

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/384240572>

Fast Computation of Ranges for Electric Vehicles with Contraction Hierarchies

Conference Paper · September 2024

CITATIONS

0

READS

94

2 authors:



Noémie Meunier

AISIN Europe

5 PUBLICATIONS 51 CITATIONS

[SEE PROFILE](#)



Jean-Sébastien Gonsette

AISIN AW, Europe

5 PUBLICATIONS 2 CITATIONS

[SEE PROFILE](#)

30th ITS World Congress, Dubai, UAE, 16-20 September 2024

Paper ID 212

Fast Computation of Ranges for Electric Vehicles with Contraction Hierarchies

Noémie Meunier^{1*}, Jean-Sébastien Gonsette²

1. AISIN Technical Center Europe, Belgium – noemie.meunier@aisin-europe.com

2. AISIN Technical Center Europe, Belgium – jean-sebastien.gonsette@aisin-europe.com

Abstract

This paper explores the usage of Contraction Hierarchies (CH) algorithm for computing reachable polygons in Electric Vehicle (EV) navigation systems, addressing the driver anxiety related to battery limits and sparse charging stations. The study examines practical challenges in determining the content of those polygons and introduces a cell-based acceleration method within a pure CH router framework. The method also accommodates two join metrics: driving time and energy consumption, reflecting EVs' ability to regenerate energy. This approach aims to enhance route planning algorithms, contributing to stress-free and efficient EV usage.

Keywords

Cells, Binary Metric, Dijkstra

Introduction

The rise of Electric Vehicles (EV) plays a vital role in transitioning to sustainable mobility, but introduces several technical challenges to the automotive industry, including their navigation systems. These systems must now accommodate EV-specific use cases, addressing driver concerns like battery capacity and scarce charging stations. One way to tackle this situation consists in displaying a reachable polygon in the navigation system, centered on the vehicle's current position. This visual representation of the area reachable with the current battery charge aids drivers in planning routes and identifying recharge opportunities, leading to more efficient and stress-free EV usage.

Computing this range polygon in a reasonable time is more challenging than the traditional shortest path navigation systems are used to process. Indeed, the number of nodes at stake can be enormous for wide ranges on continental-size maps, easily reaching several millions. Acceleration techniques are thus required to compute such ranges in a tractable yet accurate way.

In this paper, we explore the theoretical and practical aspects of computing those polygons by extending a router working with the *Customizable Contraction Hierarchies* (CCH) algorithm. The study supports several key aspects that impact the energy consumption of EVs, like the topography, traffic conditions and battery characteristics. In the following sections, we will detail the algorithms for calculating the reachable ranges, modelling considerations, as well as the practical implementation of this approach.

Related Works

There has been significant algorithmic advancement in route planning for road networks in recent years. While Dijkstra's algorithm [1] can theoretically solve the problem optimally on road network graphs, it is often too slow in practice for real-world applications. To address this, speedup strategies with an offline preprocessing phase have been introduced to accelerate query responses, generally with static arc costs for travel times [2]. In this context, Contraction Hierarchies (CH) is a technique that contracts the graph first, thereby reducing the search scope and subsequent query time. A variation known as Customizable Contraction Hierarchies (CCH) further enhances it by allowing adjustments to arc costs without redoing the contraction process. Initially focused on point-to-point shortest paths, applications now include batched shortest paths [3], electric vehicle (EV) routing [4, 5, 6], and isochrone analysis. The concept of isochrones was introduced in [7] and further elaborated in [8, 9], where the authors suggest accelerating their computation using cells. Computation of such isochrones in the special case of reachable ranges, given some battery level, requires additional development.

Contribution

This paper explores the usage of a cell-based method for computing achievable ranges within a pure CCH router framework, maintaining its efficiency in point-to-point and batched shortest path calculations. We describe how a certain kind of cell can be defined naturally in a contraction hierarchy, by leveraging on nested dissections to define its contraction order. We first study range computation involving a single scalar metric, before turning to the more complex case where two metrics are at stake. Using two metrics is essential when the shortest paths followed by the drivers and the allocated budget are not related to each other, such as minimizing driving time while budgeting energy consumption. We finally investigate the specificities of EVs, whose range computation must cope with their battery regeneration capabilities in down slopes.

Preliminary

The road network we operate on is classically modelled with a directed graph $G = (V, E)$, made of a set of n vertices V and a set of edges E . A nonnegative cost function is used to assign a positive value to each edge. Without loss of generality, let us take the travel time function $t: E \rightarrow \mathbb{R}^+$ such that $t(e) \in \mathbb{R}^+, \forall e \in E$. The graph is assumed to be strongly connected without multi-edges. An s - t path in G is a finite sequence of vertices $\rho = \rho_0 \dots \rho_k \in V^+$ such that $(\rho_i, \rho_{i+1}) \in E, \forall i \in \{0, \dots, k-1\}$, $\rho_0 = s$ and $\rho_k = t$. The cost of the path ρ w.r.t. t is defined as the sum of the weights of each edge it contains. We can formally define this cost function $\text{Cost}_t: V^+ \rightarrow \mathbb{R}^+$ for all path $\rho \in V^+$ such that $\text{Cost}_t(\rho) = \sum_{i=0}^{k-1} t(\rho_i, \rho_{i+1})$. A shortest s - t path is a path with minimum cost from s to t . This cost is denoted $\mathcal{S}_t^G(s, t)$ and can be computed with the well-known *Dijkstra* algorithm [1].

Acceleration techniques based on *Contraction Hierarchies* (CH) for shortest path computations were introduced in [10]. The basic idea consists in operating on a contracted graph $G^* = (V, E^*)$ instead of the original graph G . The contraction is a systematic procedure that defines an additional set of edges E^+ (also called *shortcuts*) of the graph, making the final undirected graph G^* *chordal*. This necessitates ordering the vertices using a specific vertex order $\pi: \{1, \dots, n\} \rightarrow V$, while the inverse π^{-1} of this bijection assigns each vertex a *rank*. This ranking allows building G^* and to split it into two directed graphs G^\wedge and G^\vee . The set of edges in E^\wedge of the *upward* graph $G^\wedge = (V, E^\wedge)$ is defined as $E^\wedge = \{(u, v) \in E \cup E^+ \mid \pi^{-1}(u) < \pi^{-1}(v)\}$, i.e., the set of edges

joining the vertices by increasing rank. A similar definition can be given to the set of edges E^V of the downward graph G^V , joining the vertices by decreasing rank. A new time cost function $t^+: E \cup E^+ \rightarrow \mathbb{R}^+$ is defined on G^* to preserve the shortest path property in G on shortcut edges E^+ , i.e., the cost of each edge $(u, w) \in E^+$ is given by $\mathcal{S}_t^G(u, w)$. This weighting is called customization and is at the basis of the *Customizable Contraction Hierarchies* (CCH) [11]. CCH have the nice property that, for any shortest $s - t$ path ρ_{st} in G , there exists a cost equivalent path in G^* that can be split into an upward path ρ_u in G^A , followed by a downward path ρ_d in G^V . This forms an *up-down path* where the vertices occur by increasing rank in ρ_u , then by decreasing rank in ρ_d . It is then possible to show that given a directed graph $G = (V, E)$, for all $s, t \in V$ there exists an up-down path $\rho = \rho_u \rho_d$ with ρ_u (resp. ρ_d) a path in G^A (resp. G^V) and $Cost_{t^+}(\rho) = \mathcal{S}_t^G(s, t)$.

Range computation with a single scalar metric

Any shortest path in G can be computed by performing two classical Dijkstra searches, respectively in G^A from the source vertex and backward in G^V from the target. A solution is found when the two exploration sets meet at some top vertex. However, the computation of a range is different in the sense that there is no such target. Instead, we want to collect all the vertices that are reachable from the source given a certain budget with the aim of computing a polygon that highlights the accessible area around some vehicle position. Formally, given a directed graph $G = (V, E)$, a time function $t: E \rightarrow \mathbb{R}^+$, and a time budget $b \in \mathbb{R}^+$, a vertex $t \in V$ is *time-reachable from s in G for b* iff $\mathcal{S}_t^G(s, t) \leq b$.

Let us denote by $Range(G, s, t, b)$ the set of time-reachable vertices from s in G within budget b . A vanilla implementation of this range computation using CCH involves two steps. During the upward phase, a Dijkstra algorithm is used to relax the vertices in G^A , starting from s . The total cost required to reach every vertex in the exploration set is maintained and the algorithm stops when there is no more vertex under the budget in the priority queue. Next, in the downward phase, a similar algorithm is used to relax the vertices in G^V , starting from all the vertices hit during the first step. Algorithms 1 and 2 give a possible implementation in pseudo-code. Function *QueryRangeDown* is not given but is almost identical to *QueryRangeUp*.

Algorithm 1: QueryRange (G^*, s, b)

```

 $Q \leftarrow \{(s, 0)\}$  // Priority queue keyed by cost.
 $X \leftarrow \{(s, 0)\}$  // Exploration set.
 $S \leftarrow QueryRangeUp(G^A, b, Q, X)$ 
 $Q.insert(X)$ 
 $S \leftarrow S \cup QueryRangeDown(G^V, b, Q, X)$ 
return S

```

Algorithm 2: QueryRangeUp(G^A, b, Q, X)

```

 $S \leftarrow \emptyset$  // Reachable set.
while  $Q \neq \emptyset$  do
     $(u, m) \leftarrow Q.\min()$  // Vertex  $u$  with minimal metric cost  $m$  in  $Q$ .
     $S \leftarrow S \cup \{u\}$  // All the dequeued nodes are reachable.
    foreach  $(u, w) \in E^A$  do
         $m' \leftarrow m + t^+(u, w)$  // Adjacent vertices in the up graph.
        if  $m' > b$  continue // Do not relax vertices that are out of budget.
        if  $w \in X$  and  $X[w].m > m'$  do // Relaxation.
             $Q.update(w, m')$ 
             $X[w].m \leftarrow m'$ 
        else if  $w \notin X$  do

```

$Q.insert(w, m')$ $X \leftarrow X \cup \{(w, m')\}$ return S	// Return the set of reachable nodes.
---	---------------------------------------

Cell-based acceleration method

CH significantly speeds up vertex relaxation in the upward graph, as the number of vertices to consider decreases exponentially fast in this direction. However, computing a range lacks a target vertex, since the whole point is precisely to compute those that are reachable. This renders the algorithm's second step in the downward graph inefficient due to the increasing vertex count when moving forward in G^V . Additionally, large queries on continental-size maps may involve collecting several million vertices, making their computation challenging within a short timeframe.

The idea we propose to accelerate the exploration of the downward graph is to define *cells* in G^V , that are a collection of vertices we can potentially validate at once when evaluating vertices reachability during the relaxation step of a query. This means that when a vertex belonging to a cell is settled, we must have a criterion enabling to establish with certainty if the whole content of the cell is reachable. The positive case saves us the burden to continue the cell exploration, hence the speedup. In the other case, the relaxation must continue.

To understand how the cells are defined, it is important to recall that CH operates on ordered vertices and that the contracted graph built on them exhibits a tree structure that is called an *Elimination Tree* (ET). A good vertex ordering is paramount to get good performances at the query time and this can be achieved with a well-balanced ET. There are a lot of ways to define an ordering, but we chose to use one resulting from a spatial and iterative dissection of the graph [12]. Figure 1 illustrates the first two steps of this iterative process.

Given a graph $G = (V, E)$, a first vertex separator $S \subset V$ is computed at step 1 to get the two disconnected subgraphs G_A and G_B composed of the respective vertex sets A and B . The most important vertices of the final ordering (i.e., the highest numbers from 850 to 900 in Figure 1) are mapped onto this separator, while the vertices of the two subgraphs share the bottom level. The resulting ET forms a linear sequence of vertices belonging to S at the top, opening on the two partially numbered graphs G_A and G_B . This process continues recursively with each subgraph until their size is so reduced that it becomes possible to optimally order their vertices trivially, yielding the final ordering used to contract G into G^* .

Because the graph G we operate on is planar, each subgraph resulting from a cut at every step of the vertex ordering process forms a geographically consistent and locally connected vertex set. Therefore, they are naturally good candidates for the cells we need to speed up range query computations. In addition, each subgraph's vertex set maps to a continuous range of vertices, requiring only two parameters for the cell encodings (i.e., the lowest vertex order and the subgraph vertex count). Finally, two different subgraphs are only connected through their shared ancestors, greatly simplifying the range query process when a vertex is relaxed in a cell of the downward graph. Indeed, when the relaxation hits a vertex in such a cell, it remains confined inside until reaching the bottom of the graph. All the other cells become completely irrelevant as they are unattainable in the strictly downward direction.

The vertex set of each subgraph at every step of the vertex ordering process can be used as a cell. However, there is some trade-off to be found to achieve good performances. Smaller cells positively impact the number of times we can validate them at once, but this reduces the speedup by postponing the moment they are hit

during a query. Bigger cells are hit quickly during a query, but the chances they are fully reachable are slimmer, which means losing the benefit of this mechanism. Practically, in this paper, cells are made up of subgraphs whose number of vertices falls below some threshold when they are sliced. For our example in Figure 1, if this threshold is 200, then the vertex sets A_1 and B_1 would become cells.

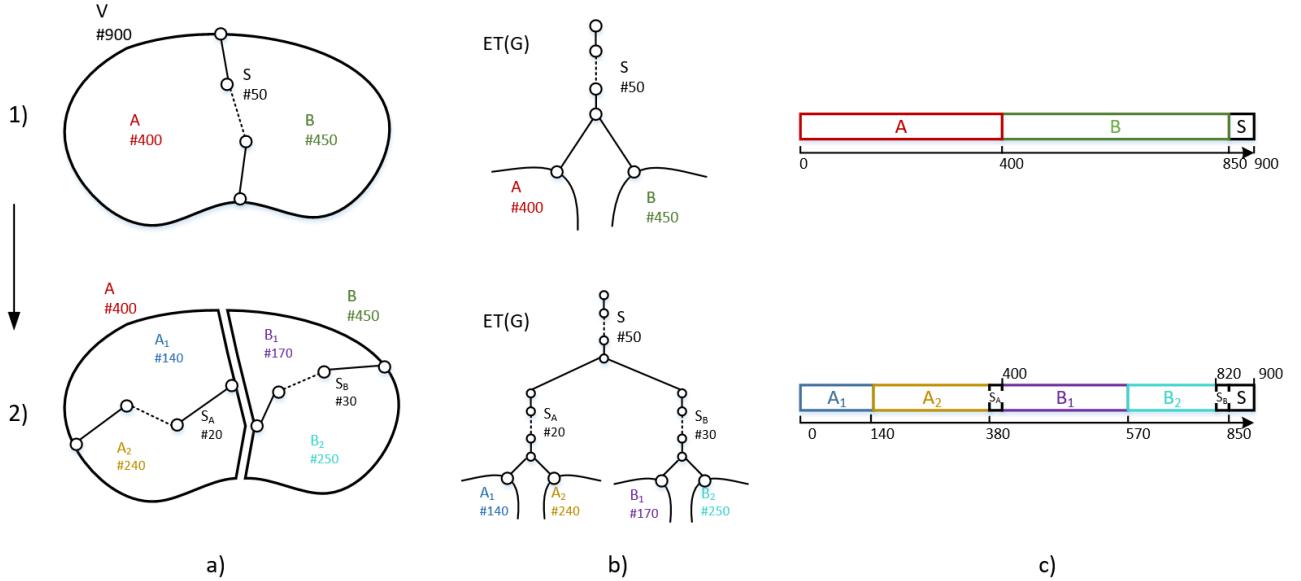


Figure 1: The first two steps of the iterative vertex ordering process. The graph cut and resulting subgraphs are displayed in a). The vertex ordering mapping is given in c), while b) illustrates the corresponding Elimination Tree.

Improved query algorithm based on CCH cells

In this section, we propose a technique to accelerate the downward phase of the query range algorithm thanks to the set of cells \mathcal{C} constructed beforehand. A cell $C \in \mathcal{C}$ generated thanks to a separator set S is composed of vertices of $V \setminus S$ with strictly smaller rank than those in S , i.e., $\{v \in V \setminus S \mid \pi^{-1}(v) < \min_{s \in S} \pi^{-1}(s)\} \subset C$. Moreover, all the vertices $(u, v) \in E^V$ starting in C also ends in C , i.e., $\forall (u, v) \in E^V$ if $u \in C$, then $v \in C$. As a consequence, a path computed in G^V will never leave a cell once inside. This property enables to define a bound that can be used to flag the cell as time-reachable at once. A possible implementation in pseudo-code is given in Algorithm 3.

A cell $C \in \mathcal{C}$ in the downward graph G^V is *time-reachable* from $s \in V$ in G for budget $b \in \mathbb{R}^+$ iff for all vertices $v \in C$ we have $\mathcal{S}_t^G(s, v) \leq b$. It is possible to mark a cell C as time-reachable only by using the diameter of C computed from a vertex in its frontier set. The frontier set F_C of C is composed of all the vertices outside C that are connected to C in G^V , formally $F_C = \{f \in V \setminus C \mid \exists c \in C, (f, c) \in E^V\}$. The diameter of C from a frontier vertex $f \in F_C$ is denoted by $\phi_t(C, f)$ and represents the maximal time cost to reach any vertex of C thanks to a time shortest path from f in G^V , i.e. $\phi_t(C, f) = \max_{v \in C} \mathcal{S}_{t+}^G(f, v)$, where $\mathcal{S}_{t+}^G(f, v)$ is the cost of a time shortest $f - v$ path in G^V . When a frontier vertex $f \in F_C$ is removed from the priority queue during the downward phase, the time value computed by the algorithm for f is exactly the cost $\mathcal{S}_t^G(s, f)$ of a time shortest $s - f$ path in G . It is easy to show that any time shortest path leading to a vertex of the cell is bounded by $\mathcal{S}_t^G(s, f) + \phi_t(C, f)$. Then, at this step, if $\mathcal{S}_t^G(s, f) + \phi_t(C, f) \leq b$, we know all the vertices of the cell C can be added to $\text{Range}(G, s, t, b)$ without exploring them. Let us formalize this with the

following property.

Property 1. Given a graph $G = (V, E)$, a time function $t: E \rightarrow \mathbb{R}^+$, a cell $C \in \mathcal{C}$ in the derived downward graph G^\vee , an initial vertex $s \in V \setminus C$ and a budget $b \in \mathbb{R}^+$. If there exists a frontier vertex $f \in F_C$ such that $\mathcal{S}_t^G(s, f) + \phi_t(C, f) \leq b$, then C is time-reachable from s in G for b .

Proof. Let us show that for all vertex $v \in C$, there is an $s - v$ path ρ in G such that $\text{Cost}_t(\rho) \leq b$. Then, because $\mathcal{S}_t^G(s, v) \leq \text{Cost}_t(\rho)$, it follows that $\mathcal{S}_t^G(s, v) \leq b$.

Let $v \in C$. Consider the path $\rho = \rho_{sf}\rho_{fv}$ with ρ_{sf} a time shortest $s - f$ path in G and ρ_{fv} the path in G corresponding to a time shortest $f - v$ path ρ'_{fv} in G^\vee . This path ρ'_{fv} exists in G^\vee because $f \in F_C, v \in C$, which means that $\pi^{-1}(f) > \pi^{-1}(v)$. By construction, $\text{Cost}_{t^+}(\rho'_{fv}) = \text{Cost}_t(\rho_{fv})$ and it follows that $\text{Cost}_t(\rho) = \text{Cost}_t(\rho_{sf}) + \text{Cost}_t(\rho_{fv}) = \mathcal{S}_t^G(s, f) + \mathcal{S}_{t^+}^{G^\vee}(f, v) \leq \mathcal{S}_t^G(s, f) + \phi_t(C, f) \leq b$. \square

Algorithm 3: QueryRangeDownWithCells(G^* , b , Q , X)

```

 $S \leftarrow \emptyset$  // Reachable set.
while  $Q \neq \emptyset$  do
     $(u, m) \leftarrow Q.\min()$  // Vertex  $u$  and metric cost  $m$ .
     $S \leftarrow S \cup \{u\}$  // All the dequeued nodes are reachable.

    if  $\exists C \in \mathcal{C}, u \in C$  do // Try to validate the cell that may cover  $u$ .
         $(u^{up}, m^{up}) = X[\text{previous}(u)]$  // Get node before on the shortest path to vertex  $u$ .
        if  $u^{up} \in F_C$  and  $m^{up} + \phi_t(C, u^{up}) \leq b$  and  $C \notin S$  do
             $S \leftarrow S \cup C$  // Practically, only the cell index needs to be saved.
            continue // Do not relax this vertex.

        foreach  $(u, w) \in E^\vee$  do // Adjacent vertices in the down graph.
            [...]
    return  $S$  // Same as in Algorithm 2.

```

Range computation with binary scalar metrics

In the previous section, we discussed techniques for computing the set of vertices $v \in V$ reachable from $s \in V$ within a time limit, focusing on time shortest $s - v$ paths. However, assessing the *feasibility* of a time-shortest path using *another* metric is often necessary. Without loss of generality, let us consider the distance function $d: E \rightarrow \mathbb{R}^+$ that assigns a distance value $d(e) \in \mathbb{R}^+$ to each edge $e \in E$. Notably, distance shortest paths may differ significantly from time shortest paths. Therefore, we use two metrics when optimizing paths for one metric (e.g., time) while adhering to a budget in another (e.g., distance).

As for the time metric, the cost of a path $\rho = \rho_0 \dots \rho_k \in V^+$ in G w.r.t. d is defined as the sum of the distance values of the edges forming ρ , i.e., $\text{Cost}_d(\rho) = \sum_{i=0}^{k-1} d(\rho_i, \rho_{i+1})$. Some adaptations need to be brought to the previous section to consider a second metric. Indeed, a time shortest path is not necessarily unique in a graph, and different time shortest paths can have different distance cost values. Let us denote by $\mathcal{S}_{t,d}^G(s, v)$ the minimum *distance* cost of all the *time* shortest $s - v$ paths in G . In this section, we are interested in computing the set of vertices so that the minimum distance cost of all the time shortest paths is below a predefined budget value. Then, we define the notion of distance reachable vertices as follows. Given a directed graph $G = (V, E)$, a time function $t: E \rightarrow \mathbb{R}^+$, a distance function $d: E \rightarrow \mathbb{R}^+$ and a distance budget $b \in \mathbb{R}^+$, a vertex $t \in V$ is *distance-reachable from s in G for b* iff $\mathcal{S}_{t,d}^G(s, t) \leq b$. Let us recall we are working in a contracted graph $G^* =$

(V, E^*) , as for the time metric, a new function $d^+: E \cup E^+ \rightarrow \mathbb{R}^+$ is defined on G^* such that the cost of each edge $(u, w) \in E^+$ is given by $\mathcal{S}_{t,d}^G(s, v)$. The cost of a path $\rho = \rho_0 \dots \rho_k$ in G^* is given by $\text{Cost}_{d^+}(\rho) = \sum_{i=0}^{k-1} d^+(\rho_i, \rho_{i+1})$.

The updated query logic for the dual metric scenario is detailed in Algorithms 4, 5, and 6 as pseudo-code. A key distinction from the single metric case is how the priority queue is managed. Before, only vertices below the budget were queued, with an empty queue signaling the end. With two metrics, including only distance-reachable vertices in the queue could incorrectly compute the time shortest path, possibly admitting a vertex v via a non-time optimal path. Therefore, all relaxed nodes are now added to the queue, while their reachability is assessed after they have been dequeued. This process continues until the queue no longer contains any reachable node. Consequently, the query must run from start to finish in both the upward and downward graphs, using a single stopping criterion.

As in the previous section, we can use cells to improve the performance of the downward phase. We can easily show that any up-down path ρ from a source vertex $s \in V \setminus C$ leading to a target vertex v of cell C with $\text{Cost}_{d^+}(\rho) = \mathcal{S}_{t,d}^G(s, v)$ includes a frontier node $f \in F_C$ and has a cost given by $\mathcal{S}_{t,d}^G(s, f) + \mathcal{S}_{t^+, d^+}^{G^V}(f, v)$, where $\mathcal{S}_{t^+, d^+}^{G^V}(f, v)$ is the minimum distance cost of the time shortest $f - v$ paths in G^V . To compute the diameter of a cell $C \in \mathcal{C}$ from $f \in F_C$, that we denote by $\phi_{t,d}(C, f)$, we compute the best $f - v$ path for all $v \in C$ and we take the maximum of such values, i.e. $\phi_{t,d}(C, f) = \max_{v \in C} \mathcal{S}_{t^+, d^+}^{G^V}(f, v)$.

However, a cell cannot be tested immediately as soon as any of its frontier vertex is hit, as we did previously when we had only one metric at stake. Indeed, there is no guarantee that a shortest $s - t$ path optimized on time will be the better one in terms of distance. When we hit some frontier vertex $f \in F_C$ with diameter $\phi_{t,d}(C, f)$, we may well have some vertex v in C for which f does not belong to any time shortest $s - v$ path. It may well go through an alternative frontier vertex f' . This means that even if the criterion at f holds, i.e. $\mathcal{S}_{t,d}^G(s, f) + \phi_{t,d}(C, f) \leq b$, there is no guarantee that $\mathcal{S}_{t,d}^G(s, f') + \mathcal{S}_{t^+, d^+}^{G^V}(f', v) \leq b$ is correct.

What we can do during the downward phase instead is to check that the criterion based on the diameter is met for all the frontier vertices of the encountered cells. In that case, the cell C is achievable within the budget and can be accepted. This is formalized in the following property.

Property 2. Given a graph $G = (V, E)$, a time function $t: E \rightarrow \mathbb{R}^+$, a distance function $d: E \rightarrow \mathbb{R}^+$, a cell $C \in \mathcal{C}$ in the derived downward graph G^V , an initial vertex $s \in V \setminus C$, and a distance budget $b \in \mathbb{R}^+$. If for all frontier vertices $f \in F_C$, $\mathcal{S}_{t,d}^G(s, f) + \phi_{t,d}(C, f) \leq b$, then C is distance-reachable from s in G for b .

Proof. Suppose that for all frontier vertex $f \in F_C$, $\mathcal{S}_{t,d}^G(s, f) + \phi_{t,d}(C, f) \leq b$. Let us show that for all vertex $v \in C$, $\mathcal{S}_{t,d}^G(s, v) \leq b$.

Let $\rho = \rho_0 \dots \rho_k$ be an up-down time shortest $s - v$ path in G^* such that $\text{Cost}_{d^+}(\rho) = \mathcal{S}_{t,d}^G(s, v)$. As $s \in V \setminus C$ and $v \in C$, $\exists i \in \{0, \dots, k-1\}, \rho_i \in F_C$ and $\rho_j \in C, \forall j > i$. Clearly $\rho_i \dots \rho_k$ is a suffix of the downward path composing ρ . If we split ρ in $\rho_0 \dots \rho_i$ and $\rho_i \dots \rho_k$, it follows that $\text{Cost}_{d^+}(\rho) = \mathcal{S}_{t,d}^G(s, \rho_i) + \mathcal{S}_{t^+, d^+}^{G^V}(\rho_i, v) \leq \mathcal{S}_{t,d}^G(s, \rho_i) + \phi_{t,d}(C, \rho_i) \leq b$. Then $\mathcal{S}_{t,d}^G(s, v) \leq b$. \square

From the previous proof, it should be noted that Property 2 can be restricted to frontier vertices $f \in F_C$ such

that there is a vertex $v \in C$ with a $f - v$ path $\rho_{fv} = \rho_0 \dots \rho_k$ in G^\vee with $\text{Cost}_{t,d}(\rho_{fv}) = \mathcal{S}_{t,d}^{\vee}(f, v) = \mathcal{S}_{t,d}^G(f, v)$ and $\forall i \in \{1, \dots, k\}, \rho_i \in C$. The frontier vertex f is said to be the witness of such vertices v . To increase the chance to accept cells at once during range computation, we can also refine the diameter computed for those frontier vertices by only considering the restricted set of vertices $C_f = \{v \in C \mid f \text{ is a witness of } v\}$. The refined value is then given by the diameter function applied on C_f , i.e., $\phi_{t,d}(C_f, f)$.

Property 3. Given a graph $G = (V, E)$, a time function $t: E \rightarrow \mathbb{R}^+$, a distance function $d: E \rightarrow \mathbb{R}^+$, a cell $C \in \mathcal{C}$ in the derived downward graph G^\vee , an initial vertex $s \in V \setminus C$, and a distance budget $b \in \mathbb{R}^+$. If for all frontier vertices $f \in F_C$ with $C_f \neq \emptyset$, $\mathcal{S}_{t,d}^G(s, f) + \phi_{t,d}(C_f, f) \leq b$, then C is distance-reachable from s in G , given budget b .

Algorithm 4: QueryRangeBinary (G^*, s, b)

```

 $Q \leftarrow \{(s, 0, 0)\}$  // Priority queue keyed by the cost of two metrics.
 $X \leftarrow \{(s, 0, 0)\}$  // Exploration set keyed by node.
 $S \leftarrow \emptyset$  // Reachable set.
 $H \leftarrow \emptyset$  // Nodes that are part of some encountered cell, indexed by cell then by node.

while  $\text{NumVertexWithinBudgetIn}(Q) > 0$  do // We stop when there is no more node within the budget.
     $\text{QueryRangeStep}(G^*, b, Q, X, S, H, \text{expand} = \text{True})$ 
while  $Q \neq \emptyset$  do // We flush the remaining nodes while avoiding expansion.
     $\text{QueryRangeStep}(G^*, b, Q, X, S, H, \text{expand} = \text{False})$ 

foreach  $C \in \mathcal{C}, H[C] \neq \emptyset$  do
    if  $\forall f \in F_C, C_f \neq \emptyset$  and  $H[C][f].m_2 + \phi_{t,d}(C_f, f) \leq b$  do
         $S \leftarrow S \cup C$  // Add all the cell nodes if each relevant frontier node is validated.
    else do
         $S \leftarrow S \cup \text{ScanCellDown}(G^\vee, b, C, H[C])$ 
return  $S$ 

```

Algorithm 5: QueryRangeStep ($G, b, Q, X, S, H, \text{expand}$)

```

 $(u, m_1, m_2) \leftarrow \min(Q)$  // Vertex  $u$ , optim. metric cost  $m_1$  and budget metric cost  $m_2$ .
 $goUp \leftarrow \text{previous}(u) < u$  // Determine if we are going up or down for this node.
 $inCell \leftarrow \text{True if } \exists C \in \mathcal{C}, u \in C \text{ else False}$ 
 $withinBudget \leftarrow m_2 \leq b$  //  $Q$  may contain vertices above budget.
if  $withinBudget$  and  $\neg inCell$  do  $S \leftarrow S \cup \{u\}$  //  $m_1$  and  $m_2$  are final, so we can make the test here
// if  $u$  is within a cell, it is not final !
 $\text{RelaxationDown}(G^\vee, b, X, Q, H, u, \text{expand})$  // Relax while taking care of having an up-down path.
if  $goUp$  do  $\text{RelaxationUp}(G^\wedge, b, X, Q, u, \text{expand})$ 

if  $goUp$  and  $inCell$  do // This happens when the range starts within a cell.
     $H[C] \leftarrow H[C] \cup \{(u, m_1, m_2)\}$  // Save to test the cells later.

```

Algorithm 6: RelaxationDown ($G^\vee, b, X, Q, H, u, \text{expand}$)

```

foreach  $(u, w) \in E^\vee$  do // Adjacent vertices in the down graph.
     $m_1' \leftarrow X[u].m_1 + t^+(u, w)$ 
     $m_2' \leftarrow X[u].m_2 + d^+(u, w)$ 
     $cellAncestor \leftarrow \text{True if } H[C] \neq \emptyset, \exists f \in F_C \mid \exists w - f \text{ path in } G^\vee \text{ else False}$ 
     $expand' \leftarrow expand \text{ or } cellAncestor$ 

    if  $\exists C \in \mathcal{C}, w \in C$  do // Save information on node that are part of a cell.
         $H[C] \leftarrow H[C] \cup \{(u, X[u].m_1, X[u].m_2)\}$ 
         $H[C] \leftarrow H[C] \cup \{(w, m_1', m_2')\}$ 
        continue // Do not relax nodes inside a cell.

```

```

if  $w \in X$  do
     $better \leftarrow (X[w].m_1 > m'_1) \text{ or } ((X[w].m_1 == m'_1) \text{ and } (X[w].m_2 > m'_2))$ 
    if  $better$  do
         $Q.update(w, m'_1, m'_2)$ 
         $X[w].m_1 \leftarrow m'_1$ 
         $X[w].m_2 \leftarrow m'_2$ 
    else if  $w \notin X$  and  $expand'$  do
         $Q.insert(w, m'_1, m'_2)$ 
         $X \leftarrow X \cup \{(w, m'_1, m'_2)\}$ 

```

Algorithm 7: RelaxationUp ($G^\wedge, b, X, Q, u, expand$)

Similar to algorithm 6, expect that cells are not tested.

Algorithm 8: ScanCellDown (G^\wedge, b, C, H)

This function relaxes all the nodes in cell C , by descending the cell elimination tree without using a priority queue. The node initial costs are those that can be found in $H[C]$, or infinity otherwise. The relaxation is otherwise similar to algorithm 6.

Range computation with EV metrics

An important aspect of range calculation relates to electric vehicles (EVs). Given the limited capacity of batteries, it is crucial for EV drivers to be aware of the geographical area they can reach with their current battery State of Charge (SoC), all while optimizing their travel time. So, given an initial vertex $s \in V$, we are interested in computing the set of vertices $v \in V$ such that there exists a time shortest $s - v$ path that is *energy feasible*. Such path is feasible if and only if the battery level never drops below a predefined level.

The EV case requires a metric for energy consumption along graph edges, that can be formalized by a function of energy consumption $c: E \rightarrow \mathbb{R}$ with potentially *negative* values for regeneration in down slopes. In addition, another complexity comes from the need to respect the physical constraints of the battery, i.e., the energy level must remain positive and bounded by the battery capacity M . Given a current battery SoC b_u at vertex $u \in V$ while traversing the edge $(u, v) \in E$, the resulting unbounded battery SoC would be given by $b_v = b_u - c(u, v)$. But because b_v must remain in $[0, M]$, edge costs must be modelled thanks to piecewise linear functions $\mathbf{f}_e: [0, M] \rightarrow [0, M] \cup \{\infty\}$ that depends on the current battery level in the general case [4, 6]. However, applying this theory to our cell-based acceleration method would necessitate significant additional development that surpasses our capacity to deal with it within the scope of this paper. Such an extension would also lead to increased algorithmic complexity and higher memory requirements for the final application. Therefore, we defer this development to future work and, for the purposes of this paper, assume that every path is energy-feasible, provided the overall energy balance remains within the specified budget end to end. It is important to note that this simplification holds true as long as the battery energy never temporarily drops below zero during the journey. For instance, this scenario might occur when a path involves a long uphill section followed by a prolonged downhill descent.

Even in the absence of negative cycles, the Dijkstra algorithm cannot be used efficiently on graphs with negative costs because it would lose its *label-setting* property. In other words, the cost of a vertex removed from the priority queue would not be guaranteed to be the cost of its shortest path. To address this problem, we employ a *potential shifting* technique [13] that eliminates all the negative costs. We can define an alternative cost function $c_p: E \rightarrow \mathbb{R}^+$ by incorporating the difference in *potential energy* across the edge $(u, v) \in E$. The

reduced cost c_p uses a potential function $p : V \rightarrow \mathbb{R}$ on each vertex of G such that $\forall (u, v) \in E$, $c_p(u, v) = c(u, v) + p(u) - p(v) > 0$. This new cost function ensures that all costs on the edges are strictly positive. Indeed, it is not physically possible for a vehicle to recover more energy than the difference in potential energy along an edge. As such, we can restrict ourselves to this class of well-formed energy metrics w.l.o.g.

The cost of a path $\rho = \rho_0 \dots \rho_k$ in G w.r.t. c is defined as the sum of the energy consumption values of the edges forming ρ , i.e., $Cost_c(\rho) = \sum_{i=0}^{k-1} c(\rho_i, \rho_{i+1})$. When applying the potential shifting technique, the cost of ρ becomes $Cost_{c_p}(\rho) = Cost_c(\rho) + p(\rho_0) - p(\rho_k)$. To compute a range, we are interested in the set of vertices whose minimum consumption value on all their time-shortest paths is below a predefined budget. Formally, given a directed graph $G = (V, E)$, a time function $t : E \rightarrow \mathbb{R}^+$, a function of energy consumption $c : E \rightarrow \mathbb{R}$ and a consumption budget $b \in \mathbb{R}^+$, a vertex $v \in V$ is *energy-reachable* from s and for budget b iff $\mathcal{S}_{t,c}^G(s, v) \leq b$, where $\mathcal{S}_{t,c}^G(s, v)$ is the minimum *energy consumption* cost of all the *time* shortest $s - v$ paths in G . Checking $\mathcal{S}_{t,c}^G(s, v) \leq b$ is equivalent to test $\mathcal{S}_{t,c_p}^G(s, v) \leq b + p(s) - p(v)$, where $\mathcal{S}_{t,c_p}^G(s, v)$ is the minimum *shifted consumption* cost of all the *time* shortest $s - v$ paths in G .

Comparing the cost of different paths that begin and end at the same location is straightforward as the costs are independent of the potential of the internal vertices. However, when it comes to determining the diameter $\phi_{t,c_p}(C, f)$ of some cell $C \in \mathcal{C}$ and frontier vertex $f \in F_C$, this computation encompasses all the ending vertices within C . To make this calculation independent of their individual potential, we can define the diameter as follows: $\phi_{t,c_p}(C, f) = \max_{v \in C} (\mathcal{S}_{t^+, c_p^+}^{G^V}(f, v) + p(v))$, where $\mathcal{S}_{t^+, c_p^+}^{G^V}(f, v)$ is the minimum energy shifted cost of the time shortest $f - v$ paths in G^V . The new cost functions c^+, c_p^+ are defined on the set of edges E^* , analogously to d^+ in the previous section.

Property 4. Given a graph $G = (V, E)$, a time function $t : E \rightarrow \mathbb{R}^+$, a function of energy consumption $c_p : E \rightarrow \mathbb{R}^+$ shifted thanks to the potential function $p : V \rightarrow \mathbb{R}$, a cell $C \in \mathcal{C}$ in the derived downward graph G^V , an initial vertex $s \in V \setminus C$, and a distance budget $b \in \mathbb{R}^+$. If for all frontier vertices $f \in F_C$, $\mathcal{S}_{t,c_p}^G(s, f) - p(s) + \phi_{t,c_p}(C, f) \leq b$, then C is energy-reachable from s in G for b .

Proof. Let us show that for all vertex $v \in C$, $\mathcal{S}_{t,c}^G(s, v) \leq b$. Let $\rho = \rho_0 \dots \rho_k$ be an up-down $s-v$ path in G^* such that $Cost_{c^+}(\rho) = \mathcal{S}_{t,c}^G(s, v)$. As $s \in V \setminus C$ and $v \in C$, $\exists i \in \{0, \dots, k-1\}$, $\rho_i \in F_C$ and $\rho_j \in C, \forall j > i$. Therefore, $\rho_i \dots \rho_k$ is a suffix of the downward path composing ρ and we have:

$$\begin{aligned} Cost_{c^+}(\rho) &= Cost_{c^+}(\rho_0 \dots \rho_i) + Cost_{c^+}(\rho_i \dots \rho_k) \\ &= Cost_{c_p^+}(\rho_0 \dots \rho_i) - p(\rho_0) + p(\rho_i) + Cost_{c_p^+}(\rho_i \dots \rho_k) - p(\rho_i) + p(\rho_k) \\ &= \mathcal{S}_{t,c_p}^G(s, \rho_i) - p(s) + \mathcal{S}_{t^+, c_p^+}^{G^V}(\rho_i, v) + p(v) \\ &\leq \mathcal{S}_{t,c_p}^G(s, \rho_i) - p(s) + \phi_{t,c_p}(C, \rho_i) \leq b. \end{aligned}$$

It then follows that $\mathcal{S}_{t,c}^G(s, v) \leq b$

□

Experiments

Table 1 provides an overview of the range queries performance for the *single* and *binary* metric cases, with and without the cell-based acceleration method. The measurements have been conducted *single thread* on an AMD Ryzen 9 5900X 12-Core Processor@3.70 GHz, for different range polygons of size N , in terms of vertices they contain. The underlying map is the one of *North America*, made freely available for the 9th DIMACS Implementation Challenge [14]. It contains around 24 M *vertices* and 29 M *edges*, rising to 91 M edges after the contraction process. This map has been augmented with elevation data coming from the ETOPO1 global elevation dataset developed by NOAA [15]. The selected model has a resolution of 60 arc-second and has been read thanks to the tools available on the Open Topo Data web site [16]. The *energy* metric has been generated in conjunction with a private EV consumption model running on the slopes and the driving speed of the road network. The average consumption of this vehicle is around 17kWh/100km.

The graph data required to run the router consume around 4.2 GB on disk, of which 2.1 GB is attributed to the three metrics *time*, *distance*, and *energy*. The contraction phase took 48 min, parallelized on 11 cores, while each metric customization took 30 sec. The size threshold used to define the cell needed for the acceleration has been empirically set to 4096, to get good performances across the different query types.

Requests are generated by choosing random center nodes across the map, then by adjusting the budget to ensure that the range polygon contains the required number of vertices $N \pm 10\%$. Each query in the table is run x times such that $x \cdot N > 10^8$. This means that a range query targeting a polygon containing 1 million vertices was run at 100 different locations and its corresponding measurements averaged. This allows for a good balance between sufficiently large test sizes and acceptable running times.

Polygon size	Single metric		Binary metric	
	time		time+distance	time+energy
$N=10^4$	~budget	59 min	61 km	8 kWh
	with / w.o. acc. [ms]	1 / 6	32 / 17	42 / 32
$N=10^5$	~budget	2h 10 min	174 km	28 kWh
	with / w.o. acc. [ms]	8 / 69	84 / 203	109 / 365
$N=10^6$	~budget	5h 35 min	528 km	99 kWh
	with / w.o. acc. [ms]	59 / 1186	241 / 2252	336 / 3550
$N=5.10^6$	~budget	9h 51 min	1165 km	229 kWh
	with / w.o. acc. [ms]	286 / 7202	689 / 12542	932 / 14868

Table 1: Average processing time of range queries for different metrics and for various sizes of polygon, in ms, along with the corresponding average budget required to reach such sizes. Each figure is given with and without the cell-based acceleration technique.

Conclusion

In this paper, we have explored in detail how to apply an acceleration technique based on cells to compute reachable ranges in the context of a strict Contraction Hierarchy router. The technique is based on the concept that whole parts of the map, i.e., cells, can be validated at once when they are hit, provided some criteria are met on the nodes at their frontier. As a result, most of the nodes in a range fall either in a validated cell that can be skipped in a snap, or in some independent part of the map that requires further investigation. However, those invalidated cells can be explored more efficiently by taking advantage of the CH structure that allows processing

them without the usage of a priority queue.

The experiments show an acceleration factor of ~ 25 on the biggest ranges with a *single metric*, compared to the naïve Dijkstra implementation. This acceleration decreases for smaller ranges, but this is expected as those later are not big enough to cover many cells entirely. Regarding the *binary metrics*, we have seen in this paper that this case is trickier to manage, and the experiments show up longer computation times. This is because the exploration must process more nodes to ensure some better shortest paths have not been missed, along with the stronger conditions that must be met to validate a cell. Still, the acceleration factor is around ~ 17 for the biggest range, allowing their computation in less than 1 second, even for those with the longest range.

If we now compare the two binary metrics, we can see that the one related to *energy* is a penalized and requires $\sim 30\%$ supplementary time to compute compared to the one related to *distance*. We think this effect comes from the potential shifting technique, as each node validation necessitates an additional step to get its altitude and to translate it in potential energy.

References

- [1] E. Dijkstra, «A note on two problems in connexion with graphs.,» *Numerische Mathematik*, vol. 1, pp. 269-271, 1959.
- [2] Madkour, Amgad, W. G. Aref, F. U. Rehman, M. A. Rahman et S. Basalamah, «A survey of shortest-path algorithms.,» arXiv preprint arXiv:1705.02044, 2017.
- [3] J.-S. Gonsette et N. Meunier, «Fast Matrix Queries and Application to Routing Optimization Problems.,» *Proceedings of the 29th ITS WORLD CONGRESS Suzhou* , 2023.
- [4] J. Eisner, S. Funke et S. Storandt, «Optimal route planning for electric vehicles in large networks.,» *Proceedings of the aaai conference on artificial intelligence.*, 2011.
- [5] M. Baum, T. Pajor, J. Dibbelt et D. Wagner, «Energy-optimal routes for electric vehicles.,» *Proceedings of the 21st ACM SIGSPATIAL international conference on advances in geographic information systems.* , 2013.
- [6] S. Storandt, «Algorithms for vehicle navigation.,» PhD thesis, 2012.
- [7] V. Bauer, J. Gamper, R. Loperfido, S. Profanter, S. Putzer and I. Timko, "Computing isochrones in multi-modal, schedule-based transport networks," in *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems.* , 2008.
- [8] M. Baum, V. Buchhold, J. Dibbelt et D. Wagner, « Fast Computation of Isochrones in Road Networks.,» *preprint arXiv:1512.09090.* , 2015.
- [9] V. Buchhold, «Fast computation of Isochrones in road networks.,» Diss. Master Thesis, 2015.
- [10] R. Geisberger, P. Sanders, D. Schultes et D. Delling, «Contraction Hierarchies: Faster and simpler hierarchical routing in road networks.,» *Experimental Algorithms: 7th International Workshop*, 2008.
- [11] J. Dibbelt, B. Strasser et D. Wagner, «Customizable Contraction Hierarchies.,» *Journal of Experimental Algorithms*, 2016.
- [12] L. Gottesbüren, M. Hamann, T. N. Uhl et D. Wagner, «Faster and Better Nested Dissection Orders for Customizable Contraction Hierarchies.,» *Algorithms*, 2019.
- [13] D. B. Johnson, «Efficient algorithms for shortest paths in sparse networks.,» *Journal of the ACM (JACM)*, 1977.
- [14] C. Demetrescu, A. V. Goldberg et D. S. Johnson, «The Shortest Path Problem: Ninth DIMACS Implementation Challenge.,» *DIMACS Book: American Mathematical Society*, vol. 74, 2009.
- [15] NCEI, «<https://www.ncei.noaa.gov/products/etopo-global-relief-model>,» National Oceanic and Atmospheric Administration (NOAA), 2022. [En ligne].
- [16] A. Nisbet, «Open Topo Data.,» [En ligne]. Available: <https://www.opentopodata.org/>.