Computing Constrained Shortest-Paths at Scale

ALBERTO VERA, Cornell University
SIDDHARTHA BANERJEE, Cornell University
SAMITHA SAMARANAYAKE, Cornell University

Motivated by the needs of modern on-demand transportation platforms for robust travel-time estimates, we study the problem of computing constrained shortest paths (CSP) at scale via preprocessing and network augmentation techniques. Our work makes two fundamental contributions in this regard:
1. We propose a preprocessing and network augmentation technique for scalable CSP queries, which builds on the *hub labels* approach for shortest path (SP) computation. The performance of our algorithms can be parametrized in terms of a new network primitive, the *constrained highway dimension*, analogous to the highway dimension for SP queries; we show how the two notions can be related, thereby providing theoretical characterization of networks which enable fast CSP queries.
2. We develop practical algorithms for scalable CSP computation, augmenting our theory with additional network clustering and ranking heuristics. We evaluate our algorithm on real-world datasets and show that our algorithms are orders of magnitude faster compared to existing techniques, and also have small per-node storage and good preprocessing times, even on a single machine.

Additional Key Words and Phrases: Distance Oracles, Highway Dimension, Provable Guarantees.

1 INTRODUCTION

Motivated by the requirements of modern transportation systems, we consider the fast computation of constrained shortest paths (CSP) in large-scale graphs. Though the basic shortest-path (SP) problem has a long history, it has been revolutionized by recent algorithmic advancements that help enable large-scale mapping applications (cf. [? ?] for surveys). In particular, the use of preprocessing techniques and network augmentation has led to dramatic improvements in the scalability of SP computation for road networks. These techniques however do not extend to the CSP.

The SP and CSP problems can be summarized as follows: Given a graph G where each edge has associated *length* and *cost*, the SP problem requires finding an (s, t)-path of minimum length for any given nodes s and t. The CSP problem inputs an additional budget b, and requires finding a minimum length (s, t)-path *with total cost less than* b. The two problems, though similar, have very different runtime complexity: SP queries admit polynomial-time algorithms (in particular, the famous Dijkstra's algorithm), while CSP computation is known to be NP-Hard [?]. That said, a standard dynamic programming computes CSPs in pseudo-polynomial time for discrete costs, and gives a natural scaling-based FPTAS for continuous costs (as in the knapsack problem).

Though there is a rich literature on CSP (cf. [?]), existing approaches do not scale to support modern applications. To this end, we study *preprocessing and network augmentation for speeding CSP computation.* Our work contributes to a growing field of algorithms for large-scale problems (non-convex methods, sketching techniques, etc.), with poor worst-case performance, but which are provably efficient for practically relevant settings.

Applications of large-scale CSP computation: Our primary motivation for scaling CSP comes from the requirements of modern transportation platforms (Lyft, Uber, Waze etc.) for accurate routing and travel-time estimates. Modern SP engines like Google Maps and OSRM do not make full use of available traffic information. In particular, they do not incorporate uncertainties in travel times, leading to inaccurate estimates in settings with high traffic variability. This can be corrected by computing shortest paths based on *robust* travel-time estimates: given s, t and parameters p, δ, we want to find an (s, t)-path P minimizing $\mathbb{E}[\ell(P)]$, subject to $\mathbb{P}(\ell(P) > \mathbb{E}[\ell(P)] + \delta) \leq p$. Computing this exactly for general distributions is expensive due to the need for computing convolutions of distributions. However, assuming uncorrelated travel-times across different road segments (Comment on conditional indpendence - cite Dawn), Chebyshev's inequality gives us that $\mathbb{P}(\sum_e T_e > \mathbb{E}[\sum_e T_e] + \delta) \leq \sum_e \mathbb{V}\mathrm{ar}(T_e)/\delta^2$. Using this, we can reformulate the robust trip-time estimation problem as $\min_{P \in \mathcal{P}_{s,t}} \sum_{i \in P} \mu_i$ s.t. $\sum_{i \in P} \sigma_i^2 \leq \delta^2 p$, which is now a CSP problem. Note that though we relax the condition $\mathbb{P}(\ell(P) > \mathbb{E}[\ell(P)] + \delta) \leq p$, our solution always respects this constraint – this is often more critical for practical applications, e.g., for ETA estimates accuracy is more important than optimality.

Another problem that can be modeled as a CSP is that of finding *reliable shortest paths*. Consider the case where each edge has a probability q_e of triggering a bad event, with resulting penalty p (for example, slowdowns due to accidents). In this case, we want to minimize the travel time as well as the expected penalty. Aassuming independence, we have the following natural problem: $\min_{P \in \mathcal{P}_{s,t}} \ell(P) + p(1 - \prod_{e \in P}(1 - q_e))$. This model is considered in [?] for routing with fare evasion, where q_e is the probability of encountering an inspector, and p the penalty; the authors suggest using a CSP formulation, wherein the non-linear objective is replaced by a linear constraint by taking logarithms.

Our Contributions. Our work aims to address the following overarching questions:
1. How can we use preprocessing and graph augmentation techniques to speed up CSP computation.
2. How can we give preprocessing/storage/query time guarantees for such techniques.

For the SP problem, there was significant initial development on the first question, leading to many different techniques [?], which worked well in practice, but had no performance guarantees. Abraham et al. [? ?] then proposed the *Highway Dimension* (HD), a graph structural metric, which they showed could parametrize the preprocessing, storage and query time of several of these heuristics. Our work adopts a similar program; in particular, our contributions are as follows:

Theoretical contributions: In sec:prelim, after introducing the problem, we extend the notion of the HD (as defined in [?]) to directed graphs and general path systems. Then, in sec:chd, we define an analogous notion of a *constrained highway dimension* (CHD) for constrained shortest paths. We then propose a construction of hub labels for CSP queries, whose complexity is parameterized by the CHD. We show that, although the CHD can be much bigger than the HD in general, the two can be related under an additional *partial witness condition*. This provides a justification for our proposed CSP hub labels, as it suggests that they should be efficient in settings where SP computation is scalable. Finally, we study average performance and obtain stronger results. In particular, we show how a small average CHD can be explained by physical overpasses in road networks.

Practical contributions: Although our theoretical results justify the use of hub labels for CSP, they are impractical for real networks. In sec:numeric, we show how to adapt our ideas to develop practical hub label constructions for CSP queries. We evaluate our algorithm on datasets with detailed travel-time information for San Francisco and Luxembourg. Our experiments show that our hub labels support query times which are orders of magnitude faster than existing techniques (without preprocessing), have small storage requirements, and good preprocessing times even on a single machine.

Related work. CSP problems have an extensive literature, surveyed in [?]. More recently, there has been significant interest in robust SP problems, as well as the related stochastic on-time arrival (SOTA) problem [?]; recent works have proposed both optimal and approximate policies [? ?].

Existing approaches for these problems, however, do not exploit preprocessing and augmentation techniques, and consequently do not support the latencies required for mapping applications.

As we mention before, our work is inspired by the recent developments in shortest path algorithms [? ? ? ? ? ?]; refer [?] for an excellent survey of these developments. The pre-processing technique we use for speeding CSP computations is hub labels (HL), first introduced for SP computations in [?]. More recently, HL was proved to have the best query-time bounds for SP computation in low HD graphs [? ?] (this was experimentally confirmed in [?], [? , Figure 7]). Finally, the HD-based bounds for hub labels was shown to be tight in [? ?], and it was also shown that finding optimal hub labels is NP hard.

Finally, a related class of problems to CSP is that of shortest paths under label constraints [?], where the aim is to find shortest paths that avoided certain labels (e.g. toll roads, ferries, etc.). [?] proposed preprocessing techniques (in particular, contraction hierarchies) for restricted class of such problems. Note however that label-constrained SP problem is essentially a concatenation of parallel SP problems, and involves only local constraints; in contrast, the CSP involves global constraints on paths, and consequently is known to be NP-hard. Moreover, our results do in fact shed light on why contraction hierarchies work well for label-constrained SP queries.

2 PRELIMINARIES

2.1 Basic Setting

We consider a directed graph $G = (V, E)$ with *length function* $\ell : E \to \mathbb{N}_+$, and *cost function* $c : E \to \mathbb{N}_+\{0\}$. For each node v, we denote its degree $\Delta(v)$ as the sum of the in-degree and out-degree, and define the *maximum degree* $\Delta := \max_v \Delta(v)$. For any source-terminal pair $s, t \in V$, we denote by $\mathcal{P}_{s,t}$ the set of all (s, t)-paths which are simple (i.e., without loops or cycles). Throughout this work, we only consider simple paths, which for brevity we refer to as paths. TODO: Move the following notation For a path Q and two vertices $u, v \in Q$ we denote $Q[u, v] \subseteq Q$ as the sub (u, v)-path. For two paths P, Q, we denote $P|Q$ as the concatenation, which is defined only when P, Q share the correct endpoint.

For any path P, we define its length $\ell(P)$ and cost $c(P)$ as the sum of edge lengths and edge costs in P. Note that any path P with more than one node has length at least 1 (since we assume lengths are integers). For $s, t \in V$, the distance from s to t, denoted $\text{dist}(s, t)$, is the smallest length among all paths $P \in \mathcal{P}_{s,t}$. The distance from a node v to a path P, denoted $\text{dist}(v, P)$, is measured as the minimum distance from v to a node $w \in P$; the distance to v, $\text{dist}(P, v)$, is defined analogously. Note that $\text{dist}(P, v)$ and $\text{dist}(v, P)$ need not be the same as the graph is directed. We denote the shortest (s, t)-path (if it exists) as $P(s, t)$, and denote the set of all shortest paths in G as $\mathcal{P}^*$. Finally, we define $D := \max_{P \in \mathcal{P}^*} \ell(P)$ to be the diameter of G.

Our goal is to develop a data-structure to answer *Constrained Shortest-Path* (CSP) queries: Given a source-terminal pair s, t and a budget b, we want to return a path P solving $\min_{P \in \mathcal{P}_{s,t}} \ell(P)$ subject to $c(P) \leq b$. We define $\text{dist}(s, t|b)$ to be the minimum of this problem. If there is no feasible solution, we define $\text{dist}(s, t|b) = \infty$. Users do not want to be presented solutions which are far away from the optimum, even if it saves them budget. TODO: Move the stretch thing The *stretch* of a path is its comparison to the shortest option. Formally, we have the following desirable property of an algorithm.

Definition 2.1 (Stretch). An algorithm for CSP has stretch $\text{St} \geq 1$ if, for every $s, t \in V$ and $b \leq B$, either outputs $\text{dist}(s, t|b)$ when $\text{dist}(s, t|b) \leq \text{St}\,\text{dist}(s, t)$ or outputs “infeasible” when $\text{dist}(s, t|b) > \text{St}\,\text{dist}(s, t)$.

We will take St as a given parameter or, equivalently, an extra constraint given by the application. Most of our results are independent of the value St. Note that the CSP problem may have multiple solutions as there could be several paths with the same length and cost lower than b. To limit these solutions to those with minimal cost, we require that the path also be *efficient.*

Definition 2.2 (Efficient Path). A path $P \in \mathcal{P}_{s,t}$ is called *efficient* if there is no other path $P' \in \mathcal{P}_{s,t}$ such that $\ell(P') \leq \ell(P)$ and $c(P') \leq c(P)$ with at least one inequality strict.

We denote the set of all efficient paths as $\mathcal{P}^E$, and define the *Pareto frontier* from s to t as $\mathcal{P}_{s,t} \cap \mathcal{P}^E$. Observe that every subpath of an efficient path is also efficient (if not, we could improve the path by replacing the subpath). For $r > 0$ and $v \in V$, we define the *forward and reverse balls of radius* r by $B^+_r(v) := \{u \in V : \text{dist}(v,u) \leq r\}$ and $B^-_r(v) := \{u \in V : \text{dist}(u,v) \leq r\}$, and also define $B_r(v) := B^+_r(v) \cup B^-_r(v)$. Finally, a graph G is said to have a *doubling dimension* α if, for any node v and any $r > 0$, the ball $B_{2r}(v)$ can be covered by at most α balls of radius r.

2.2 Hitting sets and the highway dimension

We now define some additional network primitives, which we need to parametrize the performance of our CSP algorithms. In particular, we use the *highway dimension*, introduced by **? ?**] to parametrize shortest-path computations in undirected graphs. A few of our results are generalizations of those in [?]; the technical challenges of these extensions may not be clear to a non-expert reader, thus we differ all discussions on the matter to [?]. Very broadly speaking, [?] deals only with undirected graphs and shortest paths, whereas our approach covers directed graphs and general sets of paths.

We define a *path system* $\mathcal{Q}$ as any collection of paths. We say that a set $C \subseteq V$ *hits* any given path Q if some node in Q belongs to C. Moreover, we say that C is a *hitting set for a path system* $\mathcal{Q}$ if it hits every $Q \in \mathcal{Q}$. For any $r > 0$, we say a path Q is r-significant if $\ell(Q) > r$. For a given path system $\mathcal{Q}$, we denote $\mathcal{Q}_r$ as the set of all r-significant paths in $\mathcal{Q}$. Hitting sets are useful for compressing path systems. In particular, even if the hitting set is large, the extent to which a path system can be compressed depends on the *local sparsity* of hitting sets with respect to *significant paths* of $\mathcal{Q}$.

Definition 2.3 (Locally-Sparse Hitting Sets). Given a path system $\mathcal{Q}$ and $r > 0$, an (h,r) locally-sparse hitting set (or (h,r)-LSHS) is a set $C \subseteq V$ with two properties:

(1) Hitting: C is a hitting set for $\mathcal{Q}_r$.
(2) Local sparsity: for every $v \in V$, $|B_{2r}(v) \cap C| \leq h$.

As we discuss in ssec:hldef, the existence of (h,r)-LSHS immediately enables the compression of path system $\mathcal{Q}$ via the construction of *hub labels.* However, the existence of LSHS does not guarantee the ability to efficiently compute these objects. To address this we need a stronger notion, the *highway dimension* is a property that ensures both existence and efficient computation of LSHS. To define the highway dimension (HD), we first need two additional definitions: for $v \in V, r > 0$, we define the *forward path-neighbourhood* with respect to a path system $\mathcal{Q}$ as $S^+_r(v,\mathcal{Q}) := \{Q \in \mathcal{Q}_r : \text{dist}(v,Q) \leq 2r\}$ and similarly $S^-_r(v,\mathcal{Q}) := \{Q \in \mathcal{Q}_r : \text{dist}(Q,v) \leq 2r\}$ is the reverse neighbourhood. As before, we have $S_r(v,\mathcal{Q}) := S^+_r(v,\mathcal{Q}) \cup S^-_r(v,\mathcal{Q})$. Now we can define the HD of a path system $\mathcal{Q}$. Essentially, the HD re-orders the sequence of qualifiers in the definition of (h,r)-LSHS: it requires the existence of a small hitting set for each individual neighborhood, rather than a single hitting set which is locally sparse.

Definition 2.4 (Highway Dimension). A path system $\mathcal{Q}$ has HD h if, $\forall r > 0, v \in V$, there exists a set $H_{v,r} \subseteq V$ such that $|H_{v,r}| \leq h$ and $H_{v,r}$ is a hitting set for $S_r(v,\mathcal{Q})$.

As shorthand, we refer to the HD of (G, ℓ) as that of $\mathcal{P}^*$. Note that $HD \leq h$ is a more stringent requirement than the existence of an (h, r)-LSHS C, since $C \cap B_{2r}(v)$ need not hit all the paths in $S_r(v, \mathcal{Q})$. However, if G has $HD \leq h$, then this guarantees the existence of a (h, r)-LSHS according to the following result.

Proposition 2.5. *If the path system $\mathcal{Q}$ has HD h, then, $\forall\, r > 0$, there exists an (h, r)-LSHS.*

This fact follows by adapting the proof from [? , Theorem 4.2] to our general case, for details see [?]. More importantly, note that the result is about existence and does not touch on computability. As we discuss in Section 3.2.1, if G has $HD \leq h$, then this permits efficient computation of LSHS.

2.3 Shortest-Paths via Hub Labels

Two of the most successful data-structures enabling fast shortest path queries at scale are *contraction hierarchies* (CH) [?] and *hub labels* (HL) [?]. These are general techniques which always guarantee correct SP computation, but have no uniform storage/query-time bounds for all graphs. We now study HL only; for the construction and results of CH refer to [?].

The basic HL technique for SP computations is as follows: Every node v is associated with a hub label $L(v) = \{L^+(v), L^-(v)\}$, comprising of a set of forward hubs $L^+(v)$ and reverse hubs $L^-(v)$. We also store $\mathrm{dist}(v, w)\ \forall\, w \in L^+(v)$ and $\mathrm{dist}(u, v)\ \forall\, u \in L^-(v)$. The hub labels are said to satisfy the *cover property* if, for any $s \neq t \in V$, $L^+(s) \cap L^-(t)$ contains at least one node in $P(s, t)$. In the case that t is not reachable from s, it must be that $L^+(s) \cap L^-(t) = \varnothing$.

With the aid of the cover property, we can obtain $\mathrm{dist}(s, t)$ by searching for the minimum value of $\mathrm{dist}(s, w) + \mathrm{dist}(w, t)$ over all nodes $w \in L^+(s) \cap L^-(t)$. If the hubs are sorted by ID, this can be done in time $O(|L^+(s)| + |L^-(t)|)$ via a single sweep. Moreover, by storing the second node in $P(s, w)$ for each $w \in L^+(s)$, and the penultimate node in $P(w, t)$ for each $w \in L^-(t)$, we can also recover the shortest path recursively, as each HL query returns at least one new node $w \in P(s, t)$. Note that we need to store this extra information, otherwise we could have $L^+(s) \cap L^-(t) = \{s\}$. Let $L_{\max} := \max_v |L^+(v)| + \max_v |L^-(v)|$ be the size of the maximum HL. The per-node storage requirement is $O(L_{\max})$, while the query time is $O(L_{\max} \ell(P(s, t)))$.

Although hub labels always exist (in particular, we can always choose $L^+(s)$ to be the set of nodes reachable from v, and $L^-(s)$ the set of nodes that can reach v), finding *optimal* hub-labels (in terms of storage/query-time bounds) is known to be NP-hard [?]. To construct hub labels with guarantees on preprocessing time and $L_{\max}$, we need the additional notion of a *multi-scale LSHS*. We assume that graph (G, ℓ) admits a collection of sets $\{C_i : i = 1, \ldots, \log D\}$, such that each C_i is an $(h, 2^{i-1})$-LSHS. Given such a collection, we can now obtain small HL. We outline this construction for directed graphs, closely following the construction in [? , Theorem 5.1] for the undirected case.

Proposition 2.6. *proposition]theo:construct$_h$lFor(G,ℓ), given a multi-scale LSHS collection $\{C_i : i = 0, \ldots, \log$ where each C_i is an $(h, 2^{i-1})$-LSHS, we can construct hub labels of size at most $h(1 + \log D)$.*

Proof. For each node v, we define the hub label $L(v)$ as

$$L^+(v) := \bigcup_{i=0}^{\log D} C_i \cap B^+_{2^i}(v) \quad \text{and} \quad L^-(v) := \bigcup_{i=0}^{\log D} C_i \cap B^-_{2^i}(v).$$

Since each C_i is an $(h, 2^{i-1})$-LSHS which we intersect with balls of radius $2 \cdot 2^{i-1}$, every set in the union contributes at most h elements and the maximum size is as claimed.

To prove the cover property, we note that, if t is not reachable from s, by definition $L^+(s) \cap L^-(t) = \varnothing$. This is because the elements in $L^+(s)$ are reachable from s and the elements in $L^-(t)$ reach t. On the other hand, when $P(s, t)$ exists, let i be such that $2^{i-1} < \ell(P(s, t)) \leq 2^i$. Now any point in the path belongs to both $B^+_{2^i}(s)$ and $B^-_{2^i}(t)$, and hence $C_i \cap P(s,t)$ (which is not empty since C_i hits all SP of length $\geq 2^{i-1}$) is then in both hubs, which shows the result. □

Finally, we need to compute the desired multi-scale LSHS in polynomial time. In sec:preproc we show that, if the HD is h, in polynomial time we can obtain sparsity $h' = O(h\Delta \log(h\Delta))$. In other words, the HL have size $h'(1 + \log D)$ instead of $h(1 + \log D)$ if we are not given the multi-scale LSHS and have to compute them in polynomial time. A more subtle point is that the resulting algorithm, even though polynomial, is impractical for large networks. In sec:numeric, we discuss heuristics that work better in practice.

3 SCALABLE CSP ALGORITHMS: THEORETICAL GUARANTEES

We now turn to the problem of constructing hub labels for constrained shortest-paths queries and, in particular, for computing efficient paths in G. We want to develop a data-structure that supports fast queries for *efficient paths*.

In Section 2.3 we discussed that, if a graph G has HD $\leq h$, we can simultaneously bound, as functions of h, the preprocessing time, storage requirements and query time for constructing HL. This suggests that for the construction of provably efficient HL for the CSP problem, one needs to define an analogous property for the set of *efficient paths*.

Definition 3.1 (Constrained Highway Dimension). The constrained highway dimension (CHD) of (G, ℓ, c), denoted h_c, is the HD of the efficient-path system $\mathcal{P}^E$.

Note that every shortest path is efficient, thus $h_c \geq h$, where h is the HD of G under shortest paths.

We now have two main issues with this definition: first, it is unclear how this can be used to get hub labels, and second, it is unclear how the corresponding hub labels compare with those for shortest-path computations. To address this, we first convert efficient paths in G to shortest paths in a larger *augmented graph*. In Section 3.2, we use this to construct hub labels for CSP queries whose storage and query complexity can be bounded as Bh_c (which can be strengthened further to $g(b)h_c$, where $g(b)$ measures the size of the Pareto frontier, cf. Section ??.). Finally, in Section 3.3, we show that the hub labels for CSP queries can in fact be related to the hub labels for SP queries under an additional natural condition on the efficient paths.

3.1 Augmented Graph

In order to link the constrained highway dimension to hub labels, we first convert the original graph G (with length and cost functions) into an *augmented graph* G^B with only edge lengths, such that the *efficient paths of* G *are in bijection with the shortest paths of* G^B. We achieve this as follows: Each node in G^B is of the form $\langle v, b\rangle$, which encodes the information of the remaining budget $b \geq 0$ and location $v \in V$. A node is connected to neighbors (according to E) as long as the remaining budget of that transition is non-negative. Finally, we create n sink nodes, denoted v^-, and connect node $\langle v, b\rangle$ to v^- with length $1/(b+1)$. An illustration of the construction is presented in Figure 1. Paths in G^B are mapped to paths in G in the intuitive way, by removing the budget labels and sink nodes. We call this mapping the *projection* of a path. If we use that in G^B paths consuming more budget are penalised, we can prove the following.

PROPOSITION 3.2. *A shortest path from source* $\langle s, b\rangle$ *to sink node* t^- *projects to an efficient path in* G *solving* $dist(s, t|b)$.

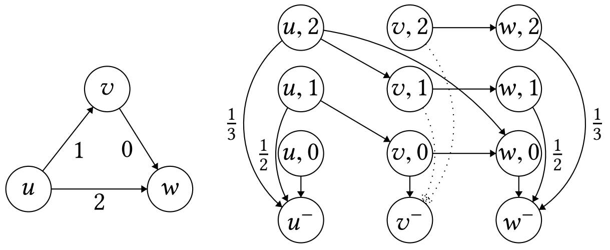

Fig. 1. Example of a graph augmentation: The original graph G has all paths of unit length, and costs as labeled on the edges. In the augmented graph G^B, the labels represent the edge lengths (unlabeled edges have length 1). Note the additional edges from (w, b) to the sink node w^- (and similarly for u and v).

Latter we give a construction that depends on LSHS for the system $\mathcal{P}^E$, we call such object an efficient path hitting set (EPHS). The next result allows us to relate EPHS of the original graph to LSHS in the augmented graph. Note that in G^B we are interested only in shortest paths ending in sink nodes (since these map to efficient paths). Let $\mathcal{P}^B$ be the path system comprising all shortest paths in G^B ending in a sink node. A hitting set for $\mathcal{P}^E$ can be used to obtain a hitting set for $\mathcal{P}^B$, but, since the augmented graph has more information, the sparsity increases. Surprisingly, in this case we can also relate the HDs of the path systems $\mathcal{P}^E$ and $\mathcal{P}^B$. Note that this does not follow from prop:lshsaug, since the HD is a stronger notion than existence of locally-sparse hitting sets.

Proposition 3.3. *proposition]prop:lshsaug Given a (h_c, r)-EPHS for the path system $\mathcal{P}^E$, in polynomial time we can construct a $(h_c B, r)$-LSHS for $\mathcal{P}^B$.*

Proposition 3.4. *proposition]prop:HDaugmented If the HD of the system $\mathcal{P}^E$ is h_c, then the HD of the system $\mathcal{P}^B$ is Bh_c.*

3.2 Hub Labels For CSP

We present a construction is similar to HL for shortest-paths (cf. ssec:hldef). A subtle difference is that we are only interested in paths ending in a sink node. Each node $\langle v, b\rangle$ has a forward hub label $L^+(\langle v, b\rangle) \subseteq V^B$, and *only sink nodes* u^- have a reverse hub $L^-(u^-) \subseteq V^B$. The cover property must be satisfied for every $\langle s, b\rangle$ and t^-. Finally, if we want to reconstruct the path, we can proceed similarly as in ssec:hldef; we can augment the hub labels with the next-hop node, and compute the entire path recursively. Putting things together, we can construct hub labels for answering CSP queries, such that their preprocessing time and storage is parameterized by the CHD h_c. For the proof see [**?**].

Theorem 3.5. *For a network (G, ℓ, c), given a multi-scale EPHS $\{C_i : i = 0, 1, \dots, \log D\}$, where C_i is an $(h_c, 2^{i-1})$-EPHS, we can construct hub labels to answer queries for s, t, b in time $O((b+1)h_c \log D)$. The total space requirement is $O(nB \cdot Bh_c \log D)$.*

3.2.1 Preprocessing. Computing hitting sets is difficult in general, but it becomes tractable when the underlying set has small VC-dimension [**?**]. Let Q be any path system. We can map Q to a set system and show that it has VC-dimension 2, hence we can compute approximate LSHS in polynomial time. Formally, we obtain the following result (for the details see [**?**]).

Proposition 3.6. *proposition]prop:poly$_l$shsIfapathsystemQhasHDh, then, foranyr>0, wecanobtaininpolyno*

3.2.2 Using the size of the Pareto Frontier. Alberto: Do we ommit this part? We could keep it if we have the space The linear dependence on B in the bound on HL sizes (cf. theo:HLeff) is

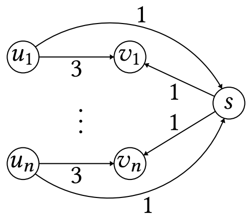

Fig. 2. Graph with small HD but large CHD: The graph comprises $2n + 1$ nodes, with the edge labels representing the lengths. Note that the shortest paths in the graph are of the form sv_i, u_is and u_isv_j (for all combinations i, j). Thus, the HD is 1 as $H_{v,r} = \{s\}$ is a hitting set for all these paths. On the other hand, if we have costs such that $c(u_iv_i) = 0 \,\forall\, i$, while all other edges have cost 1, then we have n parallel efficient paths u_iv_i, which must all be hit by any EPHS.

somewhat weak. Essentially, this corresponds to a worst-case setting where the efficient paths between any pair of nodes is different for each budget level. In most practical settings, changing the budget does not change the paths too much, and ideally the hub label sizes should reflect this fact. This is achieved via a more careful construction of hub labels, resulting in the following bound.

Theorem 3.7. *Let (G, ℓ, c) as in theo:HLeff and $g : \mathbb{N} \to \mathbb{N}$ be such that, for every $s, t \in V$, $b \in \mathbb{N}$, $|\{P \in \mathcal{P}^E_{s,t} : c(P) \leq b\}| \leq g(b)$. Then, we can construct hub labels of size $O(g(B)h_c \log D)$, and answer queries with budget b in time $O(g(b)h_c \log D)$.*

Note that there always exists such a function g and the worst case is $g(b) = b$. The proof depends on a different technique for constructing HL (refer to alg:forwardhub,alg:reversehub in sec:proofs). The main idea is to sort the efficient paths for each source node s by cost, and then carefully mark nodes when they are added to forward HL; these marked nodes are then used to construct the reverse HL. For brevity, the complete algorithmic details and proof are deferred to Appendix D.

3.3 Comparing HD and CHD

The previous sections show that we can construct hub labels for solving CSP whose preprocessing time, storage and query time can all be parameterized in terms of the constrained highway dimension h_c. This, however, still does not give a sense of how much worse the hub labels for the CSP problem can be in comparison with those for finding shortest paths. We now try to understand this question. Comparing the size of the *optimal* hub labels for SP and CSP is infeasible as even finding the optimal hub labels for SP is NP-hard [?]. However, since we can parametrize the complexity of HL construction for SP in terms of the HD, a natural question is whether graphs with small HD also have a small CHD. Note that the answer to this depends on both the graph and the costs. We now show that the CHD and, moreover, the sparsity of any EPHS, can be *arbitrarily worse* than the HD.

Proposition 3.8. *proposition]prop:example$_b$ad$_c$hdTherearenetworkswithHD1wheretheCHD, andalsothesp*

Proof. Consider the directed graph G defined in Figure 2. It is easy to see that $H_{v,r} = \{s\}$ is a shortest-path hitting set for every $r > 0$ and $v \in V(G)$; hence the HD is 1. On the other hand, suppose the costs are such that $c(u_iv_i) = 0$ for every i, while all other costs are set to 1. Note that the 1-significant efficent paths intersecting the ball $B_s(2)$ are u_iv_i, which are all disjoint. Therefore, the hitting set $H_{s,1}$ must contain at least n elements. Moreover, the same argument shows that the sparsity of the best LSHS for $\mathcal{P}^*$ is 1, whereas the sparsity of *any* EPHS is also lower bounded by n. □

Remark 3.9. *One criticism of the graph in fig:big$_c$hdisthatithasamaximumdegreeof n.However, theresulthold*
Intuitively, the separation between HD and CHD (or more particularly, the hub labels for computing SPs and CSPs) occurs due to the fact that, for arbitrary graphs and cost functions, the shortest and efficient paths may be completely unrelated. For real-world networks, however, this appears unlikely. In particular, intuition suggests that efficient paths largely comprise of segments which are in fact shortest-paths in their own right. This notion can be formalized via the following definition of a *partial witness*

Definition 3.10 (Partial Witness). Let $\beta \geq 0$. We say that a path system $\mathcal{Q}$ is β-witnessed by the path system $\mathcal{Q}'$ if, for every $Q \in \mathcal{Q}$, $\exists Q' \in \mathcal{Q}'$ such that $Q' \subseteq Q$ and $\ell(Q') \geq 2^{-\beta}\ell(Q)$.

We can now ask if the hub labels for computing SPs and CSPs can be related in settings where the shortest-path system $\mathcal{P}^*$ is a partial witness for the efficient path system $\mathcal{P}^E$. At an intuitive level, the partial witness property says that efficient and shortest paths are not completely different, i.e., if Q is efficient, a fraction $2^{-\beta}$ of Q is a shortest path. As a consequence, a node hitting numerous paths in $\mathcal{P}^*$, should also hit many paths in $\mathcal{P}^E$. Note also that asking for the witness property to hold for all lengths is too extreme, as this essentially requires that all single-hop paths with 0 costs are shortest paths. Thus, we want this property only for 'long-enough' paths.

We now show that if, for some β, the network indeed has the partial witness property for paths longer than some r_β, then we can relate the HL sizes for the two problems in terms of β and the doubling dimension α. Note that the doubling dimension depends on G and ℓ; the partial witness property depends on the interplay between G, c and ℓ. Observe also that, if α is a constant, then the requirement in theo:witness$_d$oublingisforpathslongerthanr$_\beta \sim h\alpha^{\beta-2}$.

Theorem 3.11. *Assume that G is α-doubling and $\mathcal{P}^E_r$ is β-witnessed by $\mathcal{P}^*$ for every $r \geq r_\beta$, where $r_\beta = 2^{\log_\alpha(\alpha^{\beta-2}h)}$. Then, for any $r > 0$, given an (h,r)-LSHS, we can construct, in polynomial time, an $(\alpha^\beta h, r)$-EPHS for (G, ℓ, c).*

Proof. For any r, we need to construct a hitting set C^E for $\mathcal{P}^E_r$. Assume first $r \geq r_\beta$. Take C as the hitting set for $\mathcal{P}^*_{2^{-\beta}r}$, which is guaranteed to be sparse with respect to balls of radius $2^{-\beta+1}r$. Define the desired set by

$$C^E := \{v \in C : v \text{ is in some } r\text{-efficient path}\}.$$

Since $\mathcal{P}^*$ is a $2^{-\beta}$-witness for $\mathcal{P}^E_r$, C^E is indeed a hitting set for $\mathcal{P}^E_r$. We are only left to prove the sparsity. Take some $u \in V$, by doubling dimension we can cover $B^+_{2r}(u)$ by at most α^β balls of radius $2^{-\beta+1}r$. Each of these balls contains at most h elements of C, therefore the sparsity is as claimed. The argument for reverse balls is identical.

Now we analyse the case $r < r_\beta$. It is no longer true that efficient paths are witnessed, but now the neigborhoods are small. We first claim that, for any $v \in V$ and $r > 0$, $|B_r(v)| \leq \alpha^{\log_2 r+1}$. Indeed, using the doubling property $\log_2 r + 1$ times, we can cover $B_r(v)$ with balls of radii $1/2$. Since the minimum edge length is 1, all of these balls must be singletons and the claim follows. Now we can take $C = V$ as the EPHS. Clearly C hits all the paths and the local sparsity is at most the size of the ball. Using our assumption on r_β, a simple computation shows that $|B_{2r}(v)| \leq \alpha^{\log_2 r_\beta+2} \leq h\alpha^\beta$. □

Remark 3.12. *The existence of a β-witness is not enough to bound the CHD. Nevertheless, as we discuss in Section 2.3, the existence of (h,r)-LSHS already allows the construction of HL.*

4 SCALABLE CSP ALGORITHMS: PRACTICAL IMPLEMENTATION

We start by defining a similar structure as the augmented graph, but more amenable to practical purposes. The graph G^B defined in Section 3.1 may contain a lot of redundant information, hence it is not a minimal representation. For example, the same efficient path uv can be repeated many times in the form $\langle u,1\rangle\langle v,0\rangle$, $\langle u,2\rangle\langle v,1\rangle$ and so on. By encoding this information more efficiently, we can reduce the size of the augmented graph and obtain considerable improvements both in query time and in data storage.

We construct an augmented graph without this duplicated information as follows. The nodes are, as before, pairs $\langle v,b\rangle$, but an edge $\langle v,b\rangle,\langle v',b'\rangle$ is placed only if it is essential for some efficient path, i.e., removing said edge changes the correctness of a query. If we let $\mathcal{P}^E_{s,t}$ be the set of all efficient paths from s to t, we take every $P\in\mathcal{P}^E_{s,t}$ with cost $b\le B$ and trace it in the augmented graph. The tracing is such that the augmented path is forced to end at $\langle t,0\rangle$. In other words, all efficient paths $P\in\mathcal{P}^E_{s,t}$ can be represented by a set a set of efficient paths ending at $\langle t,0\rangle$. For details see [?]. Note that in $\tilde{G}^B$ there are no sink nodes, hence it has at least n nodes and nB arcs fewer compared to G^B. In the worst case, those nB arcs are the only gain by doing this process. Nevertheless, in our experiments $\tilde{G}^B$ is up to 60% smaller than G^B.

Frontier and specific queries. We test our algorithms with two different tasks. Recall that the preprocessing is done for some fixed maximum budget B. The first task is a frontier query, wherein given s and t, we return the lengths of all efficient paths with costs $b=0,1,\dots,B$. The second is called a specific query, for given s,t,b we return $\text{dist}(s,t|b)$, i.e., only one efficient path.

HL construction via Contraction Hierarchies. We use some techniques described in [?] combined with an approach tailored for augmented graphs. The main idea is to first use contraction hierarchies to get a node ranking and then define the forward hubs of v as the nodes visited during a contraction-based forward search starting at v. The reverse hub is defined analogously. These are valid hubs, since the highest rank node in a path is guaranteed to be in both hubs.

The most important parameter in CH is the rank; results vary greatly from one choice of ranks to another. We obtain a rank in G by running a greedy shortest-path cover, defined as follows. Start with a cover $C=\varnothing$ and compute the set of all shortest paths $\mathcal{P}^*$. Take a node $v\notin C$ hitting most paths in $\mathcal{P}^*$, then remove all those paths from $\mathcal{P}^*$, add v to C and iterate. The rank is defined as n for the first node added to C, $n-1$ for the second and so on. To implement the SP cover we follow the algorithm in [?]. We stress that this is just an easy way to get a ranking; more sophisticated heuristics that decide on-line the next node to contract usually work well in practice [? ?]. Using another contraction scheme may expedite our algorithms and reduce the hub size.

For some instances we work with G^B and for others with $\tilde{G}^B$. Even though $\tilde{G}^B$ takes some time to compute, it can speed up the overall process and yield considerably better hubs. Recall that in $\tilde{G}^B$ there are no sink-nodes nor replicated information, since efficient paths are stored just once. Given a rank for nodes in G, we contract $\tilde{G}^B$ as follows. Say that V is ordered according to the rank, so node 1 is the least important and n the most important. In $\tilde{G}^B$, we first contract the nodes $\langle 1,b\rangle$ for $b=B,\dots,0$, then the nodes $\langle 2,b\rangle$ and so on till the nodes $\langle n,b\rangle$ are the last to contract. Finally, when contracting a node v, if u is a predecessor and w a successor of v, one should add the short-cut uw only, if by removing v, the distance from u to w is altered. We can go even further; the short-cut uw is unnecessary if the shortest path is not efficient, even if the distance changes.

To obtain better hubs we prune the results obtained by contraction-based searches. If w is in the forward search of v with distance d, it might be that $\text{dist}(v,w)<d$, this occurs because the search goes only to higher rank nodes and the discovered path is missing some node. When $\text{dist}(v,w)<d$,

B	Prepro [m]	Avg F Size	Avg B Size	Query Dij [ms]	Query HL [ms]
0	1	23	22	10.71	0.005
5-f	5	16	28	80.10	0.02
5-s	5	57	28	31.75	0.01
10-f	9	9	28	168.11	0.03
10-s	10	68	28	56.19	0.01
15-f	12	6	28	237.64	0.03
15-s	16	73	28	77.59	0.01
20-f	17	5	28	342.47	0.03
20-s	20	77	28	100.26	0.01
25-f	22	4	28	460.95	0.03
25-s	25	80	28	126.52	0.01
30-f	26	3	28	569.13	0.03
30-s	31	84	28	152.75	0.01

B	Prepro [m]	Avg F Size	Avg B Size	Query Dij [ms]	Query HL [ms]
0	6	18	18	16.35	0.004
5-f	1	18.0	18.0	175.04	0.02
5-s	21	42.9	18.7	72.03	0.01
10-f	2	18.0	18.0	361.15	0.04
10-s	35	49.3	18.7	102.52	0.01
15-f	3	18.0	18.0	577.89	0.06
15-s	46	53.3	18.7	140.80	0.01
20-f	4	18.0	18.0	821.94	0.07
20-s	60	56.5	18.7	183.11	0.01
25-f	5	18.0	18.0	974.84	0.09
25-s	77	59.5	18.7	227.17	0.01
30-f	7	18	18	1247.72	0.10
30-s	93	62.3	18.7	272.41	0.01

Table 1. Experimental results for San Francisco (left) and Luxembourg City (right). Query times are measured with 1000 random s, t pairs for each network and multiple maximum budget levels B. Results on rows $B - f$ correspond to computing the solution frontier for all budget levels $b \leq B$ while the results on rows $B - s$ correspond to computing the solution for the budget level b.

we can safely remove w from the hub of v, since the highest ranked node in a shortest path will have the correct distance. For frontier queries, we can also prune away a node w if the (v, w)-path has a surplus of budget. The entire process can be summarized in the following steps.

(1) Compute the shortest paths in G and use a greedy approach to obtain a cover C
(2) Compute the pruned augmented graph $\tilde{G}^B$
(3) Contract $\tilde{G}^B$ using the rank induced by C
(4) Create hubs $L^+(v), L^-(v)$ using CH
(5) Prune the hubs by running HL queries between v and nodes in $L^+(v)$. Run a similar process for $L^-(v)$.

Recall that, for some instances, we skip step 2 and contract G^B instead. Note that in the last step we bootstrap HL to improve it. This works because the fact that some nodes have incorrect distance labels does not impact the correctness of a HL query; a node minimizing the distance is returned and such node must have a correct label.

Remark 4.1. *For the first step, a direct approach requires storing all the shortest paths in memory. In many cases, this can impose a prohibitive memory requirement. There are two ways around this issue. We can compute the cover by processing one shortest path tree at a time, this translates to $\Omega(n^3 \log n)$ running time, but $O(n)$ memory. The second approach we suggest is to compute only $k << n$ shortest path trees and cover these greedily. We use an off-the-shelf clustering method to obtain k cluster centers, from which we compute the shortest paths. As shown in Figure 3, the clustering approach provides a very good approximation of the hubs with even a small k.*

4.1 Experiments

All our experiments were performed on a 64-bit desktop computer with a 3.40GHz Intel Core i7-6700 processor and 16GB RAM running Ubuntu 16.04. Our code is written in Python 2.7. We use the library Networkx for the graph representation and Dijkstra's algorithm. Although all the steps can be parallelized, we did not implement this. The code is available at github.com/albvera/HHL_CSP.

We evaluated the performance of our algorithms with real-world test networks: a downtown San Francisco network with 2139 nodes and 5697 edges for which real-world travel-time data was

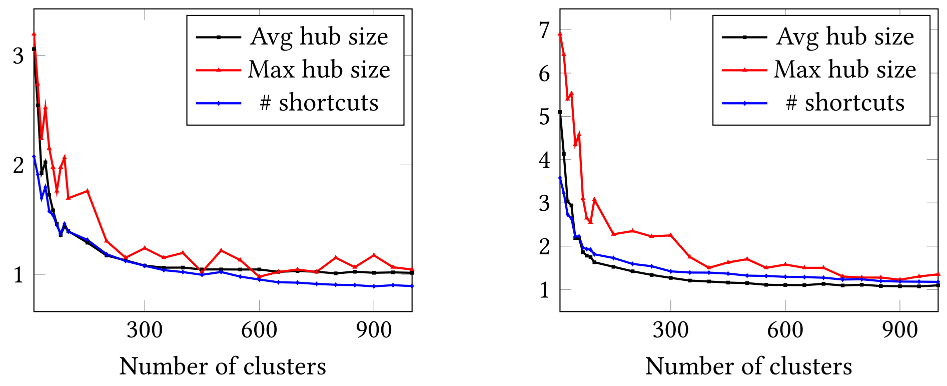

Fig. 3. Performance of clustering for San Francisco (left) and Luxembourg (right). In the y-axis the quantities are normalized by the best greedy cover, i.e., using $k = n$. Note that, while the number of short-cuts is an indicator of performance, it is not perfectly correlated with the hub size.

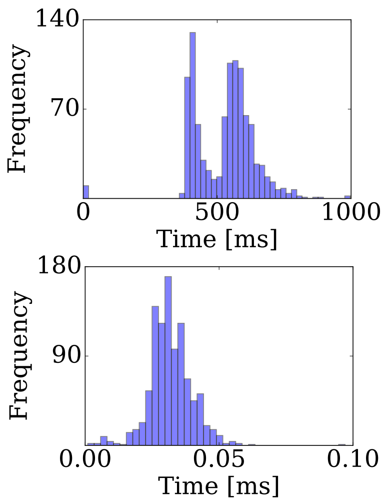

Fig. 4. Histogram frontier queries for Dijkstra (top) and HL (bottom). Times are 1000 random pairs in San Francisco augmented with $B = 25$.

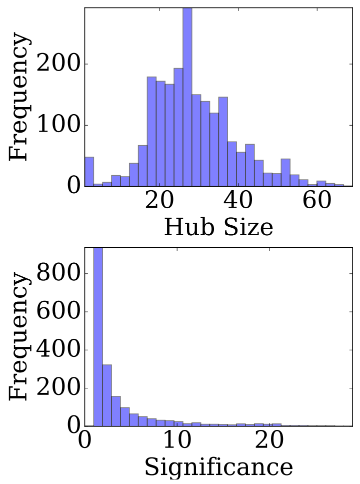

Fig. 5. Size of reverse hubs (top) and significance (bottom) for frontier queries in San Francisco augmented with $B = 25$.

available as a Gaussian mixture model [?], and a Luxembourg City network with 4026 nodes and 9282 edges for which travel-time distributions were synthesized from speed limits [?], as real-world data was unavailable.

In our experiments, we use the mean travel times for our length function and the following cost structure; the top 10% of edges with the highest variance are assigned cost 1 and the rest cost 0. This is a measure of risk, since edges with high variance are prone to cause a delay in the travel time.

4.1.1 Query-time performance. Table 1 presents the CSP computation times for different maximum budgets B. For frontier queries, labelled as 'f', the query times are measured as the average of 1000 random s, t. For specific queries, labelled as 's', the times are measured as 1000 random

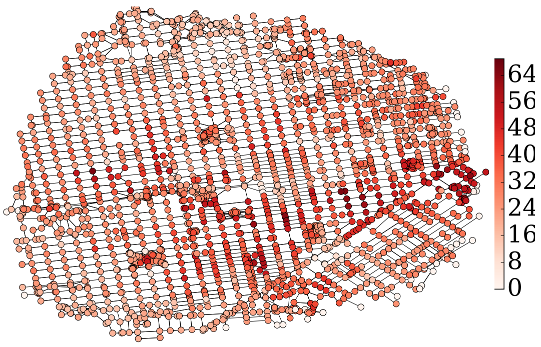

Fig. 6. Heat map of hub size for frontier queries in San Francisco augmented with $B = 25$. Note that the size is not homogeneous, but rather we can observe clusters and neighborhoods tend to be similar.

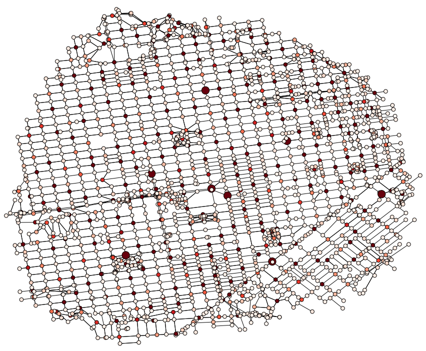

Fig. 7. Heat map of significance for frontier queries in San Francisco augmented with $B = 25$. The most significant nodes have been drawn bigger. The top 3 most significant correspond to Geary Blvd & Gough St, Franklin St & O'Farrel St and Market St & Polk St.

triplets s, t, b. The column for $B = 0$ represents the original graph (without augmentation). As can be seen in the experimental results, our method finds the constrained shortest path solution on average four orders of magnitude faster than running Dijkstra's algorithm on the augmented graph. Preprocessing for frontier queries results in a more compact set of hub labels, since a node $\langle s, b \rangle$ needs to store information for paths with budget exactly equal to b (in case the path is efficient, otherwise it is not stored). On the other hand, for specific queries, $\langle s, b \rangle$ needs to store information for all budgets up to b. The preprocessing time does not include the cover computation, since this is a flat cost of at most the time to preprocess the instance $B = 0$.

Note that preprocessing frontier queries in Luxembourg is faster, despite the network being bigger, this can be explained by the structural properties. For example, in Luxembourg there are more highways and fast roads. Observe that the *average hub size decreases* in San Francisco, this is because in this instance we use $\tilde{G}$, which prunes away most of the nodes, thus many nodes $\langle v, b \rangle$ are isolated and have empty hubs. The longer preprocessing time for frontier queries can be explained as follows. There are many cases when two nodes are not reachable, to detect this requires Dijkstra to explore the entire graph. In contrast, for specific queries we add extra edges $\langle s, b \rangle \langle s, b - 1 \rangle$, hence a reachability test ends, in average, earlier. In the contraction step, we want to remove a node without altering the shortest path, a process that requires many reachability tests.

4.1.2 Hub sizes and node significance. We focus the analysis on two meaningful quantities. The first is hub size, which is well captured by $|L^{-}(\langle t, 0 \rangle)|$ for $t \in V$. Indeed, for frontier queries the reverse hub is bounding the space requirements; for specific queries the same is true up to a constant factor. For the second quantity, we define the significance of $s \in V$ as the number of hubs containing s, i.e., $\sum_t \sum_b \mathbb{1}_{\{\langle s, b \rangle \in L^{-}(\langle t, 0 \rangle)\}}$. Intuitively, a node is highly significant if belongs to many efficient paths. Figure 5 shows a histogram of these metrics.

Figures 6 and 7 show the spatial relationships between the hub size and significance in the San Francisco network. We observe that highly significant nodes tend to have small hub size. The intuition is simple, if most hubs contain s, then is easier for s to satisfy the cover property with a small hub. Note also that the hub size resembles an harmonic function; nodes are mostly similar to their neighbors.

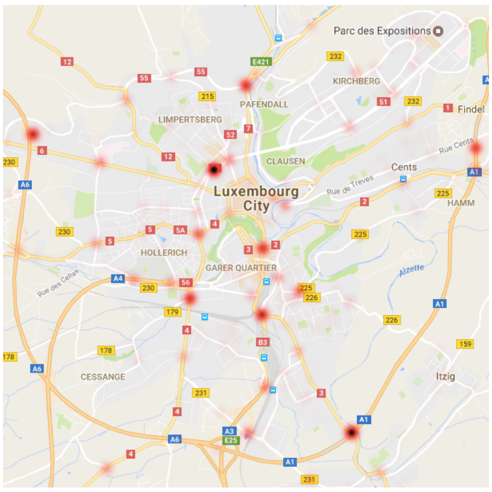

Fig. 8. Heat map of significance for frontier queries in Luxembourg City augmented with $B = 25$. Notice how the highly significant nodes are in main road crossings.

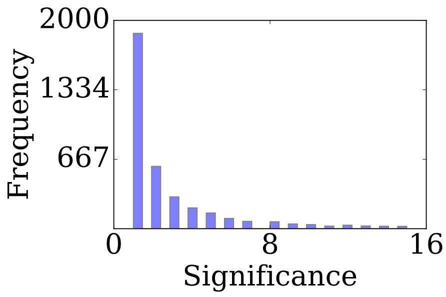

Fig. 9. Histogram of significance in Luxembourg City augmented with $B = 25$. The top 10% most significant nodes were out of scaled and have been collapsed for exposition purposes.

ACKNOWLEDGEMENTS.

The authors would like to thank Moritz Kobitzsch for sharing sanitized versions of the San Francisco and Luxembourg road networks.

A DIFFERENT NOTIONS OF HD

Alberto: Review this, I just moved stuff here and haven't made it readable.

Finally, we briefly note that Definition 2.4 modifies the original definition of [?] in two ways: (i) we consider a less restrictive definition of path neighborhoods, that is appropriate for our needs, and (ii) we generalize the notion to directed graphs and general path systems. For $\mathcal{P}^*$, a consequence of considering less-restrictive path-neighborhoods is that the highway dimension returned by our definition is smaller than that of [?]. In particular, unlike [?], the HD of G as per our definition is not an upper bound to the maximum degree Δ or the doubling dimension α.

About theo:witness$_d oubling : Moreover, theaboveargumentdoesindeedgiveaboundforaweakerdefinitiono$

Remark A.1. *The algorithm in theo:preproc$_a vgmakesonecalltotheVC-dimensionsolverforeachC_i$. On the other hand, the algorithm in [?] calls up to n times the solver for each C_i. Finally, there is an extra* $\log n$ *factor in the approximation guarantee, but now the value of h can be much smaller.*

The definition of HD that we use is tailored for HL note, however, that it is the ones used in [?]. We now discuss how our results extend to the stronger definition.

The stronger version defines a path P to be r-significant if, by adding at most one hop at each end, we get a shortest path P' longer than r. The path P' is called an r-witness for P. Intuitively, a path is significant if it represents a long path. Observe that, if $P \in \mathcal{P}^*$ is such that $\ell(P) > r$, then P is r-significant by definition. We remark also that a path can have many r-witnesses.

Finally, the path neighborhood must also be strengthened. The path $P \in \mathcal{P}^*$ belongs to $S_r^+(v)$ if, P has some r-witness P' such that $\text{dist}(v, P') \leq 2r$. The reverse neighborhood $S_r^-(v)$ is defined

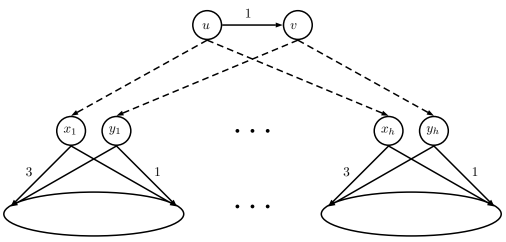

Fig. 10. Example where the EPHS is much larger than the LSHS.

analogously. With this modified versions of r-significant and neighborhood, the notions of LSHS and HD are the same as our previous definitions.

Under this stronger notion, we have $\Delta \leq h$ and $\alpha \leq h+1$. Additionally, this definition allows proving results for CH. Finally, we show that under the stronger definition, CHD and HD can still be off by a factor of n.

Proposition A.2. *For any h, we can construct a family of networks such that the sparsity of LSHS is h and that of EPHS is arbitrarily worse than h.*

Proof. First, we construct an example where the sparsity grows from h to h^2. Consider an h-ary tree rooted at u with three levels, i.e., with $1+h+h^2$ nodes. Now add a node v with h children as in Figure 10. The grandchildren of v are the same as the grandchildren of u.

All the edges are bidirectional and have unit cost. The lengths are as follows: ux_i and vy_i (dashed in Figure 10) are zero; uv and from y_i to the leafs is one; from x_i to the leafs is three. It is easy to see that the sparsity of a LSHS is $h+1$.

On the other hand, every leaf w is a 2-efficient path. Indeed, it can be extended to $x_i w$ that is the shortest path from x_i to w with constraint 1. All the leafs are in the ball $B_4(u)$, so the sparsity is at least h^2.

The general case works in the same fashion. We make the sparsity grow to h^k by creating two complete, k-level, h-ary trees T and T'. Connect the root of T to the root of T' and the leafs of both trees are shared. Observe that the number of nodes is

$$\begin{aligned} n &= [k\text{-level } h\text{-ary tree}] + [(k-1)\text{-level } h\text{-ary tree}] \\ &= (h^{k+1}-1)/(h-1) + (h^k-1)/(h-1), \end{aligned}$$

therefore the sparsity is $\Theta(n)$, the worst possible. □

B CONTRACTION HIERARCHIES

We present here how to extend the concept of HD in order to prove the efficiency of CH in directed graphs. Given a rank in the nodes, the shortcut process works as in the non-directed case:

(1) Let G' be a temporary copy of G.
(2) Remove nodes of G' and its edges in increasing rank.
(3) When removing v, if some unique shortest path in G uses uvw, add (u,w) to G' with length $\ell(u,v)+\ell(v,w)$.

Call E^+ the set of edges created in the shortcut process. A source-destination query runs bidirectional Dijkstra, but each search only considers paths of increasing ranks.

As in the non-directed case, let $Q_i = C_i \setminus \cup_{j>i} C_j$ be the partition of V. All the ranks in Q_i are smaller than those in Q_{i+1}, within each Q_i the rank is arbitrary.

Lemma B.1. *Let P be a shortest path in the original graph. If P has at least three vertices and $\ell(P) > 2^\gamma$, then some internal vertex of P belongs to a level Q_x, $x > \gamma$.*

Proof. The path P' obtained by removing the endpoints of P is $\ell(P)$-significant. By definition of the C_i's, $C_{\gamma+1}$ hits P' at some node u. By construction of the partition, $u \in Q_x$ with some $x > \gamma$. □

Now we show that each node adds at most h to its out-degree for each Q_i, so the process adds at most $h \log D$ to the out-degree of each node.

Lemma B.2. *Assume the network admits the C_i's. For any v and fixed j, the number of shortcuts (v, w) with $w \in Q_j$ is at most h.*

Proof. Let i be the level such that $v \in Q_i$ and define $\gamma := \min(i, j)$. We claim that $w \in B^+_{2^\gamma}(v)$. Assume the claim, then the number of shortcuts is at most $|Q_j \cap B^+_{2^\gamma}(v)|$, but using local sparsity and set inclusion:

$$|Q_j \cap B^+_{2^\gamma}(v)| \le |C_j \cap B^+_{2\cdot 2^{j-1}}(v)| \le h.$$

All that remains is to prove the claim. The shortcut (v, w) was created when the process removed the last internal vertex of the shortest path $P(v, w)$ in G. Necessarily all the internal vertices are in levels at most γ, because they were removed before v and w, hence they have lower rank. Finally, apply Lemma B.1 to conclude that $\ell(P(v, w)) \le 2^\gamma$. □

We need to bound the in-degree, because it could be that some node v is receiving many edges. The proof is basically the same.

Lemma B.3. *Assume the network admits the C_i's. For any v and fixed j, the number of shortcuts (w, v) with $w \in Q_j$ is at most h.*

Proof. Same as in the previous lemma, but now $w \in B^-_{2^\gamma}(v)$. □

We can conclude now that the number of shortcuts, i.e. $|E^+|$, is at most $2nh \log D$.

As we mentioned before, the query performs Dijkstra from the source and target, but always constructing paths of increasing rank. When scanning a vertex v, the forward search has a label $\text{dist}(s, v)'$. The labels always satisfy $\text{dist}(s, v)' \ge \text{dist}(s, v)$, but, since the algorithm only goes to higher ranks, equality is not guaranteed.

We add a pruning rule analogous to the non-directed case: when the forward search scans a node v, if $(v, w) \in E \cup E^+$ and $w \in Q_i$, then w is added to the priority queue only if $\text{rank}(w) > \text{rank}(v)$ and $\text{dist}(s, v)' + \ell(v, w) \le 2^i$. For the reverse search, the condition is the analogous $\text{dist}(v, t)' + \ell(w, v) \le 2^i$ when $(w, v) \in E \cup E^+$.

Proposition B.4. *The query with additional pruning returns the correct distance. Additionally, each Dijkstra scans at most h nodes in each level.*

Proof. Let us analyse the forward search. Say the node v is being scanned, $w \in Q_i$ is a candidate and $\text{dist}(s, v)' + \ell(v, w) > 2^i$. If the current path P' to w is optimal, then $P(s, w)$ is 2^i-significant and it is hit by C_{i+1}. As a consequence, $P(s, w)$ contains an internal vertex with higher rank than w. This vertex cannot be in P' nor a shortcut containing it, thus contradicting the optimality of P'. We conclude that P' is not optimal and w can be ignored.

Bounding the number of scanned nodes is easy; every $w \in Q_i$ added to the queue satisfies $w \in B^+_{2^i}(s)$, so applying local sparsity we finish the proof. □

As a result, the forward search adds at most $h \log D$ nodes to the queue; each of node amounts to $O(\text{outdeg}(G^+))$ operations, i.e., $O(\text{outdeg}(G) + h \log D)$ operations.

C CHD VS. HD: EXTENSIONS

So far we have only used the structure of shortest paths in G, which, naturally, does not capture all the information in the network. It is natural to think that there is structure if we look, for example, only free edges.

Let G_0 be obtained from G by removing all the edges with cost. The networks G and G_0 define two hierarchies of roads; shortest paths in G_0 are free, but not as fast as the ones in G. Our main hypothesis now is that an efficient path P does not alternate between these two hierarchies. For example, a path that enters and exits multiple times a highway is not desirable because of turning costs.

Proposition C.1. *Let $\mathcal{Q}, \mathcal{Q}'$ be two path systems with HD h and h' respectively. The HD of the system $\mathcal{Q} \cup \mathcal{Q}'$ is at most $h + h'$.*

Proof. Given $v \in V$, the union of $H_{v,r}$ and $H'_{v,r}$ hits all the paths in $S_r(v, \mathcal{Q}) \cup S_r(v, \mathcal{Q}')$. □

We now relax the assumption that a system witnesses another. It could be that the efficient paths are sometimes witnessed by free paths and sometimes by shortest paths.

Theorem C.2. *Assume that G has doubling dimension α and $\mathcal{Q}, \mathcal{Q}'$ are systems with HD h, h' respectively. Moreover, suppose $\mathcal{P}^E$ does not alternate between $\mathcal{Q}$ and $\mathcal{Q}'$, that is, for some $\beta, \beta' > 0$, each path $P \in \mathcal{P}^E$ is either β-witnessed by some $Q \in \mathcal{Q}$ or β'-witnessed by $Q' \in \mathcal{Q}'$. Then G admits $(\alpha^\beta h + \alpha^{\beta'} h', r)$-EPHS.*

C.1 Correlated Costs

We have studied so far the case where $c(P)$ is just the sum of individual edge costs. In practice it could be that the cost depends on combinations of arcs. Think of a turn in a road network; we can turn right quickly, but turning left means waiting for a green arrow in most cases. Another example is minimizing expectation subject to bounded variance. If there is no independence, the variance of a path is not the sum of individual variances.

We explain now how to deal with more general cases using the same framework. Assume the cost function $c_2 : E \times E \to \mathbb{N} \cup \{0\}$ depends on pairs of edges, so if a path is $P = e_0 e_1 \ldots e_k$, then the cost would be $c_2(P) = \sum_{i=1}^{k} c_2(e_{i-1}, e_i)$. The nodes in the augmented graph will be triplets $\langle u, v, b \rangle$, where v is the current state, u is the previous state and b is the available budget. The arcs are given by

$$(\langle u, v, b \rangle, \langle v, w, b' \rangle), \quad uv, vw \in E, b' = b - c_2(u, v, 2).$$

Define analogously the concept of efficient paths. It is easy to see that, as in the previous case, shortest paths in the augmented graph are efficient paths. The system $\tilde{\mathcal{P}}^E$ of such paths may also allow for a β-witness. With the previous properties we can construct the hub labels in the same fashion to prove the following result.

Theorem C.3. *Assume the system $\tilde{\mathcal{P}}^E$ has HD $\tilde{h}$. Then, there exists HL such that queries s, t, b can be answered in time $O(b\Delta\tilde{h}\log D)$ and the space requirement is $O(Bn \cdot \Delta B\tilde{h}\log D)$. In particular, if $\mathcal{P}^*$ is a β-witness for $\tilde{\mathcal{P}}^E$, then $\tilde{h} \leq h\alpha^\beta$.*

D ADDITIONAL PROOFS

thm:markedhubs To get this stronger bound, we need to modify the HL construction. The algorithm for forward hub construction is given in Algorithm 1, and for reverse hubs in Algorithm 2. Note that the two must be run sequentially, as the latter uses the nodes marked in the former. We make the forward hubs $L^+(\langle v,b\rangle)$ slightly bigger by storing, for each node the distance from $\langle v,b\rangle$ and also the *budget surplus*. Let C_i be the $(h_c, 2^{i-1})$-EPHS and $\mathcal{P}^E_{s,t}$ the efficient paths from s to t.

Observe that, whenever a node $v \in C_i$ is added, $v \in B^+_{2^i}(s)$ guarantees that at most h_c such points are needed for the whole process. Additionally, every such v is added at most $g(b)$ times in the hub of $\langle s,b\rangle$. The data requirement guarantee follows.

The bound for data requirements is $g(B)h_c \log D$, the argument is analogous to the forward case. Finally, we need to prove the cover property. Take any query $SP(\langle s,b\rangle, t^-)$ and let P be the solution. In $Lf(\langle s,b\rangle)$ there is a node v_P added by Algorithm 1. By construction, the same node v_P was added to $L^-(\langle d,0\rangle)$. The result follows.

Algorithm 1 Construction of forward hub

```
Input: Node $s \in V$, efficient paths $\mathcal{P}^E_{s,t}$ $\forall t$, EPHS $\{C_i\}$.
Output: Forward hubs $Lf(\langle s,b\rangle)$ for $b = 0,\dots,B$ and a marked node $v_P$ for every path.
 1: Order each $\mathcal{P}^E_{s,t}$ by increasing cost and remove paths consuming more than $B$.
 2: for $t \in V \setminus s$ do
 3:     for $P \in \mathcal{P}^E_{s,t}$ do
 4:         $b \leftarrow c(P)$, $b' \leftarrow c(P')$, where $P'$ is the next path in the list ($b' = B$ if no such path).
 5:         Find the largest $i$ such that $P$ is $2^{i-1}$-efficient.
 6:         Find $v \in C_i$ hitting $P$ and mark $v$ as $v_P$.
 7:         Add $\langle v, c(P[v,t])\rangle$ to $L(\langle s,b\rangle)^+$ with distance $\ell(P[s,v])$ and surplus zero.
 8:         for $x$ between $b$ and $b'$ do
 9:             Add $\langle v, c(P[v,t])\rangle$ to $L(\langle s,x\rangle)^+$ with distance $\ell(P[s,v])$ and surplus $x-b$.
10:         end for
11:     end for
12: end for
```

Algorithm 2 Construction of reverse hub

```
Input: Node $t \in V$, efficient paths $\mathcal{P}^E_{s,t}$ $\forall s$, marked nodes and EPHS $C_i$.
Output: Backward hub $L^-(\langle t,0\rangle)$.
 1: Order each $\mathcal{P}^E_{s,t}$ by increasing cost and remove paths consuming more than $B$.
 2: $L^-(\langle t,0\rangle) \leftarrow \varnothing$
 3: for $s \in V \setminus t$ do
 4:     for $P \in \mathcal{P}^E_{s,t}$ do
 5:         Find the largest $i$ such that $P$ is $2^{i-1}$-efficient.
 6:         Take $v$ as the marked node $v_P$.
 7:         Add $\langle v, c(P[v,t])\rangle$ to $L^-(\langle t,0\rangle)$ with distance $\ell(P[v,t])$.
 8:     end for
 9: end for
```

prop:poly$_lshsWeextendsomeargumentsfrom[?, Theorem 8.2].Denote S_r(v) := S^+_r(v,Q) \cup S^-_r(v,Q)$. Observe that, for fixed $v \in V$, the set system $(E, \{\pi(Q) : Q \in S_r(v)\})$ admits a hitting set of size $h\Delta$. Indeed, we know that exists $H_{v,r} \subseteq V$, $|H_{v,r}| \leq h$, hitting every path in $S^+_r(v,Q)$ and in $S^-_r(v,Q)$. The desired hitting set consists of all the edges adjacent to a node in $H_{v,r}$.

If the minimum size of a set system is s and the VC-dimension is d, then the algorithm in [?] obtains, in polynomial time, a hitting set of size at most $O(sd\log(sd))$. In particular, we can use the algorithm to obtain a set $\tilde{F}_{v,r} \subseteq E$, of size at most $h' = O(h\Delta\log(h\Delta))$, hitting the set system $(E, \{\pi(Q) : Q \in S_r(v)\})$.

Consider the set $F_{v,r} \subseteq V$ that contains all the endpoints of edges in $\tilde{F}_{v,r}$. It follows that $F_{v,r} \subseteq V$ can be obtained in polynomial time and is a hitting set for $S_r(v)$ of size $|F_{v,r}| \leq 2h'$. Assume for now that we know the value of h. Note that the value h' can be computed from h and the guarantee given by the oracle, i.e., the constant inside the big-O. We construct the $(2h', r)$-LSHS iteratively. At each iteration i we maintain the following invariant: C_i hits every path in $\mathcal{Q}_r$. In an iteration we check if C_i is locally sparse, if not, we strictly reduce the cardinality of C_i while maintaining the invariant. Start with $C_0 = V$. Let $B_{2r}(v) := B^+_{2r}(v) \cup B^-_{2r}(v)$. Assume $v \in V$ is such that $|B_{2r}(v) \cap C_i| > 2h'$ and let $C_{i+1} := (C_i \setminus B_{2r}(v)) \cup F_{v,r}$. The cardinality strictly decreases and we only need to check the invariant. Consider the paths hit by nodes removed in C_i, this set is

$$\begin{aligned}&\{Q \in \mathcal{Q}_r : Q \cap C_i \cap B_{2r}(v) \neq \varnothing\}\\ &\quad \subseteq \{Q \in \mathcal{Q}_r : Q \cap B_{2r}(v) \neq \varnothing\} \subseteq S_r(v).\end{aligned}$$

Since $F_{v,r}$ hits $S_r(v)$, the proof is completed.

If we do not know the value of h, we can do a doubling search for h'. Indeed, if the guess of h' is low, then at some point it could be that $|F_{v,r}| > 2h'$, then we double h' and restart the process.