pair we: (i) Fetch two stored functions from the matrix in $\mathcal{O}(1)$ each. (ii) Compose them (lower-envelope of up to n candidates) in $\mathcal{O}(n \log n)$. (iii) If Ω was not yet built for a given arc (p, s), we incur an extra $\mathcal{O}(\mathscr{B})$ once. Hence, each (l, s) costs $\mathcal{O}(\mathscr{B} + n \log n)$, giving $\mathcal{O}(n^2(\mathscr{B} + n \log n))$ per node. With $\mathscr{R} = \mathcal{O}(|\sigma(r)|)$ internal nodes, the claimed bounds follow. □

Corollary 3 (PO-BTP, singleton clusters) *If each cluster is a singleton, then* ***Precomputed:*** $\mathcal{O}(\mathscr{R})$*;* ***On-the-fly:*** $\mathcal{O}(\mathscr{R}\,(\mathscr{B} + \log \mathscr{R}))$.

Proof. Precomputed: compositions are constant-time lookups across $\mathscr{R}$ nodes. On-the-fly: each node builds Ω once in $\mathcal{O}(\mathscr{B})$ and composes two small CPLs in $\mathcal{O}(\log \mathscr{R})$; sum over $\mathscr{R}$. □

Proposition 11 (Full-matrix complexity of PO-BTP) *Filling all $\{\mathscr{F}_{\nu(i),\nu(j)}\}_{0 \le i < j < \mathscr{R}}$ by querying each (i, j) with caching yields:* ***Precomputed:*** $\mathcal{O}(\mathscr{R}^2\, n^2\, \log n\, \log \mathscr{R})$*;* ***On-the-fly:*** $\mathcal{O}(\mathscr{R}^2\, n^2\, (\mathscr{B} + n \log n)\, \log \mathscr{R})$.

Proof. Each of the $\binom{\mathscr{R}}{2} = \mathcal{O}(\mathscr{R}^2)$ queries (i, j) is answered in one of two ways: If $\mathscr{F}_{\nu(i),\nu(j)}$ was already stored (either as a tree node or from a prior query), it is fetched in $\mathcal{O}(1)$. Otherwise, we invoke the bidirectional search: it visits $\mathcal{O}(\log \mathscr{R})$ tree nodes, retrieves at most $\mathcal{O}(\log \mathscr{R})$ stored functions, and composes them in sequence. Each composition of two cluster-to-cluster functions costs $\mathcal{O}(n^2 \log n)$, and may incur an $\mathcal{O}(\mathscr{B})$ arc-cost if an Ω is newly built (once per arc). Hence each cache-miss query costs $\mathcal{O}\big(\log \mathscr{R} \times n^2 \log n\big)$ (precomputed) or $\mathcal{O}\big(\log \mathscr{R} \times n^2(\mathscr{B} + n \log n)\big)$ (on-the-fly), summing over $\mathcal{O}(\mathscr{R}^2)$ queries yields the stated bounds. □

Corollary 4 (Full-matrix, singleton clusters) *With singleton clusters:* ***Precomputed:*** $\Theta(\mathscr{R}^2)$*;* ***On-the-fly:*** $\Theta(\mathscr{R}^2 \log \mathscr{R})$.

Proof. Here $n = 1$; compositions are $\mathcal{O}(1)$ (precomputed) or $\mathcal{O}(\log \mathscr{R})$ (on-the-fly). There are $\Theta(\mathscr{R}^2)$ queries. □

F Mathematical Contents of the Post-Processing Section

Proposition 12 (Correctness of optimal node selection-Algorithm 11) *Algorithm 11 returns, for each cluster $\nu(j)$, a node s_j^* with $\mathscr{F}^{s_j^*,0}_{\nu(j),\nu(d)}\big(\mathscr{F}^{0,s_j^*}_{\nu(o),\nu(j)}(t^*)\big) = \mathscr{F}^{0,0}_{\nu(o),\nu(d)}(t^*)$.*

Proof. By DP, $\mathscr{F}^{0,0}_{\nu(o),\nu(d)}(t) = \min_{s\in\nu(j)} \mathscr{F}^{s,0}_{\nu(j),\nu(d)}\big(\mathscr{F}^{0,s}_{\nu(o),\nu(j)}(t)\big)$. At $t = t^*$, Algorithm 11 evaluates all candidates and selects a minimizer s_j^*. □

Proposition 13 (Complexity of optimal node selection-Algorithm 11) *If each envelope evaluation $\mathscr{F}^{u,v}_{\nu(i),\nu(j)}(x)$ is $\mathcal{O}(\log \mathscr{B})$, the total time is $\mathcal{O}(L\,n\,\log \mathscr{B})$ and extra space $\mathcal{O}\big(\sum_j |\nu(j)|\big)$.*

Proof. There are $\mathcal{O}(L\,n)$ candidates; each uses two $\mathcal{O}(\log \mathscr{B})$ lookups. □

Proposition 14 (Correctness of path recovery-Algorithm 12) *Assuming $\mathscr{F}$ stores exact lower envelopes for each $(\nu(k), \nu(k+1))$, Algorithm 12 computes labels so that $\mathrm{arr}_{u_{k+1}} = \mathscr{F}^{u_k,u_{k+1}}_{\nu(k),\nu(k+1)}(\mathrm{dep}_{u_k})$, $\mathrm{dep}_{u_{k+1}} = \max\{\mathrm{arr}_{u_{k+1}}, a_{u_{k+1}}\}$, and ends with $\mathrm{dep}_{sink} = \mathscr{F}^{0,0}_{\nu(0),\nu(2L+1)}(t^*)$.*

Proof. This is a forward relaxation on the layered directed acyclic graph (DAG); by induction on k, we maintain that dep_u is the earliest feasible departure from u. At the last layer, we recover the global optimum.□

Proposition 15 (Complexity of path recovery-Algorithm 12) *If each envelope lookup is $\mathcal{O}(\log \mathscr{B})$, then Algorithm 12 runs in $\mathcal{O}(L\,n^2 \log \mathscr{B})$ time and $\mathcal{O}\big(\sum_k |\nu(k)|\big)$ extra space.*

Proof. There are $\mathcal{O}(L\,n^2)$ inter-cluster arcs; each costs one $\mathcal{O}(\log \mathscr{B})$ query plus constant bookkeeping. □