Graph Indexing of Road Networks for Shortest Path Queries with Label Restrictions

Michael Rice
University of California, Riverside
Riverside, CA 92521
mrice@cs.ucr.edu

Vassilis J. Tsotras
University of California, Riverside
Riverside, CA 92521
tsotras@cs.ucr.edu

ABSTRACT

The current widespread use of location-based services and GPS technologies has revived interest in very fast and scalable shortest path queries. We introduce a new shortest path query type in which dynamic constraints may be placed on the allowable set of edges that can appear on a valid shortest path (e.g., dynamically restricting the type of roads or modes of travel which may be considered in a multimodal transportation network). We formalize this problem as a specific variant of formal language constrained shortest path problems, which we call the *Kleene Language Constrained Shortest Paths* problem. To efficiently support this type of dynamically constrained shortest path query for large-scale datasets, we extend the hierarchical graph indexing technique known as Contraction Hierarchies. Our experimental evaluation using the North American road network dataset (with over 50 million edges) shows an average query speed and search space improvement of over 3 orders of magnitude compared to the naïve adaptation of the standard Dijkstra's algorithm to support this query type. We also show an improvement of over 2 orders of magnitude compared to the only previously-existing indexing technique which could solve this problem without additional preprocessing.

1. INTRODUCTION

Due to its ubiquitous usage over the web and in many commercial navigation products, point-to-point shortest path search on graphs has again become a major topic of interest over the last decade, with much research being devoted to designing practical indexing techniques for extremely fast graph searches. Graph indexing techniques have been widely explored for establishing efficient data structures for pruning and/or directing the search of shortest path algorithms, while still guaranteeing the optimality of the resulting paths. Such techniques have resulted in many improvements over the standard Dijkstra's algorithm [6], and may also be used to minimize the overall I/O costs incurred by the graph search for very large, external-memory graph datasets [10, 15]. However, focus thus far has been mostly on static shortest paths with no constraints.

Proceedings of the VLDB Endowment, Vol. 4, No. 2

In this research, we focus on a variant of shortest path queries in which dynamic constraints may be placed upon the type of edges which may appear on a valid shortest path. For example, the shortest path from Irvine, CA to Riverside, CA travels along State Route 261, which is a local toll road through this area. However, consider the case where the traveler does not wish to pay the toll fee, and would therefore rather find the shortest path from Irvine to Riverside that actually avoids all toll roads. As yet another example, trucks delivering certain hazardous materials may not be allowed to cross over some types of roadways, such as bridges or railroad crossings, due to the public health and safety risks of any potential accidents. Therefore, this query type can be seen to have practical applications in both personalized location-based services, as well as in many logistics and commercial transportation scenarios. Making this query highly efficient on real-world, large-scale graphs, such as the road network of the continental United States, is therefore crucial to effectively supporting such practical applications.

1.1 Related Work

In recent years, hierarchical graph indexing techniques have been shown to be some of the most time- and space-efficient approaches towards indexing graphs for shortest path computations [8, 15, 16, 4, 13, 14, 12]. Hierarchical techniques generally involve some classification of the vertices/edges within the graph into mutually-exclusive, ordered levels of hierarchy, based on some notion of importance within the graph structure. Shortest path queries carried out on a hierarchical graph index typically prefer searching towards higher (i.e., more important) levels of the graph hierarchy, while progressively ignoring lower (i.e., less important) levels of the hierarchy, in order to more effectively reduce the overall search space explored by the query.

Schultes and Sanders [16] have previously explored a variant of their hierarchical indexing techniques designed to support dynamic changes in graph edge weights or cost functions. However, support for this dynamic approach requires either explicit recomputation of the graph index online as the weights (or cost functions) change or the query algorithm must make increasingly limited use of the information available in the static graph index based on the dynamic changes.

Yet another practical graph indexing approach is the goal-directed approach of the ALT algorithm [9, 10]. The ALT algorithm is based primarily on the concepts of A* search [11], in which the search from a source node is "directed"

towards the target node by the use of a potential function to estimate the shortest path cost to the target. The ALT algorithm allows preprocessing in which a set of so-called *landmark* nodes is selected from the graph and the shortest path is computed for each landmark node to/from all other nodes in the graph. Using properties of the triangle inequality derived from the costs to/from all landmark nodes, a highly efficient potential function can be constructed, thus greatly reducing the resulting search space. This technique has been further studied within the context of dynamic graphs in [5], and it can be shown that the potential functions from the original landmark preprocessing remain correct for all shortest paths as long as the edge weights can only *increase* in a dynamic scenario.

In the context of our own constrained shortest path query presented here, the idea of dynamically restricting an edge from being allowed in the search for a particular query can be seen as equivalent to simply increasing the weight of that edge to infinity for the lifetime of the query. Thus, the ALT technique is the only existing indexing technique directly applicable to our query type without requiring additional or specialized preprocessing.

1.2 Our Contributions

To the best of our knowledge, this is the first work to address this practical variant of shortest path query. In particular, our contributions can be summarized as follows. We formalize this problem as a restricted class of language constrained shortest paths, thus tying it to the existing literature and giving this new problem some relative context.

To efficiently support this type of dynamically constrained shortest path query, we detail a practical and efficient approach to extend the hierarchical graph indexing technique known as Contraction Hierarchies [8, 7]. Given implicit knowledge of the range of possible constraints for shortest path queries on a graph, we propose to incorporate this knowledge directly into the graph index construction to avoid the overhead of reconstructing the index for each possible constraint scenario at query time.

Using one of the largest commercial real-world road network datasets, we present experimental results with improvements of over 3 orders of magnitude compared to the naïve adaptation of the standard Dijkstra's algorithm[1] to support this query type. We also show an improvement of over 2 orders of magnitude compared to the dynamic ALT algorithm examined in [5].

The remainder of the paper is organized as follows. In Section 2, we present the concept of constraints on the allowable edges for a given shortest path query as a specific variant of language constrained shortest paths. Section 3 presents an overview of Contraction Hierarchies. Section 4 extends this technique with the proposed algorithms for constructing and querying the hierarchical graph index to support these constraints for shortest path queries. Section 5 presents our experimental analysis of this technique. Finally, Section 6 concludes the paper with future research.

2. LANGUAGE CONSTRAINED SHORTEST PATHS

Language constrained shortest paths [3] are shortest paths whose edge labels must satisfy some formal language constraint over a fixed alphabet Σ. We define this concept more formally as follows. Let $G = (V, E, w, \Sigma, \ell)$ be a directed graph, where V is the set of vertices in G, E is the set of edges in G, $w : E \rightarrow \mathbb{R}^+$ is a function mapping edges in G to a positive, real-valued weight, Σ is a finite alphabet used for labeling of edges in G, and $\ell : E \rightarrow \Sigma$ is a function mapping edges in G to a label in Σ.

Let $P_{s,t} = \langle e_1, e_2, \cdots, e_k \rangle$ be any path in G from some vertex $s \in V$ to some vertex $t \in V$, such that $e_1 = (s, v_1) \in E$, $e_k = (v_{k-1}, t) \in E$, and for $1 < i < k$, $e_i = (v_{i-1}, v_i) \in E$. Let $w(P_{s,t}) = \sum_{1 \leq i \leq k} w(e_i)$ be the total weight of all edges in $P_{s,t}$. Let $\ell(P_{s,t}) = \ell(e_1)\ell(e_2)\cdots\ell(e_k)$ be the concatenation of the labels of all edges in $P_{s,t}$. Given any formal language $L \subseteq \Sigma^*$, a *language constrained shortest path* is a path $P'_{s,t}$ in G such that $\ell(P'_{s,t}) \in L$ and $\forall\ P_{s,t}$ in G where $\ell(P_{s,t}) \in L$, $w(P'_{s,t}) \leq w(P_{s,t})$.

The *Regular Language Constrained Shortest Paths* (RLCSP) problem is a basic variant of language constrained shortest paths where the constraint language L must be a regular language. In [3, 1], Barrett et al. show that RLCSP is solvable in polynomial time by performing a shortest path search in the product graph of the original graph and the non-deterministic finite automaton (NFA) graph representing the specified regular language.

The *Linear Regular Expression* (LRE) constrained shortest paths problem [2] is a variation of RLCSP in which the regular expressions representing the constraint-language L must be of a specific form related to a restricted subclass of regular languages. In particular, linear regular expressions must be of the form $x_1^+ x_2^+ \cdots x_k^+$, where for $1 \leq i \leq k$, $x_i \in \Sigma$, and $x_i^+ = x_i x_i^*$.

LRE is presented primarily as a means of expressing modal constraints on real-world transportation networks, where a traveler knows the exact modes of travel (i.e., labels) they wish to consider and the exact order in which they wish to travel through these modes. One drawback to this approach is that such information may not always be known by the traveler in advance. For example, the traveler may not know the best order of modes to take in their trip; however, they are still likely to know exactly which modes they are ultimately *willing* to take (as well as those modes which they are *unwilling* to take). Therefore, we present a new variant of language constrained shortest paths (below) designed specifically to support this more flexible scenario.

2.1 Kleene Language Constrained Shortest Paths

We present the *Kleene Language Constrained Shortest Paths* (KLCSP) problem as a variant of language-constrained shortest paths based on another (simpler) subclass of regular languages which we shall call here the *Kleene languages*.

A *Kleene language* may be defined in this context as the Kleene closure of any subset of Σ. More formally, $\forall A \subseteq \Sigma$, $L(A^*)$ defines a Kleene language over alphabet A. Note that the subset alphabet A merely defines the set of *allowable* labels that can appear on a valid shortest path for a KLCSP problem. However, unlike LRE, the labels in A are not required to appear on a shortest path for a KLCSP problem and the sequence of the labels of such a path is irrelevant. Additionally, for any Kleene language over $A \subseteq \Sigma$, there is an implicitly defined subset of *restricted* labels $R = \Sigma \setminus A$, such that no labels in R may appear on any valid KLCSP solution. A Kleene language over $A \subseteq \Sigma$ may therefore

[1]We refer here to the more efficient bidirectional version.

be equivalently defined simply by specifying the set of such *restricted* labels, R, where $A = \Sigma \setminus R$. Given this definition, the KLCSP problem is designed to support the specification of language constraints on the allowed (restricted) set of labels which may (not) appear over a given shortest path, in any permutation. It is considered more common in practice to specify this constraint as the set of restricted labels, R, so we will adopt this approach for the remainder of this document.

For example, consider a transportation network consisting of labels $\Sigma = \{l, h, i, t, f\}$, which represent local roads, highways, interstates, toll roads, and ferries, respectively. A traveler may wish to find the shortest path between two locations in the network that avoids both toll roads and ferries. A Kleene language supporting this constraint could be defined as $L((\Sigma \setminus \{t, f\})^*)$.

The practical applications of KLCSP are also not restricted merely to modal constraints on a shortest path query. A label in Σ can correspond to any arbitrary predicate condition associated with the edges of the graph. In later sections dealing with the graph index construction, we must extend the notion of edge labels to include support for multiple labels per edge. This also proves highly useful in scenarios where a given edge can support multiple such predicate conditions simultaneously.

In order to support this, we redefine the function ℓ to support multiple labels per edge as follows: $\ell : E \rightarrow \mathcal{P}(\Sigma)$ is the labeling function mapping edges to a set of labels in Σ (where $\mathcal{P}(\Sigma)$ denotes the *power set* of Σ). Since this new function can now map a given edge to multiple potential labels, we must also redefine what it means for a path $P_{s,t}$ to be valid for a given Kleene language constraint. We say that an *R-restricted path* is any path $P_{s,t} = \langle e_1, e_2, \cdots, e_k \rangle$, such that, for $1 \leq i \leq k$, $\ell(e_i) \cap R = \emptyset$ (i.e., the path avoids all restricted labels in R). We denote the *shortest* R-restricted path from $s \in V$ to $t \in V$ as $P^R_{s,t}$.

Unlike the algorithms for RLCSP and LRE, which require a search through a product graph, this simple subclass of regular languages allows for a much more efficient optimization of the constrained shortest path search. In particular, we need now only verify that a given edge's labels do not belong to the restricted subset of labels, as indicated by R, before relaxing the edge in the search. We present the pseudocode for solving the KLCSP problem using a straightforward adaptation of Dijkstra's algorithm in Algorithm 1. Note that a similar bidirectional search can also be performed instead of the unidirectional search presented in this pseudocode. We present the unidirectional variant here merely for simplicity and greater ease of understanding.

Algorithm 1 KLCSP-Dijkstra(G, s, t, R)

Input: *Graph* $G = (V, E, w, \Sigma, \ell)$, $s, t \in V$, *restricted alphabet* $R \subseteq \Sigma$
Output: *Cost of shortest path* $P^R_{s,t}$

```
1: PQ ← ∅
2: for all v ∈ V do
3:    d[v] ← ∞
4: end for
5: d[s] ← 0
6: PQ.Insert(s, d[s])
7: while ¬PQ.Empty() do
8:    u ← PQ.ExtractMin()
9:    if u = t then
10:      return d[t]
11:   end if
12:   for all e = (u, v) ∈ E do
13:      if ℓ(e) ∩ R = ∅ ∧ d[u] + w(e) < d[v] then
14:         d[v] ← d[u] + w(e)
15:         if v ∉ PQ then
16:            PQ.Insert(v, d[v])
17:         else
18:            PQ.DecreaseKey(v, d[v])
19:         end if
20:      end if
21:   end for
22: end while
23: return ∞
```

3. CONTRACTION HIERARCHIES (CH)

CH [8, 7] have been proposed as an efficient graph indexing technique for supporting static point-to-point shortest path queries. The primary idea of CH is to establish some absolute ordering of the vertices in the graph (i.e., the ordering defines a bijective function $\phi : V \rightarrow \{1, ..., |V|\}$) with respect to some notion of general, relative importance. Given such an ordering, preprocessing proceeds by "contracting" one vertex at a time, in increasing order of importance. When a vertex, v, is contracted, it is removed from the current graph "in such a way that shortest paths in the remaining...[sub]graph are preserved" [8]. In particular, for any pair of remaining vertices, u and w, adjacent to v in the original graph whose only shortest u-w path is $\langle u, v, w \rangle$, a so-called *shortcut* edge (u, w) must be added with the weight of the original shortest path cost through v (see Figure 1 for an example). However, if there is an equivalent- or lesser-cost path from u to w other than $\langle u, v, w \rangle$, then no such shortcut edge is needed. Such a path is called a *witness* path. In order to detect witness paths, a local search from all nodes u, such that $(u, v) \in E$ and $\phi(u) > \phi(v)$, to all nodes w, such that $(v, w) \in E$ and $\phi(v) < \phi(w)$, is carried out to determine if a (u, w) shortcut edge is necessary.

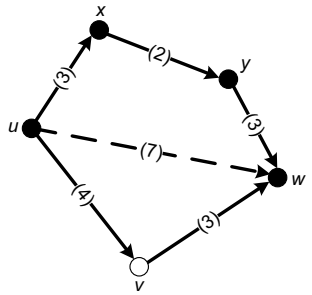


Figure 1: Contracting node v. Edges are labeled with their weights. The shortcut edge is represented with a dashed line.

Note that the number of shortcut edges added when contracting a graph is heavily dependent on the given ordering. Therefore, establishing a good ordering is one of the most crucial aspects of this methodology. In [8], Geisberger et al. establish several metrics to be associated with a given node that can help in determining the overall priority of that node in the ordering. In this context, vertex order-

ing is directly integrated into the contraction phase by first simulating the contraction of a given node to determine its resulting priority terms, and ordering the nodes in a priority queue based on a linear combination of these terms. Some of these metrics include: the difference between the number of shortcut edges added and the number of adjacent edges removed when contracting a node (*edge difference*), the number of neighbors of a node that have already been contracted (*contracted neighbors*), and the number of original edges represented by any new shortcuts added when contracting a node (*original edges*). The interested reader is referred to [8, 7] for a more exhaustive list and greater details on each priority term considered. At each iteration, the node with minimum priority value is removed from the priority queue, contracted, and the priority values of all of its neighboring vertices are updated for the next iteration.

Once the set of shortcut edges, E', has been established for a given ordering, shortest path queries may then be carried out using a bidirectional Dijkstra search variant which performs a simultaneous forward search in the *upward* graph $G_\uparrow = (V, E_\uparrow)$, where $E_\uparrow = \{(v, w) \in E \cup E' \mid \phi(v) < \phi(w)\}$, and backward search[2] in the *downward* graph $G_\downarrow = (V, E_\downarrow)$, where $E_\downarrow = \{(u, v) \in E \cup E' \mid \phi(u) > \phi(v)\}$. A tentative shortest path cost is maintained and is updated only when the two search frontiers meet to form a shorter path. The search in a given direction may be aborted once the minimum key for the priority queue in that direction exceeds the cost of the best tentative path seen so far. Once both search directions are finished, the best path seen thus far represents the shortest path cost. An illustration of this bidirectional search is given in Figure 2.

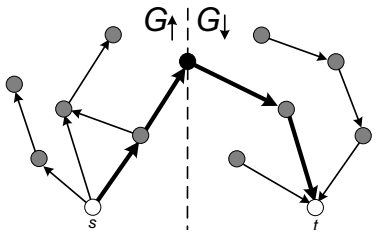


Figure 2: CH bidirectional search query. The resulting shortest path is indicated by the thick lines.

As with any graph search algorithm, the efficiency of the search process is directly proportional to the number of nodes and edges explored during the search. The effectiveness of the CH search technique therefore comes from the use of the newly-added shortcut edges, which allow the Dijkstra search to effectively *bypass* irrelevant nodes during the search, without invalidating correctness, thus resulting in a greatly-reduced search space (and therefore, better runtime), as compared to the standard Dijkstra search on the original graph.

4. CONTRACTION HIERARCHIES WITH LABEL RESTRICTIONS (CHLR)

Despite the naïve adaptation of Dijkstra's algorithm to support the Kleene language constrained shortest paths, as presented in Algorithm 1, this variation is still prohibitively slow on large graph datasets, as will be demonstrated later in our experimental results section. We therefore present the first enhancements to the hierarchical graph indexing concepts of Contraction Hierarchies to support KLCSP problems as follows. We start with a brief overview of the existing limitations of Contraction Hierarchies for solving this particular problem below.

4.1 Limitations of CH

In order to showcase the limitations of CH for Kleene language constrained shortest paths, let us consider a simple example graph with label alphabet $\Sigma = \{r, g, b\}$, representing the colors *red*, *green*, and *blue*, respectively. This example graph is illustrated in Figure 3, where the edges have been colored according to their respective labels. In this scenario, when node v is contracted, a local search will be performed to find a potential witness path from node u to node w in the graph induced by the set of nodes "higher" in the hierarchy than node v (e.g., nodes u, w, x, and y). This local search will find a witness path, $\langle u, x, y, w\rangle$, with cost equal to 8, which happens to be less than the cost of the path $\langle u, v, w\rangle$, which is 10. In this case, no shortcut will be added between nodes u and w during the pre-processing. However, if we later wish to perform a Kleene language constrained shortest path query from u to w, in which we restrict the color *red* from our shortest path (i.e., our language constraint is $L((\Sigma \setminus \{r\})^*)$), then the bidirectional search will be unable to find any such path between u and w (since there are no valid shortcuts between u and w and the edge (x, y) will be invalid based on its *red* label), even though there exists a valid shortest path that avoids the color *red* in this graph: the path $\langle u, v, w\rangle$ with cost 10.

One naïve solution to this problem would be to establish a separate graph index for all possible subsets of the label alphabet Σ, and then use the appropriate index based on the incoming query constraints R. However, this is prohibitive, and would require the construction and maintenance of $2^{|\Sigma|}$ separate index datasets. Therefore, in the following sections, we propose methods to extend the concepts of Contraction Hierarchies to properly support any Kleene language constraints, and we prove the correctness of this approach, as well as providing experimental evidence in favor of this approach over other existing techniques (e.g., ALT).

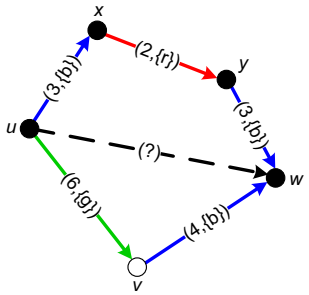


Figure 3: Contracting a labeled graph. Each edge, e, is labeled as $(w(e), \ell(e))$.

4.2 CHLR Index Construction

The revised contraction algorithm for graph index construction (shown in Algorithm 2) works as follows. The

[2]Backward search in a graph $G = (V, E)$ is the equivalent of performing a standard (i.e., forward) search in the graph $\bar{G} = (V, \bar{E})$, where $\bar{E} = \{(v, u) \mid (u, v) \in E\}$.

Algorithm 2 KLCSP-Contraction(G, ϕ)

Input: *Graph* $G = (V, E, w, \Sigma, \ell)$ *and bijective node order function* $\phi : V \rightarrow \{1, ..., |V|\}$
Output: *Augmented graph* $G' = (V, E \cup E', w, \Sigma, \ell)$, *where* E' *represents newly-added shortcut edges*

```
1: G' ← G
2: E' ← ∅
3: for all v ∈ V ordered by φ do
4:   for all e↓ = (u, v) ∈ E ∪ E' ordered by w(e↓) :
     φ(u) > φ(v) do
5:     for all e↑ = (v, w) ∈ E ∪ E' ordered by w(e↑) :
       φ(v) < φ(w) ∧ w ≠ u do
6:       G'_v ← G'[{z ∈ V | φ(v) < φ(z)}]
7:       R ← Σ \ {ℓ(e↓) ∪ ℓ(e↑)}
8:       shortcutCost ← w(e↓) + w(e↑)
9:       witnessCost ← KLCSP-Dijkstra(G'_v, u, w, R)
10:      if shortcutCost < witnessCost then
11:        e' ← (u, w)
12:        w(e') ← shortcutCost
13:        ℓ(e') ← {ℓ(e↓) ∪ ℓ(e↑)}
14:        E' ← E' ∪ {e'}
15:        G' ← G' ∪ E'
16:      end if
17:    end for
18:  end for
19: end for
20: return G'
```

algorithm processes each node $v \in V$ in the order defined by ϕ (which, for simplicity, we may assume is pre-defined). For each such node v, the algorithm considers all possible pairs of incoming edges $e_\downarrow = (u, v)$ and outgoing edges $e_\uparrow = (v, w)$, such that both u and w occur after v in the ordering defined by ϕ (i.e., they occur "higher" in the hierarchy). For each such pair of edges, the algorithm performs a KLCSP-Dijkstra search in the subgraph defined by G'_v (the subgraph of G' induced by nodes with "higher" hierarchy than v), using the set of restricted labels, R, defined to be the set of labels "avoided" (or not supported) by both $e_\downarrow$ and $e_\uparrow$. If the KLCSP-Dijkstra search is able to find an equivalent- or lesser-cost path than the path $\langle u, v, w \rangle$, which also avoids the same set of restricted labels avoided by both $e_\downarrow$ and $e_\uparrow$, then no shortcut edge is necessary (since there can be no possible constraint scenario for which the path $\langle u, v, w \rangle$ is required). Edges are processed in order of increasing weight (see Lines 4 and 5) to ensure that the total number of shortcut edges constructed by this process is minimal for the given ordering ϕ. See the appendix for a formal proof of both correctness and minimality.

4.2.1 Multi-Edge Support

One important aspect of the enhancements to the graph contraction algorithm shown above is that our graph index must now support multi-edges (i.e., parallel edges) due to the potential for multiple possible paths between a given pair of nodes in the graph, depending upon the set of restricted labels chosen for the query. For example, in the graph illustrated in Figure 4, if the nodes are contracted in order from bottom to top, we must now insert two separate shortcut edges between nodes u and w: edge e is necessary when contracting node v and edge e' is necessary when contracting node v'. Note that, in this particular case, we cannot simply replace one shortcut edge with the other when added, since they might both be necessary for ensuring correctness of the resulting shortest paths, depending upon the set of restricted labels. In particular, if the restricted label set is $R = \{r, b\}$, then the shortest path between u and w will make use of the shortcut edge e, giving a cost of 10 and a final (expanded) path of $\langle u, v, w \rangle$. However, if the restricted label set is $R = \{r, g\}$, then the shortest path between u and w will make use of the shortcut edge e', giving a cost of 12 and a final (expanded) path of $\langle u, v', w \rangle$.

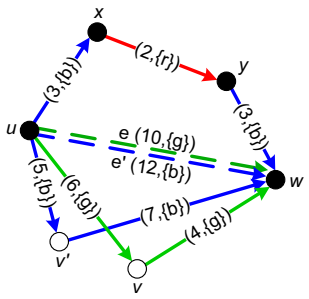


Figure 4: Multi-edge example.

4.3 CHLR Index Queries

Once the CHLR hierarchy has been established with the shortcut edge set, E', shortest path queries for any given restricted label set, $R \subseteq \Sigma$, may then be carried out as follows. The search algorithm employed is the same bidirectional Dijkstra search variant as is used for the static CH query algorithm (described in Section 3). However, we must now further augment the resulting upward and downward search graphs explored for a given query, respective of R. We redefine the upward search graph as $G_\uparrow = (V, E_\uparrow)$, where $E_\uparrow = \{e = (v, w) \in E \cup E' \mid \phi(v) < \phi(w) \wedge \ell(e) \cap R = \emptyset\}$, and the downward graph as $G_\downarrow = (V, E_\downarrow)$, where $E_\downarrow = \{e = (u, v) \in E \cup E' \mid \phi(u) > \phi(v) \wedge \ell(e) \cap R = \emptyset\}$. The CHLR query will now explore only those edges whose label sets are valid for the given query constraints.

4.4 Optimizations

As indicated in the KLCSP-Contraction index construction algorithm, during the contraction of a given node v, where $I_v^\downarrow = \{(u, v) \in E \cup E' \mid \phi(u) > \phi(v)\}$ and $O_v^\uparrow = \{(v, w) \in E \cup E' \mid \phi(v) < \phi(w)\}$, the algorithm performs a total of $|I_v^\downarrow| \cdot |O_v^\uparrow|$ calls to KLCSP-Dijkstra[3]. While correct and minimal (for a given ordering ϕ), the overall efficiency of the contraction of v can be improved by instead performing only a single local search from the source, u, of each incoming edge $e_\downarrow = (u, v) \in I_v^\downarrow$ until all nodes in the set $W = \{w \in V \mid (v, w) \in O_v^\uparrow\}$ have been settled, or until a distance of $w(e_\downarrow) + max\{w(e_\uparrow) \mid e_\uparrow = (v, w) \in O_v^\uparrow, w \neq u\}$ has been reached (this is similar to the approach used in [8]). Using this approach we can set $R \leftarrow \Sigma \setminus \ell(e_\downarrow)$ and pass this restricted label set to the augmented version of KLCSP-Dijkstra. Note that this does not affect the correctness of the resulting index, since the set R that we pass to KLCSP-Dijkstra in this case is a superset of the restricted label set passed to the KLCSP-Dijkstra calls in the original algorithm, for all possible pairs of incoming and outgoing

[3]Pairs $\langle e_\downarrow = (u, v), e_\uparrow = (v, w) \rangle$ where $u = w$ are ignored.

edges. This means that any resulting witness paths are still valid (i.e., they are more constrained than normal) and this approach can only result in a superset of (potentially superfluous) shortcuts to that of the original approach. Therefore, by taking this approach, we lose the property of minimality. However, initial experiments indicate that this approach scales much better in practice.

A more complex bidirectional version of this technique is used in [8] in which they first perform a single-hop backward search from all nodes $w \in W$ to their immediate neighbors in $X = \{x \in V \mid (x, w) \in E \cup E', w \in W, x \neq v\}$, and then perform the forward search from u to the target set X (instead of W). This allows the distance bound of the forward search to be further reduced to $w(e_\downarrow) + max\{w(e_\uparrow) - min\{w(e) \mid e = (x, w) \in E \cup E'\} \mid e_\uparrow = (v, w) \in O_v^\uparrow, w \neq u\}$. We further adapt this technique to our own language constrained variant by performing the restricted forward search from u as indicated in the paragraph above and by relaxing only edges $e = (x, w)$ for each node $w \in W$ in the single-hop backward search if $\ell(e) \subseteq \{\ell(e_\downarrow) \cup \ell(e_\uparrow)\}$, where $e_\uparrow = (v, w)$, thus preserving correctness.

Additionally, we employ the technique of using *hop limits* [8], in which we specify a limit on the number of hops that the paths on our local search can take. Each local search is aborted once the number of hops on the paths found by the search exceeds the specified constant limit. This can greatly speed up the local search times, but, like our other optimization, may result in unnecessary shortcuts being added during contraction. We use this optimized version of our algorithm for all subsequent experimental results.

5. EXPERIMENTAL ANALYSIS

5.1 Environment and Implementation

All experiments were carried out on a 64-bit server machine running Linux CentOS 5.3 with 2 quad-core CPUs clocked at 2.53 GHz with 72 GB RAM (although only one core was used per experiment). Our implementation of the CHLR technique is an extended implementation of the original Contraction Hierarchies source code, written in C++, and further detailed in [7]. Our implementation of the ALT algorithm (used for comparison against CHLR) is based on the algorithm described in [5]. All programs were compiled using gcc version 4.1.2 with optimization level 3.

5.2 Test Instances

For our experiments, we used the continent-wide graph dataset of North America (this includes only the US and Canada), represented by a total of $21,133,774$ nodes and $52,523,592$ edges. $6,779,795$ edges support one or more labels in this dataset, with 0.21 labels per edge, on average. Table 1 offers some additional information on the 16 different real-world labels supported in the North American graph dataset. This dataset (including labels) was derived from NAVTEQ transportation data products, under their permission.

5.3 Node Ordering

Our initial experiments were focused on determining a good approach for node ordering in the context of Kleene language constrained shortest paths. For this experiment, we took an approach similar to that of [7], in which we considered several different ordering metrics, along with several different combinations of weighted coefficients for each metric tested. In particular, we considered 6 unique ordering metrics (the first 5 of which come from [7]): *edge difference*, *contracted neighbors*, *original edges*, *search space depth*, *local search space size*, and a new priority term introduced here, which represents the number of new multi-edges introduced during the contraction of a node (*new multi-edges*).

Table 1: Graph Label Support for North America

Label	# Edges
Ferry	2,610
Toll Road	47,388
Unpaved Road	3,645,458
Private Road	1,662,314
Limited Access Road	682,396
4-Wheel-Drive-Only Road	139,284
Parking Lot Road	160,850
Hazmat Prohibited	45,950
All Vehicles Prohibited	64,414
Delivery Vehicles Prohibited	148,010
Trucks Prohibited	475,472
Taxis Prohibited	147,628
Buses Prohibited	151,272
Automobiles Prohibited	114,192
Pedestrians Prohibited	1,253,030
Through Traffic Prohibited	2,050,562

For each metric, we defined a range of possible values for their associated weight coefficient (e.g, $0-300$), as well as an incremental step size (e.g., 100). We then carried out experiments on all possible combinations of coefficients for these metrics, using the specified ranges and step sizes. In all, we tested $4,096$ (i.e., 4^6) combinations of the 6 different ordering metrics on a subgraph of the North American graph, representing the state of Virginia (with $483,504$ nodes and $1,113,602$ edges). For each configuration of coefficient values for these 6 metrics, the graph index was constructed using that particular configuration, and then a series of $10,000$ uniform random shortest path queries were run on the index (the same random pairs were used for each configuration for consistency). For each pair of nodes in the set of random test cases, we ran both a non-restricted search (i.e., no labels were restricted; $R = \emptyset$) and a fully-restricted search (i.e., all labels were restricted; $R = \Sigma$)[4].

From these results, we calculated the product of the construction time of the index and the average overall query time (considering both the unrestricted and restricted results together), and then chose the configuration with the smallest such product value. The smallest of these products can be seen as a good compromise of construction time and query time. From these experiments, we found that a combination of only 2 particular ordering metrics was sufficient to produce the best overall results for the graphs tested here. In particular, for all subsequent experiments carried out here, we have chosen to use only the *edge difference* metric, with a weighted coefficient of 100, and the *original edges* metric, with a weighted coefficient of 200.

5.4 Comparative Results

In Table 2, we present the results of this approach when applied to the full North American graph. This table com-

[4]This is feasible since not all edges support labels in our test datasets.

pares both the preprocessing and query results of CHLR against the bidirectional adaptation of Dijkstra's algorithm, as well as the ALT algorithm, constructed using 64 landmarks (ALT-64). As with the original node ordering experiments, for the queries, we take the averages of 10,000 random unrestricted queries (where $R = \emptyset$) and 10,000 random restricted queries (where $R = \Sigma$). Even though the CHLR technique requires nearly 3 times the preprocessing time than that of ALT-64 for the North American graph, we are able to achieve 3 orders of magnitude improvements in both search space and query times over both the Dijkstra algorithm and ALT-64, on average (this is due primarily to the effectiveness of the shortcut edges in CHLR, which greatly reduce the resulting search space, and thus, the query times). However, as we will see in later experiments, the overall performance of these techniques can strongly depend on the chosen set of restricted labels.

Table 2: Experiments on the North American Graph Dataset

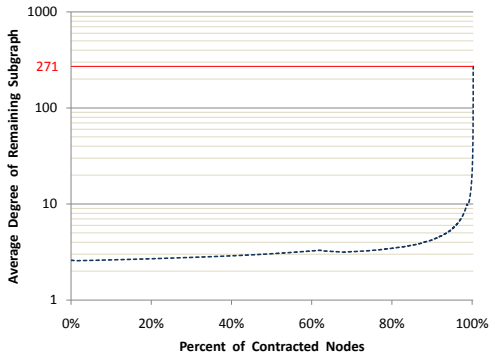
	Preprocessing		Queries	
Technique	Time [H:M]	Space [B/node]	# Settled Nodes	Time [ms]
Bidir. Dijkstra	0:00	0	6,799,486	3,043.89
ALT-64	0:49	512	1,141,430	1,528.80
CHLR	2:10	62	993	2.18

5.5 Degree Limits

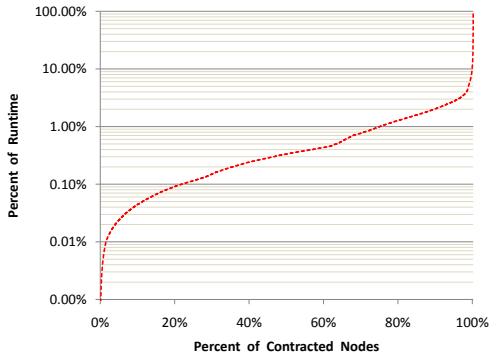
An adverse side effect to the fact that this new approach must now support multi-edges is that of degree explosion during the graph index construction. As indicated in Figure 5a, as the index construction proceeds for the North American graph, the average degree of the remaining subgraph quickly grows from 10 to 271 during contraction of the last 2% of the nodes. This degree explosion can also be seen to have a strong impact on the overall runtime of the index construction algorithm in practice, where, for the North American graph, roughly 90% of the runtime was spent contracting only the last 1% of the nodes (see Figure 5b).

In order to combat this effect, we introduce the concept of a *degree limit* within the construction algorithm, in which contraction of the remaining nodes is aborted as soon as the average degree of the remaining nodes reaches some critical threshold, as defined by the limit. The remaining (uncontracted) nodes in the graph make up what are called the *core* nodes of the graph index (a concept introduced and explored in [13] and also in [7] for many-to-many shortest path searches and in [4] for goal-directed routing). Once the contraction is aborted after reaching the degree limit, then, for all remaining nodes, v, in the core, we set $\phi(v) = |V|$ [5]. To maintain correctness of results, we then need only adjust our search graphs as follows. We set $G_\uparrow = (V, E_\uparrow)$, where $E_\uparrow = \{e = (v, w) \in E \cup E' \mid \phi(v) \leq \phi(w) \wedge \ell(e) \cap R = \emptyset\}$ and $G_\downarrow = (V, E_\downarrow)$, where $E_\downarrow = \{e = (u, v) \in E \cup E' \mid \phi(u) \geq \phi(v) \wedge \ell(e) \cap R = \emptyset\}$ (i.e., we no longer maintain a strict node ordering, but instead rely only on a partial ordering). Using these search graphs, the algorithm still maintains correctness; however, searching in the core becomes more exhaustive due to the relaxed filtering.

[5]The function ϕ is no longer a bijective function in this context.



(a) Average Degree Progression



(b) Runtime Percentage

Figure 5: Effects of Degree Explosion During Construction of the North American Graph Dataset

Table 3 shows the results of our experiments over several different degree limits on the North American graph. As can be seen, the index construction time can be greatly reduced by using reasonable degree limits, without sacrificing too much of the overall speed of any subsequent queries on the index. Even for the smallest degree limit of 10, with the worst query times, we are still able to outperform the ALT-64 results from Table 2 by an order of magnitude (on average), requiring only 6 minutes of preprocessing time.

Table 3: Degree Limit Experiments on the North American Graph Dataset

	Preprocessing		Queries		
Degree Limit	Time [H:M]	Space [B/node]	# Settled Nodes	Time [ms]	**Core Size**
10	0:06	60	238,513	130.14	252,719
20	0:13	61	59,244	40.44	64,153
30	0:18	62	28,732	25.32	30,863
40	0:23	62	16,807	17.12	17,677
50	0:29	62	11,212	11.63	11,541
100	0:57	62	3,577	4.96	3,184
200	1:43	62	1,236	2.63	498

5.6 Effects of Restriction Cardinality

Here we present experiments on the overall effects of the number of restricted labels chosen for a given KLCSP query. For this set of experiments, we compare the CHLR technique against the ALT-64 technique. Since the North American

graph supports only 16 different labels, we perform 17 sets of experiments, one for each possible size of the restricted label set, $|R| = 0, \cdots, 16$. For each of the 17 possible cardinalities of R, we perform a set of 10,000 uniform random shortest path queries. For each of the random pairs of vertices in the test set for a given cardinality, i, we choose a random restricted label set $R \subseteq \Sigma$, such that $|R| = i$. The results of this experiment are presented as a box-and-whisker plot in Figure 6.

An interesting property emerges from our proposed CHLR technique, as compared to ALT-64 in these experiments. In particular, we can see that, the more restricted the shortest path query is, the better the CHLR technique performs, in general. Alternatively, the performance of ALT-64 actually becomes much worse as the query becomes more restricted (by up to an order of magnitude). The improvements in performance of the CHLR technique as the queries become more restricted can be attributed to the fact that more of the shortcut edges are also now likely to be restricted, thus pruning the search space even more than in the relatively unrestricted cases. The degradation of performance for ALT-64 is primarily due to the fact that the potential functions computed during preprocessing become much weaker in general as the dynamic constraints on the graph continue to change, as indicated in [5].

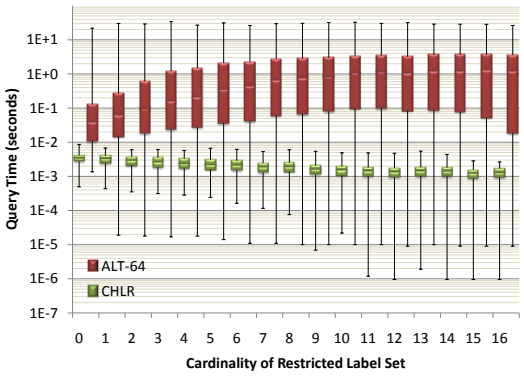


Figure 6: Experiments on Restriction Cardinality

6. CONCLUSION

We have presented and formalized a new shortest path query type as a variant of language constrained shortest path problems. We have also successfully extended the graph indexing technique known as Contraction Hierarchies to efficiently support this new dynamically constrained query type. Experimental results on real-world graph data indicate that this new technique is several orders of magnitude better than Dijkstra's algorithm and the ALT algorithm, both in terms of query time and search space. Additionally, the performance of this technique also seems to improve under more heavily constrained query scenarios, making it a perfect candidate for supporting this new query type.

While this technique has proven highly applicable on real-world road network data, in the future, we would like to further explore the overall robustness of our technique on different synthetically labeled graph configurations. It is anticipated that this will allow us to examine the properties of graph labeling which can affect the relative performance and scalability of our proposed technique. Initial experiments indicate that this technique remains practical for graphs which exhibit high average label autocorrelation (i.e., local self-similarity of edge labels) and/or high average label density (i.e., average proportion of supported labels per edge), although more thorough experimentation is needed.

Additional future work also includes extending the concepts of this research to more complex edge restriction types, such as height and weight restrictions for road networks.

7. ACKNOWLEDGEMENTS

We would like to thank R. Geisberger and P. Sanders for providing us with their original implementation of the source code for static Contraction Hierarchies. We would also like to thank NAVTEQ for allowing us the use of their transportation data products in our analysis. Finally, we thank M. Chrobak for his helpful feedback. This work was partially supported by NSF grants IIS-0705916 and IIS-0803410.

8. REFERENCES

[1] C. L. Barrett, K. Bisset, M. Holzer, G. Konjevod, M. V. Marathe, and D. Wagner. Engineering label-constrained shortest-path algorithms. In *AAIM*, pages 27–37, 2008.

[2] C. L. Barrett, K. Bisset, R. Jacob, G. Konjevod, and M. V. Marathe. Classical and contemporary shortest path problems in road networks: Implementation and experimental analysis of the transims router. In *ESA*, pages 126–138, 2002.

[3] C. L. Barrett, R. Jacob, and M. V. Marathe. Formal-language-constrained path problems. *SIAM J. Comput.*, 30(3):809–837, 2000.

[4] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, and D. Wagner. Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm. In *WEA*, pages 303–318, 2008.

[5] D. Delling and D. Wagner. Landmark-based routing in dynamic graphs. In *WEA*, pages 52–65, 2007.

[6] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[7] R. Geisberger. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. Master's thesis, Institut fur Theoretische Informatik Universitat Karlsruhe, 2008.

[8] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*, pages 319–333, 2008.

[9] A. V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. In *SODA*, pages 156–165, 2005.

[10] A. V. Goldberg and R. F. Werneck. Computing point-to-point shortest paths from external memory. In *ALENEX/ANALCO*, pages 26–40, 2005.

[11] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. In *IEEE Transactions on System Science and Cybernetics*, volume 4, 1968.

[12] M. Holzer, F. Schulz, and D. Wagner. Engineering multilevel overlay graphs for shortest-path queries. *ACM Journal of Experimental Algorithmics*, 13, 2008.

[13] S. Knopp, P. Sanders, D. Schultes, F. Schulz, and D. Wagner. Computing many-to-many shortest paths using highway hierarchies. In *ALENEX*, 2007.

[14] P. Sanders and D. Schultes. Engineering highway hierarchies. In *ESA*, pages 804–816, 2006.

[15] P. Sanders, D. Schultes, and C. Vetter. Mobile route planning. In *ESA*, pages 732–743, 2008.

[16] D. Schultes and P. Sanders. Dynamic highway-node routing. In *WEA*, pages 66–79, 2007.

APPENDIX

A. CORRECTNESS AND MINIMALITY

LEMMA A.1. *Let $P^{R'}_{s,t}$ define an R'-restricted shortest path from $s \in V$ to $t \in V$ for some $R' \subseteq \Sigma$. For any $R \subseteq R'$, $w(P^R_{s,t}) \leq w(P^{R'}_{s,t})$.*

PROOF. Suppose there exists an R-restricted shortest path $P^R_{s,t}$ such that $w(P^R_{s,t}) > w(P^{R'}_{s,t})$. The path $P^{R'}_{s,t}$ is clearly a valid path for the restricted label set R too, since $R \subseteq R'$ and, by definition, $P^{R'}_{s,t}$ must therefore avoid all restricted labels in R as well. However, this contradicts the optimality of $P^R_{s,t}$. □

THEOREM A.2. *Given a graph $G' = (V, E \cup E', w, \Sigma, \ell)$ constructed by the KLCSP-Contraction algorithm, the query algorithm is **correct** for any $s \in V$, $t \in V$, and $R \subseteq \Sigma$.*

PROOF. For consistency, we extend the original proof of correctness presented in [7] for static Contraction Hierarchies to support our new language constrained variant. For a given path $P_{s,t} = \langle s = v_0, \cdots, v_i, \cdots, v_k = t \rangle$, let $M_{P_{s,t}} = \{v_i \in P_{s,t} \mid 0 < i < k, \phi(v_{i-1}) > \phi(v_i) < \phi(v_{i+1})\}$ (i.e., the set of all local minima in $P_{s,t}$ with respect to ϕ). We can classify all paths, $P_{s,t}$, in a given graph into one of two basic forms: (1) those with $M_{P_{s,t}} = \emptyset$ and (2) those with $M_{P_{s,t}} \neq \emptyset$.

Since the search algorithm only searches forward in the *upward* graph $G_\uparrow$ and *backward* in the downward graph $G_\downarrow$ (i.e., ϕ is strictly increasing in each search direction), then it will explore only paths of the form (1) during the search. For any origin node $s \in V$, destination node $t \in V$, and restricted label set $R \subseteq \Sigma$, suppose there exists a shortest path $P^R_{s,t}$ of the form (2) above in the original graph. We must now prove the claim that there must also exist an alternate (and equivalent) shortest path of the form (1) above after the KLCSP-Contraction algorithm has been run on the graph.

Since $M_{P^R_{s,t}} \neq \emptyset$, let $m(P^R_{s,t}) = min\{\phi(v) \mid v \in M_{P^R_{s,t}}\}$. Let v_i be the node in path $P^R_{s,t}$ such that $\phi(v_i) = m(P^R_{s,t})$ (i.e., v_i is the lowest order node in $M_{P^R_{s,t}}$). For edges $e_i = (v_{i-1}, v_i)$ and $e_{i+1} = (v_i, v_{i+1})$ in a shortest path $P^R_{s,t}$ of the form (2) above, let $R' = \Sigma \setminus \{\ell(e_i) \cup \ell(e_{i+1})\}$. We first demonstrate that $R \subseteq R'$.

Suppose for the sake of contradiction that $R \nsubseteq R'$. This implies that $\exists \alpha \in R$, such that either $\alpha \in \ell(e_i)$ or $\alpha \in \ell(e_{i+1})$. In either case, the subpath $\langle e_i, e_{i+1} \rangle$ is invalid for any R-restricted shortest path, contradicting the validity of $P^R_{s,t}$. Therefore, in this context, $R \subseteq R'$ must be true.

Next, let us consider the hypothetical scenario where we perform a call to the KLCSP-Dijkstra search algorithm to find an R-restricted shortest path $P^R_{v_{i-1},v_{i+1}}$ in the subgraph $G'_{v_i} = G'[\{z \in V \mid \phi(v_i) < \phi(z)\}]$. If $w(P^R_{v_{i-1},v_{i+1}}) < w(e_i) + w(e_{i+1})$, then there exists a *shorter* R-restricted path between v_{i-1} and v_{i+1} in G'_{v_i} (that does not include e_i or e_{i+1}), contradicting the optimality of $P^R_{s,t}$. Therefore, $w(P^R_{v_{i-1},v_{i+1}}) \geq w(e_i) + w(e_{i+1})$ must hold true. Given that $R \subseteq R'$, then by Lemma A.1, we know that $w(P^{R'}_{v_{i-1},v_{i+1}}) \geq w(P^R_{v_{i-1},v_{i+1}})$ must also hold true. This gives us $w(P^{R'}_{v_{i-1},v_{i+1}}) \geq w(e_i) + w(e_{i+1})$. Note that R' is exactly equal to the restricted label set used in the search for a restricted witness path during the graph index construction of the KLCSP-Contraction algorithm when processing node v_i, where $e_\downarrow = e_i$ and $e_\uparrow = e_{i+1}$. In the case where $w(P^{R'}_{v_{i-1},v_{i+1}}) > w(e_i) + w(e_{i+1})$, then the KLCSP-Contraction algorithm will have added a shortcut edge from v_{i-1} to v_{i+1} with weight $w(e_i) + w(e_{i+1})$. In the case where $w(P^{R'}_{v_{i-1},v_{i+1}}) = w(e_i) + w(e_{i+1})$, then this means that there already exists an alternate and equivalent-cost path in the subgraph G'_{v_i}, defined above. Either way, we can construct a new path $\bar{P}^R_{s,t}$ which bypasses v_i altogether (using either the shortcut or the path between v_{i-1} and v_{i+1} in G'_{v_i}; since $R \subseteq R'$, either is valid for R), such that $w(\bar{P}^R_{s,t}) = w(P^R_{s,t})$. If $\bar{P}^R_{s,t}$ is of the form (1), then the proof is complete. If $\bar{P}^R_{s,t}$ is itself of the form (2), then, since $v_i \notin \bar{P}^R_{s,t}$ and $\phi(v_i) = m(P^R_{s,t})$, we know that $m(\bar{P}^R_{s,t}) > m(P^R_{s,t})$, and we can apply the same argument (as above) recursively to $\bar{P}^R_{s,t}$. Since there are only a finite number of possible levels in ϕ (i.e., the function m cannot increase indefinitely), then this recursive argument must eventually produce an alternate path $\bar{P}^R_{s,t}$, such that $M_{\bar{P}^R_{s,t}} = \emptyset$.

Therefore, for any shortest path $P^R_{s,t}$ of the form (2) above, there also exists an alternate and equivalent shortest path of the form (1) above. Since the query algorithm performs a shortest path search amongst all and only the paths of the form (1), then the query algorithm is correct for any $s \in V$, $t \in V$, and $R \subseteq \Sigma$. □

Another property that we wish to discuss in this work, which has not been previously addressed even for static Contraction Hierarchies, is that of edge minimality for a given ordering ϕ. One might be easily tempted to believe that, when processing edges $e_\downarrow = (u, v)$ and $e_\uparrow = (v, w)$ in arbitrary order, where $R = \Sigma \setminus \{\ell(e_\downarrow) \cup \ell(e_\uparrow)\}$, if $P^R_{u,w} \nsubseteq G'_v$ then a shortcut edge (u, w) is absolutely necessary for correctness. However, this is not always the case. Consider the example graph in Figure 7. If we start the contraction of v by first processing edges $e_\downarrow = (u, v)$ and $e_\uparrow = (v, w)$, then clearly $P^R_{u,w} \nsubseteq G'_v$ (since G'_v contains only the edges (u, x) and (y, w)). Regardless, it turns out that there is still no need to add a (u, w) shortcut edge for this scenario (since $P^R_{u,w} \neq \langle u, v, w \rangle$). To demonstrate why, consider what happens if we had first processed the edges $e_\downarrow = (x, v)$ and $e_\uparrow = (v, y)$. In this case, we would have added a shortcut edge (x, y) with a weight of 3. If we then process edges $e_\downarrow = (u, v)$ and $e_\uparrow = (v, w)$, $P^R_{u,w} \subseteq G'_v$ will be true, in which case, no shortcut is necessary. In fact, in the extreme case for this degenerate example, if we process the pairs of edges in the order $\langle e_\downarrow = (u, v), e_\uparrow = (v, w) \rangle$, then $\langle e_\downarrow = (u, v), e_\uparrow = (v, y) \rangle$, then $\langle e_\downarrow = (x, v), e_\uparrow = (v, w) \rangle$, and finally $\langle e_\downarrow = (x, v), e_\uparrow = (v, y) \rangle$, this will result in the addition of 4 separate shortcut edges (one for each pair). However, if we process these same pairs of edges in the reverse order of that above, we will have added only 1 shortcut edge (x, y). Also note that we cannot simply remove edges (u, v) or (v, w) from this graph, since they may be necessary depending on the incoming label constraint (e.g., $P^{\{b\}}_{u,v} = (u, v)$).

Therefore, even in the case where $P^R_{u,w} \neq \langle u, v, w \rangle$, if we are not careful to process the adjacent edges in the correct order, we may be unable to find a valid path $P^R_{u,w} \subseteq G'_v$, resulting in unnecessary shortcut edges. In particular, we need a way to ensure that, when processing edges $e_\downarrow = (u, v)$ and

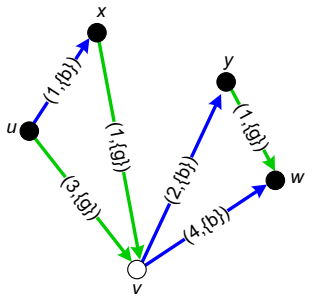


Figure 7: Counter-example showing lack of minimality when edges are considered in arbitrary order.

$e_\uparrow = (v, w)$, either $P^R_{u,w} = \langle u, v, w\rangle$ or $P^R_{u,w} \subseteq G'_v$ always holds true. The following lemma suggests that this property will be met if we process all adjacent edges in order of increasing weight (as shown in Algorithm 2).

LEMMA A.3. *Let $e_\downarrow = (u, v)$ and $e_\uparrow = (v, w)$ be the pair of edges currently being processed by the KLCSP-Contraction algorithm during contraction of node $v \in G'$. Either $P^R_{u,w} = \langle u, v, w\rangle$ or $P^R_{u,w} \subseteq G'_v$ must hold true.*

PROOF. Suppose for the sake of contradiction that $P^R_{u,w} \neq \langle u, v, w\rangle$ and $P^R_{u,w} \nsubseteq G'_v$. Note that $P^R_{u,w} \nsubseteq G'_v$ implies that $v \in P^R_{u,w}$, while $P^R_{u,w} \neq \langle u, v, w\rangle$ implies that $(u, v) \notin P^R_{u,w}$, or $(v, w) \notin P^R_{u,w}$, or both.

Consider the case where $e = (v, w) \notin P^R_{u,w}$. Since we know that $v \in P^R_{u,w}$, then $P^R_{u,w} = \langle u, \cdots, v, y, \cdots, w\rangle$ such that $e' = (v, y) \in E \cup E'$ (i.e., if $(v, w) \notin P^R_{u,w}$ and $v \in P^R_{u,w}$, then v must reach node w through some other edge $e' = (v, y)$). This means that $w(e') < w(e)$, otherwise we could construct a lesser-cost path $P^R_{u,w}$ that actually includes (v, w). However, since we process all outgoing edges $e_\uparrow$ in order of increasing weight in the construction algorithm, then $w(e') < w(e)$ implies that we must have already processed the pair $\langle e_\downarrow = (u, v), e_\uparrow = (v, y)\rangle$. By definition, this means that $P^R_{u,y} \subseteq G'_v$, so, using this subpath, we can construct a path from u to w that avoids v such that $P^R_{u,w} \subseteq G'_v$, leading to a contradiction. A symmetric argument holds for the case where $e = (u, v) \notin P^R_{u,w}$, relative to the fact that we process all incoming edges in order of increasing weight as well. □

THEOREM A.4. *Given a fixed node ordering function, ϕ, the edge set E' constructed by the KLCSP-Contraction algorithm is **minimal** (i.e., there is no algorithm which can produce a smaller set of shortcut edges, while still guaranteeing correctness).*

PROOF. Suppose there exists some algorithm which can construct a set of shortcut edges, E'', from the ordering ϕ, such that $|E''| < |E'|$, and the set E'' is correct for any possible restricted label set $R \subseteq \Sigma$. This means there must exist some edge $e = (u, w)$, such that $e \in E'$ and $e \notin E''$. Since $e \in E'$, then by definition, there must also exist some node v, such that $\phi(u) > \phi(v) < \phi(w)$, $e_\downarrow = (u, v), e_\uparrow = (v, w) \in E \cup E'$, and $w(e) = w(e_\downarrow) + w(e_\uparrow)$. Let $R = \Sigma \setminus \{\ell(e_\downarrow) \cup \ell(e_\uparrow)\}$. By Lemma A.3, we know that, when processing $e_\downarrow$ and $e_\uparrow$ to contract node v, either $P^R_{u,w} = \langle u, v, w\rangle$ or $P^R_{u,w} \subseteq G'_v$ must hold true. If $P^R_{u,w} \subseteq G'_v$, then, by definition, if such a path exists, the index construction algorithm would not have added a shortcut from u to w, contradicting the fact that $e \in E'$. However, if $P^R_{u,w} = \langle u, v, w\rangle$, then E'' is incorrect for the query to find $P^R_{u,w}$, since $e \notin E''$. Either case leads to a contradiction. □

B. ALTERNATIVE INDEX CONSTRUCTION

Another way of looking at the previous problem suggested by Figure 7 is that, by always omitting the junction v from the induced subgraph G'_v, the local search will never be able to find witness paths of the form $P^R_{u,w} = \langle u, \cdots, x, v, y, \cdots, w\rangle$ such that $u \neq x$ and/or $y \neq w$, resulting in the possibility of adding (u, w) shortcuts unnecesssarily. One alternative solution to that suggested above is not to omit v from G'_v, but rather, to include v, and instead model a "turn restriction" in the local search, in which we do not allow the local search to perform a transition from the incoming edge (u, v) to the outgoing edge (v, w). This will ensure that we find the best possible path from u to w other than the path $\langle u, v, w\rangle$, including any valid witness paths of the above form. If this alternate path cost is less than or equivalent to the cost of path $\langle u, v, w\rangle$, then no shortcut is needed, and both minimality and correctness remain preserved.

Using this alternative approach, the order of the local searches relative to v then becomes irrelevant, making this methodology more efficient in practice (since we no longer have to rely on sorting adjacent edges). However, this requires a more complex redefinition of the local search algorithm.

Here we present the pseudocode for these alternative local search and index construction algorithms, as well as a brief discussion of correctness and minimality. For the purposes of this discussion, we shall call the new local search and index construction algorithms KLCSP-Dijkstra-Alt and KLCSP-Contraction-Alt, respectively.

We start with the revised local search algorithm (KLCSP-Dijkstra-Alt). This local search algorithm behaves almost exactly the same as before, except we now keep track of the parent node for each node in the current shortest path tree. We store this information in the newly added p array. This information is used to ensure that we do not allow the local search to make a transition from the incoming edge (s, r) to the outgoing edge (r, t), as defined by the input parameter constraints. This "turn restriction" is enforced in Line 13 when deciding which edges to relax from the current node during the search. Here, the search will skip the relaxation of the edge (u, v) if $(p[u] = s) \wedge (u = r) \wedge (v = t)$ is true, which indicates the restricted transition from (s, r) to (r, t). We note, however, that this constraint alone is not sufficient to fully guarantee that we always find the best alternate path from s to t, other than $\langle s, r, t\rangle$.

For example, consider the graph presented in Figure 7. Assume that we have reduced the weight of edge (u, v) in this graph to 2. If we call KLCSP-Dijkstra-Alt to find the shortest path from u to y other than $\langle u, v, y\rangle$, then node v will be relaxed from parent edge (u, v) first during the search (giving $p[v] = u$ and $d[v] = 2$). By the time we relax edge $e = (x, v)$ in the local search, we will not be able to improve the value $d[v]$ at Line 15 (since $d[x] + w(e) = d[v]$). If we leave $p[v] = u$, then we will be unable to find *any* valid path from u to y, since we will ultimately restrict the relaxation of edge (v, y) due to the transition from (u, v). However, there is still an equivalent cost shortest path from u to y

other than $\langle u, v, y\rangle$: $\langle u, x, v, y\rangle$.

To ensure that we are able to find such alternate, equivalent paths, we must include the additional condition at Line 23 to always "prefer" equivalent-cost paths from node s to node r other than the incoming edge (s, r) in the local search, thus eliminating this problem. Note that we do not have to add similar constraints for preferring equivalent-cost paths from r to t other than (r, t), since the edge (r, t) will only be relaxed if $p[r] \neq s$.

Algorithm 3 KLCSP-Dijkstra-Alt(G, s, r, t, R)

Input: *Graph* $G = (V, E, w, \Sigma, \ell)$, $s, r, t \in V$, *restricted alphabet* $R \subseteq \Sigma$
Output: *Cost of shortest path* $P^R_{s,t}$, such that $P^R_{s,t} \neq \langle s, r, t\rangle$

```
1: PQ ← ∅
2: for all v ∈ V do
3:    d[v] ← ∞
4:    p[v] ← null
5: end for
6: d[s] ← 0
7: PQ.Insert(s, d[s])
8: while ¬PQ.Empty() do
9:    u ← PQ.ExtractMin()
10:   if u = t then
11:      return d[t]
12:   end if
13:   for all e = (u, v) ∈ E : (p[u] ≠ s) ∨ (u ≠ r) ∨ (v ≠ t) do
14:      if ℓ(e) ∩ R = ∅ then
15:         if d[u] + w(e) < d[v] then
16:            d[v] ← d[u] + w(e)
17:            p[v] ← u
18:            if v ∉ PQ then
19:               PQ.Insert(v, d[v])
20:            else
21:               PQ.DecreaseKey(v, d[v])
22:            end if
23:         else if d[u] + w(e) = d[v] ∧ u ≠ s ∧ v = r then
24:            p[v] ← u
25:         end if
26:      end if
27:   end for
28: end while
29: return ∞
```

The index construction algorithm (KLCSP-Contraction-Alt) is then changed to omit the ordering of the edges relative to v by weight (since this is no longer necessary, as we will show below), as well as to include v in the induced subgraph G'_v, and, finally, to call the new local search algorithm.

For the remainder of the discussion, we must first clarify some potentially troublesome notation. We note that, in the context of the original KLCSP-Contraction algorithm pseudocode, the induced subgraph G'_v is defined such that $v \notin G'_v$ when contracting node v. However, in the context of the new KLCSP-Contraction-Alt algorithm pseudocode, we have that $v \in G'_v$. We shall refer here only to this latter subgraph definition of G'_v.

In Lemma A.3, we showed that, by processing adjacent edges in order of increasing weight when contracting node

Algorithm 4 KLCSP-Contraction-Alt(G, ϕ)

Input: *Graph* $G = (V, E, w, \Sigma, \ell)$ *and bijective node order function* $\phi : V \to \{1, ..., |V|\}$
Output: *Augmented graph* $G' = (V, E \cup E', w, \Sigma, \ell)$, *where* E' *represents newly-added shortcut edges*

```
1: G' ← G
2: E' ← ∅
3: for all v ∈ V ordered by φ do
4:    for all e↓ = (u, v) ∈ E ∪ E' : φ(u) > φ(v) do
5:       for all e↑ = (v, w) ∈ E ∪ E' : φ(v) < φ(w) ∧ w ≠ u do
6:          G'_v ← G'[{z ∈ V | φ(v) ≤ φ(z)}]
7:          R ← Σ \ {ℓ(e↓) ∪ ℓ(e↑)}
8:          shortcutCost ← w(e↓) + w(e↑)
9:          witnessCost ← KLCSP-Dijkstra-Alt(G'_v, u, v, w, R)
10:         if shortcutCost < witnessCost then
11:            e' ← (u, w)
12:            w(e') ← shortcutCost
13:            ℓ(e') ← {ℓ(e↓) ∪ ℓ(e↑)}
14:            E' ← E' ∪ {e'}
15:            G' ← G' ∪ E'
16:         end if
17:      end for
18:   end for
19: end for
20: return G'
```

v, we can guarantee that, when processing edges $e_\downarrow = (u, v)$ and $e_\uparrow = (v, w)$, either $P^R_{u,w} = \langle u, v, w\rangle$ or $P^R_{u,w} \subseteq G'_v \setminus \{v\}$ must (already) be true. Here, we prove a slightly different claim for the KLCSP-Contraction-Alt algorithm.

LEMMA B.1. *Let* $e_\downarrow = (u, v)$ *and* $e_\uparrow = (v, w)$ *be the pair of edges currently being processed by the KLCSP-Contraction-Alt algorithm during contraction of node* $v \in G'$. *Either* $P^R_{u,w} = \langle u, v, w\rangle$ *is true or* $P^R_{u,w} \subseteq G'_v \setminus \{v\}$ *will* ***eventually*** *be true (specifically, by the time we are finished contracting node* v*).*

PROOF. It suffices to consider the case where $P^R_{u,w} \neq \langle u, v, w\rangle$ in G'_v and $P^R_{u,w} \nsubseteq G'_v \setminus \{v\}$. This implies that $v \in P^R_{u,w}$, and, therefore, $P^R_{u,w} = \langle u, \cdots, x, v, y, \cdots, w\rangle$ such that $u \neq x$ and/or $y \neq w$. However, since $P^R_{x,y} = \langle x, v, y\rangle$, then when the construction algorithm (eventually) processes the edges $e_\downarrow = (x, v)$ and $e_\uparrow = (v, y)$, the algorithm will be forced to add shortcut edge (x, y), by definition. Therefore, when the contraction of v is complete, there must exist a path $P^R_{u,w} = \langle u, \cdots, x, y, \cdots, w\rangle \subseteq G'_v \setminus \{v\}$. □

Note that this property holds true even in the context of the original KLCSP-Contraction algorithm. However, for the original construction algorithm, we had to prove the stronger claim that, if $P^R_{u,w} \neq \langle u, v, w\rangle$, then $P^R_{u,w} \subseteq G'_v \setminus \{v\}$ must (already) be true. This is because the previous algorithm would only avoid adding a (u, w) shortcut if this latter condition already held. However, the new local search algorithm is able to find witness paths of the form shown in the above lemma (to detect that $P^R_{u,w} \subseteq G'_v \setminus \{v\}$ will eventually be true), and no shortcut edge will be added in this scenario. We now have that the KLCSP-Contraction algorithm and the KLCSP-Contraction-Alt algorithm will both only add a shortcut edge (u, w) if $P^R_{u,w} = \langle u, v, w\rangle$ (and this is the only shortest path) when processing edges

$e_{\downarrow}$ and $e_{\uparrow}$ (based on the properties of Lemmas A.3 and B.1, respectively). Therefore, they will generate the exact same shortcut edge set for a given ordering ϕ. Correctness and minimality of the KLCSP-Contraction-Alt algorithm thus follows from equivalence.

C. ADDITIONAL EXPERIMENTS

In this section, we present additional experiments on several large, statewide graphs, with different overall topologies and label distributions (based on the same 16 available labels presented in the earlier Experiments section). This is intended to further showcase the CHLR technique's general applicability across a range of large, relatively-diverse, real-world road networks. In Table 4, we present the datasets tested here, including their respective sizes.

Table 4: Statewide Graph Datasets

Name	# Nodes	# Edges
Alabama	405,205	988,040
California	1,478,976	3,623,111
Georgia	640,455	1,542,577
Louisiana	330,250	805,565
New York	624,220	1,536,789
North Carolina	687,648	1,609,475
Pennsylvania	718,318	1,805,931
South Carolina	381,349	929,420
Virginia	483,504	1,113,602

For each of these statewide graphs, we have constructed the CHLR index and performed 10,000 random unrestricted ($R = \emptyset$) queries and 10,000 random restricted ($R = \Sigma$) queries. The index construction times and data storage overhead for each graph are presented in Table 5, along with the average query search space size and runtimes.

Table 5: Experiments on the Statewide Graph Datasets

	Preprocessing		**Queries**	
Dataset	Time [H:M:S]	Space [B/node]	# Settled Nodes	Time [ms]
Alabama	0:00:18	59	374	0.215
California	0:02:16	61	369	0.295
Georgia	0:00:37	57	632	0.42
Louisiana	0:00:16	61	255	0.135
New York	0:01:06	66	453	0.34
North Carolina	0:00:44	54	425	0.305
Pennsylvania	0:01:07	64	507	0.395
South Carolina	0:00:13	57	371	0.175
Virginia	0:00:24	56	305	0.185

As can be seen from these results, the overall performance metrics remain very efficient across each of these separate graphs. Despite the variable differences in some of the metrics shown here for each graph (which arise due to differences in graph topologies, edge costs, and label distributions), the results suggest that the CHLR technique performs quite successfully across all of these diverse, real-world datasets. Specifically, preprocessing times on the order of minutes and sub-millisecond query times for such large graphs are considered highly-effective and practical for any real-world applications.

However, a more thorough analysis is still required to further assess the actual effects that different graph topologies, edge costs, and label distributions and densities can have on the resulting graph index construction for CHLR. As indicated in the Conclusion section of this paper, a reasonable next step in this assessment would be to carry out more rigorous experiments on different synthetically-labeled graph datasets, in order to simulate the range of possible labelings for a given graph topology.