Shortest-Path Indices: Establishing a Methodology for Shortest-Path Problems ⋆

Reinhard Bauer, Daniel Delling, and Dorothea Wagner

Universität Karlsruhe (TH), 76128 Karlsruhe, Germany, {rbauer,delling,wagner}@ira.uka.de

Abstract. During the last years, impressive progress has been achieved in the field of algorithm engineering. A problem becoming more and more important is that of *data dependency*: the performance of an algorithm often highly depends on the input. Yet, for a given input, it is highly non-trivial to select the best solving strategy.
In this work, we introduce a new methodology for evaluating speed-up techniques for DIJKSTRA's algorithm: we examine the shortest-path structure of networks using simple indices in order to predict how well speed-up techniques perform on specific networks. More precisely, we introduce the *ReCo-Index* that indicates the strength of hierarchy of the input. In addition, we present a second index, called *shortest-path entropy*, taking into account the mutual importance of consecutive edges. As a third index, the *update impact* measures the change of the shortest-paths structure in a network whenever it is altered. For each index, we present an algorithm for computing it efficiently. An experimental evaluation confirms the correlation of indices and speed-up techniques.

⋆ Partially supported by the Future and Emerging Technologies Unit of EC (IST priority – 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

1 Introduction

Algorithm engineering exhibited an impressive surge of interest during the last years, spearheaded by one of the showpieces of algorithm engineering: computation of shortest paths. In this field, many speed-up techniques for DIJKSTRA's algorithm have been developed (see [1] for an overview). Recent research [2, 3] even made the calculation of the distance between two points in road networks of the size of Europe a matter of microseconds. One problem arising for algorithm engineering in general, and shortest paths in particular, is the following: Performance of algorithms highly depends on the used input, e.g. [2, 3] were developed for road networks and use properties of those networks in order to gain their enormous speed-up. Due to the availability of huge road networks, recent research in shortest paths solely concentrated on those networks [4]. However, fast algorithms are also needed for other applications, e.g. timetable information or routing in sensor networks.

In this work, we introduce a new methodology for evaluating speed-up techniques for DIJKSTRA's algorithm. More precisely, we introduce the notion of *shortest-path indices* which predict the performance of certain speed-up techniques. The first index, called *ReCo-Index*, is designed to predict the performance of hierarchical speed-up techniques while the second, *shortest-path entropy*, scopes goal directed approaches. Hierarchical techniques exploit the importance of nodes (or edges) and construct a routing hierarchy from this information. The key observation for the ReCo-Index is that *centrality measures* also assign importance to graph elements. The distribution of centrality reflects the strength of routing hierarchy in the graph, thus, the ReCo-Index assesses this distribution. However, goal directed speed-up techniques need not depend heavily on the hierarchical structure of a network. Therefore, we introduce a second index—called *shortest-path entropy*—which scopes the intuition of goal-directed approaches. The index identifies the uncertainty of a given edge in a shortest path about how to continue. Imagine you drive on a road and approach the next junction. A low entropy indicates that for most targets in the network you will always take the same road, while a high entropy means that all outgoing roads target many locations and are thus equally valueable. Moreover, we present how to compute the indices. In case of ReCo, two problems arise. On the one hand, a straight-forward implementation involves solving APSP and thus is prohibitive for huge datasets. On the other hand, no unbiased approximation algorithm for reach is known. Our approach is based on branch-and-bound and computes the reach of a single edge exactly. By computation of the reach values of a random sample of edges, we are able to approximate ReCo efficiently. An experimental evaluation of the most common speed-up techniques confirms the correlation of indices and speed-up techniques. More precisely, ALT [5] correlates with our shortest-path entropy, while Highway Hierarchies [6] and the RE algorithm [7] correlate with our ReCo-Index.

The same problem of data dependency arises in dynamic scenarios, a field that is currently tackled [8, 9]. In such a scenario, an edge weight may change between two requests. In order to preserve correctness of speed-up techniques, the preprocessed data has to be updated (at least sometimes) whenever an edge is altered or the metric of the graph changes. Thus, we introduce a measure called *update impact* depicting the impact of the update for shortest-path computation. A high impact indicates that many shortest paths must be rerouted while a small impact means that the structure of the graph with respect to routing has hardly changed. In addition, we show how to approximate the update impact very efficiently.

We point out that our approach is applicable to solve the problem of data-dependency in other fields of algorithm engineering: Generally speaking, we evaluate input data using simple indices in order to predict how well complex algorithms perform on specific datasets.

1.1 Related Work

To the best of our knowledge, no work has been published on analyzing the shortest-path structure of a given network. However, centrality measures relying on shortest-path algorithms are used to analyze a

network with respect to its communication structure, e.g. in social network analysis. Among the most prominent are betweenness, reach, and stress centrality. See [10] for an overview.

As already mentioned, recent research on speed-up techniques for DIJKSTRA's algorithm concentrated on road networks. However, additional tests besides road networks have been published in [11, 7]. Solely in [12] systematic experiments on different types of graphs have been evaluated. As a consequence, the authors state that the choice of speed-up technique depends on the input.

1.2 Overview

This paper is organized as follows. Section 3 introduces our two shortest-path indices: the ReCo-Index and shortest-path entropy. The efficient computation of both indices is located in Section 4. We introduce our update impact in Section 5, including an algorithm to approximate the index efficiently. In Section 6 we evaluate the introduced indices on different types of networks (road and railway networks, small-world graphs, and synthetic data) and show that they do correlate with the speed-up achieved by the most common speed-up techniques on the same networks. Since most techniques have only been tested on road networks, these experiments are of independent interest. Section 7 concludes our work with a brief summary and planned future work.

2 Preliminaries

Throughout the whole work we restrict ourselves to directed graphs $G = (V,E)$ with positive length function $len : E \to \mathbb{R}^+$. The *single-source shortest-path problem* (SSSP) is that of computing the shortest paths from a given node to all other nodes in the graph. The *all-pairs shortest-path problem* (APSP) is that of computing the shortest paths between all nodes in the graph. Fundamental algorithms on these shortest-path problems can be found in [13]. With $\sigma_{st}(u)$ we denote the number of shortest paths from s to t containing the node u. Analogously, $\sigma_{st}(u,v)$ denotes the number of shortest s-t-paths containing the edge (u,v) and $\sigma_{st}(u,v,w)$ denotes the number of shortest s-t-paths containing the edges $(u,v),(v,w)$. By $\sigma_{\bullet\bullet}(\ldots)$ we denote $\sum_{s\in V}\sum_{t\in V}\sigma_{st}(\ldots)$. With $dist(s,t)$ we denote the length of a shortest path from s to t. A *shortest-path directed acyclic graph* (SP-DAG) D of a graph G is a cycle-free subgraph of G for which the following property holds: each path in D is a shortest path in G.

A widely-spread centrality measure is *betweenness*. Let $\delta_{st}(u,v)$ denote the fraction of shortest paths between s and t that contain the edge (u,v), i.e. $\delta_{st}(u,v) = \sigma_{st}(u,v)/\sigma_{st}$. Then, the betweenness centrality $C_B(u,v)$ of an edge (u,v) is defined to be $C_B(u,v) = \sum_{s\in V}\sum_{t\in V}\delta_{st}(u,v)$.

An edge contraction for a node v of degree 2 consists in contracting the two edges incident to v. Two graphs G_1 and G_2 are *homeomorphic* if they become isomorphic after edge contraction for all nodes with degree 2.

Let X denote a discrete random variable with finite co-domain $\{x_1,x_2,\ldots,x_n\}$ for which the probability that X equals x_i is p_i. The *(Shannon-)entropy* of X is defined to be $-\sum_{i=1}^{n} p_i \log(p_i)$. More information about the Shannon-entropy can be found in [14]. When dealing with measures that are mainly computed by counting or summarizing another measure, the *sample technique* (see [10] for an example) can often be used for approximation. This technique first computes the summarized values only for a small subset of the actual required values and then estimates the real value out of this sample. When doing so, it is desireable to get good confidence intervals for the estimated values. In case the estimation is unbiased, one can easily apply Hoeffdings Inequality [15] to get such an error bound: If $X_1,X_2,\ldots,X_K$ are real valued independent random variables with $a_i \le x_i \le b_i$ and expected mean $\mu = \mathbb{E}[\sum X_i/K]$, then for $\xi > 0$

$$\mathbb{P}\left\{\left|\frac{\sum_{i=1}^{K} X_i}{K} - \mu\right| \ge \xi\right\} \le 2e^{-2K^2\xi^2/\sum_{i=1}^{K}(b_i-a_i)^2} \quad .$$

3 Shortest-Path Indices

In this section, we pursue two completely new approaches to obtain information about the shortest-path structure of a graph. The first, the *ReCo-Index* measures the strength of the hierarchy in a given network. The second, *shortest path entropy* tries to measure the mutual importance of consecutive edges.

3.1 The ReCo-Index

Reach. *Reach* is a centrality measure introduced in [16] that is used in the reach-based pruning speed-up technique. The version of reach described here corresponds to the one stated in [7]. The notion of reach derives from the observation that often, when having a long distance shortest path, only at the beginning and the end of the path 'local' edges are used. Intuitively speaking, the reach of an edge is high, if it lies in the middle of at least one long shortest path.

Given a path P_{st} from s to t and an edge (u,v) on P_{st}. The reach of (u,v) with respect to P_{st} is the minimum of the length of the two subpaths P_{sv} and P_{ut}, i.e

$$reach_{P_{st}}(u,v) = \min\{\text{length}(P_{sv}), \text{length}(P_{ut})\} \ .$$

The reach of an edge (u,v) is defined to be the maximum over all shortest paths P containing (u,v) of the reach values of (u,v) with respect to P, i.e.

$$reach(u,v) = \max_{P \in SP(u,v)} \{reach_P(u,v)\}$$

where $SP(u,v)$ denotes the set of all shortest paths containing (u,v).

Lorenz-Curve. One can use the *Lorenz-curve* to visualize the concentration of high centrality measure values on few edges: Given an (w.l.o.g. ordered) set of positive, real valued variables $x_1 \leq x_2 \leq \ldots \leq x_n$, the Lorenz-curve is the piecewise linear function $L : [0,1] \to [0,1]$ connecting the points (u_i, v_i), where $(u_0, v_0) = (0,0)$ and $u_i = i/n$, $S_i = \sum_{j=1}^{i} x_i$, $v_i = S_i/S_n$ for $i = 1$ to n. Assume that $x_1 = x_2 = \ldots = x_n$. Then, the graph of the Lorenz-curve L is a line from $(0,0)$ to $(1,1)$, called *the line of perfect equality*. See Fig. 1 for an example.

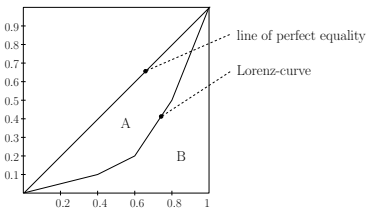

Fig. 1. Lorenz curve of the set $\{1,1,2,6,10\}$

Gini-Coefficient. The *Gini-coefficient* condenses the information given by the Lorenz-curve to one single concentration measure: Let A be the area between the line of perfect equality and the Lorenz-curve and B be the area under the Lorenz-curve. Then the Gini-coefficient is defined to be $A/(A+B)$. At a glance, the Gini-coefficient is high if a big amount of the total sum of all variables is given by the sum of few variables. For a sample X, the Gini-coefficient is computed by the formula $G_X = 1/n \cdot (n+1-2\sum_{i=1}^{n}(n+1-i)x_i/(\sum_{i=1}^{n} x_i))$. More information on the Gini-coefficient and the Lorenz-curve can be found in [17].

ReCo-Index. Given a graph G, the *Reach-Concentration Index* (ReCo-Index) of G is defined to be the Gini-coefficient of the reach values of all edges on the graph. This index is motivated by the observation that on road networks, railway systems and many other graph classes, most edges have low reach while only some high reach edges exist. The distribution of the reach values implies a hierarchy in the graph. The ReCo-Index measures the strength of that hierarchy by measuring the concentration of the reach values. Summarizing, the ReCo-Index is high for graphs with a strong hierarchy and low otherwise.

3.2 Shortest-Path Entropy

While the ReCo-Index measures the shortest-path importance of an edge (u,v) independently from the other edges, the *shortest-path entropy* is an approach to include the mutual importance of consecutive edges. The basic idea is as follows: We choose uniformly at random one of all possibilities to extend a given shortest path that ends with the edge (u,v). The entropy of this edge (u,v) measures the uncertainty about which edges, outgoing from v, will resume the shortest path.

Definition. Let S and T be two independent random variables that choose a node uniformly at random. For an edge (u,v) let $E_{(u,v)}$ be the random variable whose co-domain is the set of edges outgoing of v and whose probability to equal a given edge (v,w) is

$$\mathbb{P}\{E_{(u,v)} = (v,w)\} = \frac{\sigma_{\bullet\bullet}(u,v,w)}{\sigma_{\bullet\bullet}(u,v)} \quad .$$

That is $\mathbb{P}\{E_{(u,v)} = (v,w)\}$, is the conditional probability that (v,w) is on a shortest path between S and T if (u,v) is on a shortest path between S and T. Then, the shortest-path entropy $H(u,v)$ of the edge (u,v) is the entropy of $E_{(u,v)}$:

$$H(u,v) = \sum_{(v,w)\in E} \mathbb{P}\{E_{(u,v)} = (v,w)\} \cdot \log\left(\mathbb{P}\{E_{(u,v)} = (v,w)\}\right)$$

The shortest-path entropy of a graph is the average shortest-path entropy of all of its edges. The interpretation of the shortest path entropy is easy: Given a graph with low shortest path entropy. If we extend a shortest path ending with an edge (u,v) and choose one of all possibilities uniformly at random, (u,v) is very likely to be succeeded by a specific edge (v,w).

Sophisticated Variants. Another way of measuring the entropy of a graph is to compute the *weighted graph entropy*. Here the average over the entropy values of all edges is weighted by their importance which can be measured by betweenness or reach.

Furthermore we want to propose an additional change in the computation of the shortest-path entropy (and, for simplicity, think of undirected graphs). When computing $\sigma_{\bullet\bullet}(\ldots)$ we only count shortest paths whose start- and end-nodes that do not have degree 2. Analogously, we forbid these nodes for the random variables S and T. This change in the definition of entropy has the advantage that homeomorphic graphs have the same entropy.

This proceeding is motivated by the way input-data for shortest-path algorithms is generated. For example, when mapping road networks to graphs, it might be interesting to geographically know the course of the road. Therefore, often long road segments are mapped to many edges covering the course. The precision of this mapping strongly affects the final graph but all possible resulting graphs are homeomorphic to each other.

4 Computation of Shortest-Path Indices

We now show how to compute shortest-path entropy and the ReCo-Index exactly. For the latter, we also describe an approximation algorithm that depends on computing the exact reach values for a given set of sample edges. To this end, we present a branch-and-bound algorithm that allows to compute the values for these samples.

4.1 ReCo-Index

To compute the ReCo-Index of a graph one has to compute the reach values of the graph and then compute the Gini-coefficient of these values. The exact computation of reach requires solving the APSP-problem which is prohibitive for large graphs. In [18] a method is described that partitions the underlying graph to heuristically speed up the computation of reach values. On the other hand [7] gives a method that computes upper bounds for reach values, that is fast enough to work on huge road networks. Unfortunately, this approach is not suitable for our scenario as the difference between the computed and the real values is too big. As the definition of reach is strongly based on maximizing some values it seems unlikely to find good unbiased estimators for reach centrality. Therefore, if one deals with huge graphs where the computation of APSP is prohibitive, another strategy must be employed.

We estimate the ReCo-Index by computing the exact ReCo-Index for a given sample of edges. This approach enables us to work with huge graphs. Unfortunately, just applying the plain definition of reach yields the same effort to compute the reach of only one edge as to compute the reach of all edges. We propose another strategy, i.e. a branch-and-bound algorithm that computes the exact reach for a given edge (u,v) heuristically faster.

The algorithm basically does the same as the straightforward algorithm solving the APSP would do. Additionally, the algorithm first grows a shortest-path DAG $\tilde{T}$ rooted at v on the graph with the reversed edge set. It is easy to see that a shortest-path that is responsible for the reach of (u,v) has to start on the branch T of $\tilde{T}$ that starts with the edge (v,u). Hence, we do not have to consider all shortest paths that start with a node outside T. Then, like the straightforward approach, shortest-paths trees are grown on the original graph. However, we detect early which shortest-paths trees cannot be responsible for the reach of an edge. That way we can stop growing such a shortest-path tree or can even completely avoid growing the tree. See Alg. 1 for the pseudocode of the algorithm.

Algorithm 1: COMPUTESINGLEEDGEREACH(u,v)

input : graph $G=(V,E)$, edge (u,v), tuning-parameter $\gamma \in [0,1]$
output: exact reach of the edge (u,v)

1 $\tilde{T}$:= SP-DAG with root v on reverse edge-set
2 T := branch of $\tilde{T}$ that starts with the edge (u,v)
3 **forall** $w \in V$ **do** compute height of w in T; set w to unblocked
4 reach(u,v):=0
5 **while** *there are unblocked nodes in* T **do**
6 P:=unblocked subpath of a path (in T) from s to a deepest unblocked node in T
7 w:=node in P such that the length of P's unblocked subpath to w is approx $\gamma \cdot len(P)$
8 R:=grow SP-tree rooted at w; when settling nodes, always prefer the branch that contains (u,v); stop growing the tree if
9 (α) all descendants of v in R are finished or
10 (β) for at least one settled descendant d of v: $dist(w,v) \leq dist(u,d)$
11 $reach(u,v) := \max\{reach(u,v), min\{dist(w,v), height(u) \text{ in } R\}\}$
12 **if** *not* (β) **then**
13 delete w from T
14 delete all nodes from T which are not connected with v in T any more
15 **if** (β) *or* $reach(u,v)$ *has been increased* **then**
16 block all nodes d in T with $dist(d,v) \leq reach(u,v)$

To obtain an algorithm with relative error bound ε, line 5 has to be replaced by *while there exist unblocked nodes* d *with* $dist(u,d) > reach(u,v) \cdot (1+\varepsilon)$. Note that for updating the heights in T after a node has been deleted in line 14 we just have to exploit the topological ordering implicitely given by the

distances labels in T. Therefore, we can update the height of all nodes on the tree in $O(n)$ runtime. A proof of correctness is given in Appendix B.

4.2 Shortest-Path Entropy

Here we sketch an algorithm that computes the shortest-path entropy of a given graph in $O(n(n\log n + m\cdot\Delta))$ where n denotes the number of nodes, m the number of edges and Δ the maximum outgoing degree over all nodes in the graph. The algorithm is similar to the betweenness algorithm in [19]. Its pseudocode can be found in Appendix C.

The basic work to compute the shortest-path entropy consists of computing $\sigma_{\bullet\bullet}(u,v)$ for each edge (u,v) and $\sigma_{\bullet\bullet}(u,v,w)$ for each pair of edges (u,v), (v,w). To compute $\sigma_{\bullet\bullet}(u,v)$ we sum up $\sigma_{s\bullet}(u,v)$ for each node s. The computation of $\sigma_{s\bullet}(u,v)$ for a single node s is done as follows: It is easy to see that $\sigma_{s\bullet}(u,v) = \sigma_{su}\sum_{t\in V}\sigma_{vt}\cdot 1_{\{dist(s,u)+len(u,v)+dist(v,t)=dist(s,t)\}}$. Therefore we grow a SP-DAG D rooted at s. To compute σ_{su} we can exploit the equation $\sigma_{su} = \sum_{w\in Pred_s(u)}\sigma_{sw}$ where $Pred_s(u)$ denotes the set of all direct predecessors of u in D: When settling a node of D we can compute σ_{su} by summarizing σ_{sx} for all direct predecessors x of u in D

After growing D we can compute $\sigma_{sv\bullet} := \sum_{t\in V}\sigma_{vt}\cdot 1_{\{dist(s,v)+dist(v,t)=dist(s,t)\}}$ for each node by visiting the nodes in descending distance from s. We know that $\sigma_{sv\bullet} = 0$ if v is a leaf. Otherwise we exploit $\sigma_{sv\bullet} = \sum_{(v,w)\in E} 1_{\{dist(s,w)=dist(s,v)+len(v,w)\}}(1+\sigma_{sw\bullet})$ and compute $\sigma_{sv\bullet}$ by summarizing $1+\sigma_{z\bullet}$ for every direct successor z of v in D. The computation of $\sigma_{\bullet\bullet}(u,v,w)$ works analogously.

5 Update Impact

Given two graphs G_A and G_B with the same topology but different length functions *lenA*, *lenB* (either representing completely different profiles or updates of some edges because of traffic jams, etc). We present a new index for measuring the similarity of these graphs with respect to their shortest-path structure. Such a measure can be used for benchmarks in dynamic shortest-path scenarios. Furthermore, one can compute such a measure to check whether it is feasible to compute a dynamic update within reasonable time.

Definition. For two nodes s and t let $\varphi_{st}(u,v) = 1$ if the edge (u,v) is on a shortest s-t-path in exactly one of two graphs G_A and G_B, otherwise $\varphi_{st}(u,v) = 0$. Accordingly, $\varphi_s(u,v)$ equals 1 if the edge (u,v) is in the SP-DAG rooted at s in exactly one of both graphs, otherwise $\varphi_s(u,v) = 0$. The *update impact* $UI(lenA, lenB)$ is the probability that a uniformly at random chosen edge (u,v) is on a shortest s-t-path on exactly one of the two graphs G_A and G_B, where s and t are chosen independently and uniformly at random.

$$UI(lenA, lenB) = \frac{\sum_{s\in V}\sum_{t\in V}\sum_{(u,v)\in E}\varphi_{st}(u,v)}{|V|^2|E|} = \frac{\sum_{s\in V}\sum_{(u,v)\in E}\varphi_s(u,v)}{|V||E|}$$

Estimate Computation. The exact update impact of two graphs can be computed in $O(n(n\log n+m))$ time by growing, for each node, one shortest-path tree w.r.t. *lenA* and one shortest-paths tree w.r.t. *lenB*. Only for updates that are sufficiently different from each other it is reasonable to approximate the update impact. Given $G=(V,E)$, *lenA*, *lenB* and a significance niveau α, the estimation algorithm works as follows: We iteratively choose a node at random and grow a SP-DAG on $(G, lenA)$ and one on $(G, lenB)$. At step K we count the number of edges L_K that are contained in exactly one of both DAGs. The current estimate for the update impact is $\sum_{i=1}^{K} L_i/(|E|K)$. The relative length of the current confidence interval is $\gamma_i := \sqrt{K\cdot\log(2/\alpha)/(2(\sum L_i)^2)}$. We stop if γ_i is smaller than a pre-chosen value γ.

Justification. Now we are going to justify the estimation algorithm. We want to compute $UI(lenA, lenB) = \sum_{s \in V} \sum_{(u,v) \in E} \varphi_s(u,v)/(|V||E|)$. With $L_s := \sum_{(u,v)} \varphi_s(u,v)/|E|$ it follows that $UI(lenA, lenB) = \sum_{s \in V} L_s/|V|$. It holds that $0 \leq L_s \leq 1$ and $\mathbb{E}[L_s] = \sum L_s/|V| = UI(lenA, lenB)$. Therefore the value computed by the approximation algorithm, $\sum_{i=1}^{K} L_{s_i}/K$ is an unbiased estimate for the update impact. To obtain the confidence intervals we apply Hoeffding's Bound: Let $s_1, \ldots, s_K$ be independent, randomly chosen nodes. Then

$$\mathbb{P}\left\{ \left| \sum_{i=1}^{K} L_{s_i}/K - UI(lenA, lenB) \right| \geq \xi \right\} \leq 2e^{-2\xi^2 K^2/K} = 2e^{-2\xi^2 K}.$$

To obtain a confidence interval with length γ relative to the computed value of $I(lenA, lenB)$ and significance niveau α we set $\gamma := \xi \cdot K / \sum_{i=1}^{K} L_{s_i}$

$$\mathbb{P}\left\{ \left| \frac{\sum_{i=1}^{K} L_{s_i}/K - UI(lenA, lenB)}{\sum_{i=1}^{K} L_{s_i}/K} \right| \geq \gamma \right\} \leq 2e^{-2\xi^2 K} = 2e^{-2\gamma^2 (\sum L_s)^2/K} \leq \alpha$$

This yields

$$\sqrt{K \cdot \log(2/\alpha)/(2(\sum L_{s_i})^2)} \leq \gamma$$

and thus the confidence interval computed by the algorithm is verified.

6 Experiments

Here, we present an experimental evaluation of our shortest-path indices from Section 3 and our update impact from Section 5. In Section 6.1, we use different types of graphs and evaluate the scores achieved with respect to our indices and show that these scores correlate with certain speed-up techniques. For showing the applicability of our update impact, we evaluate the impact of changing the metrics of a road network in Section 6.2. Our implementation is written in C++ solely using the STL. Our experimental evaluation was done on one core of an AMD Opteron 2218 running SUSE Linux 10.1. The machine is clocked at 2.6 GHz, has 16 GB of RAM and 2 x 1 MB of L2 cache. The program was compiled with GCC 4.1, using optimization level 3.

6.1 Shortest-Path Indices

In order to show the correlation of our ReCo-Index with hierarchical techniques on the one hand, and shortest-path entropy with goal-directed approaches on the other hand, we evaluated a collection of networks (European and US road networks, timetable information, internet graphs, citation graphs, and synthetic data) with different metrics. We here present a summary of the most interesting results. Note that we also evaluated the Gini-coefficient of the distributions obtained from betweenness values. However, it turned out that, computing these values for different graphs, the observed variance was too small to interpret. Therefore, we do not report these results here.

Speed-Up Techniques. Due to the fact that many speed-up techniques exist, we restrict ourselves to those used most often in literature and which do *not* require a layout of the input graph. As most speed-up techniques use bidirectional search, we use bidirectional DIJKSTRA as a reference. We evaluate the speed-up techniques with respect to settled nodes which denotes the number of nodes taken from the priority queues.

We use the latest variant of ALT, introduced in [20], with 16 *maxCover* landmarks as representative of goal-directed search. It pushes the search towards a target by adding a potential—obtained from the distances to certain landmarks in the graph—to the priority of each node. As an approach based on *edge-labels*, we evaluate Arc-Flags [21]. This approach partitions the graph into cells and attaches a label to

each edge. A label contains a flag for each cell indicating whether a shortest path to the corresponding cell starts with this edge. In our setup, we use 128 cells obtained from METIS [22]. Reach has originally been introduced as *node-label* [16]. These labels are used to prune unnecessary nodes during the search. In [7], reach is redefined as an edge-label and combined with ALT leading to significant speed-ups. Here, we use Reach (RE) and its combination with ALT (REAL) as described in [7]. Note, that RE can be interpreted as a *hierarchical* method that tries to exploit the hierarchy of a given network. As a purely hierarchical approach, we use Highway Hierarchies (HH) [6]. We use a distance table at a level with about 1 000 nodes. Note that we tune preprocessing parameters of HH and RE manually.

Inputs. We use two different road networks as inputs. One represents New York. It is taken from the DIMACS webpage [4], and has 264 346 nodes and 730 100 edges. The second network is a subgraph of the German road network, provided by PTV AG for scientific use. It has 225 624 nodes and 572 182 edges and shows the city Karlsruhe and surrounding areas. As a further transportation network, we use a condensed railway graph of Europe (29 578 nodes 86 566 edges), provided by HAFAS. Such a network is obtained from timetable information by modeling stations as nodes and inserting an edge if at least one direct connection between two stations exists. Transportation networks have in common, that they incorporate some kind of hierarchy. Moreover, we evaluate a subgraph of an internet graph based on router level (54 824 nodes, 591 600 edges), taken from the CAIDA webpage [23]. This graph has small-world properties. As a second small-world graph, we use a co-authorship graph [24] obtained from crawling the literature database DBLP [25]. It has 50 004 nodes and 495 052 edges. In addition, we use synthetic data. The advantage of synthetic data is that we can exactly tune which behavior we want to evaluate. We use multi-dimensional grids and leave the number of nodes untouched (100 000) but increase the number of dimensions. As a consequence the number of edges increases and the diameters decrease with increasing dimension. We use 2, 3, and 4 dimensions.

If applicable, we additionally used different metrics for evaluation. More precisely, we use travel times, distances, unit lengths, and edge weights picked uniformly at random (in the following denoted as *uniform*) as metrics.

Note that we restrict ourselves to mid-size graphs although in the field of shortest path computation networks with several millions of nodes and edges are commonly used. As we have systematically evaluated a set of graphs and speed-up techniques, preprocessing of the latter would have taken to much CPU-time on big networks.

Evaluation. In general, we evaluated the value scored by our indices and the speed-up achieved by the most important speed-up techniques with respect to settled nodes. Table 1 reports these figures in the graphs introduced in the last paragraph. Recall that a *high* ReCo index indicates a strong hierarchy in the network and thus should predict high speed-ups of RE and HH. On the contrary, *low* entropy values should predict high speed-ups for goal-directed approaches like ALT.

Analyzing our shortest-path indices, we clearly observe a correlation between ReCo and RE/HH on the one hand, and a dependency between entropy and ALT on the other hand. In general, a high ReCo index yields high speed-ups for RE and HH, while a low entropy correlates with high speed-ups for ALT.

Examining all three metrics on the Karlsruhe network, both indices predict the lowest speed-ups on the unit metrics. Indeed, ALT, RE, REAL, and HH achieve the lowest speed-up on the unit metrics. While ALT favors travel times, hierarchical techniques work better (relatively) on the distance metrics. Note that the situation changes for absolute values, but as all algorithms are based on bidirectional DIJKSTRA, we naturally have to compare the speed-up with respect to this reference. The observation for different metrics on the same graph also holds for New York and the railway graph.

Stunningly, using a small world graph like the router based internet graph as input, we observe no speed-up (except Arc-Flags) at all. Even worse, the number of settled nodes even increase. This is predicted by our indices: ReCo scores a very low value of below 0.3, while entropy scores a very high

Table 1. Results of performance indices and speed-up techniques on the Karlsruhe road graph using different metrics and different subgraphs. The figures for our shortest-path indices are based on exact computation. Results for speed-up techniques are based on 1 000 random queries. Column *#settled* depicts the number of nodes settled by the algorithm and *spd* indicates the speed-up with respect to settled nodes compared to bidirectional DIJKSTRA.

INPUT			SP-INDEX		SPEED-UP TECHNIQUE											
					BiDij.	ALT		Arc-Flags		RE		REAL		HH	
type	graph	metrics	ReCo	entropy	#settled	#settled	spd	#settled	spd	#settled	spd	#settled	spd	#settled	spd
road	Karlsruhe	time	0.71	0.13	53 229	1 781	29.9	262	203.3	1 667	31.9	380	140.0	401	132.7
		dist	0.77	0.17	76 280	2 672	28.5	1 018	74.9	1 727	44.2	290	262.7	510	149.6
		unit	0.57	0.23	38 873	1 915	20.3	218	178.2	1 388	28.0	364	106.8	530	73.3
road	New York	time	0.67	0.15	64 422	2 244	28.7	465	138.7	1 532	42.1	235	274.1	483	133.4
		dist	0.74	0.16	81 231	2 858	28.4	1 065	76.3	1 620	50.1	270	300.9	563	144.3
		unit	0.64	0.18	70 764	2 884	24.5	494	143.3	1 951	36.3	284	249.2	595	118.9
railway	Europe	time	0.76	0.15	7 594	346	21.9	49	154.7	298	25.5	75	101.1	85	89.3
		dist	0.78	0.19	8 443	460	18.4	75	113.0	295	28.6	67	125.2	67	126.0
		unit	0.49	0.22	1 192	109	10.9	19	62.7	400	3.0	61	19.5	63	18.9
small world	router	unit	0.16	2.06	147	286	0.5	26	5.7	142	1.0	175	0.8	12 611	0.0
	coAuthors	unit	0.28	1.46	137	202	0.7	44	3.1	108	1.3	222	0.6	19 174	0.0
grids	2 dim	uniform	0.69	0.15	32 977	1 051	31.4	643	51.3	2 139	15.4	344	95.9	574	57.5
	3 dim	uniform	0.51	0.24	16 728	799	20.9	718	23.3	9 387	1.8	1 273	13.1	1 077	15.5
	4 dim	uniform	0.45	0.29	8 331	706	11.8	1 145	7.3	8 689	1.0	1 592	5.2	40 714	0.2

value of above 1. Highway Hierarchies fail on this graph due to their weaker stopping criterion (cf. [6]). These general small speed-ups originate from the fact that switching from uni- to bidirectional search already results in a big speed-up (cf. Tab. 3 in Appendix A) which is due to the small diameter and high connectivity of small-world networks.

For grid graphs, we observe that with increasing number of dimensions, shortest-path computation gets more and more complicated. This is indicated by our indices and speed-up techniques only achieve mild speed-ups on 4-dimensional grids while their performance is still pretty good for 2-dimensional grids.

In general, we observe that the values scored by entropy indicate the speed-up that is achieved by ALT: A value between 0.13 and 0.15 yield speed-ups of around 30, while values above 0.2 yield speed-ups of below 20. The same holds for the ReCo-Index and RE/HH. However, other aspects of the graph are important as well, e.g. the number of landmarks for ALT, parameter settings for RE/HH, and size of input. Thus, it is clear that a certain index value does not indicate the exact speed-up of a technique. However, the indices give a strong indication, whether goal-directed or hierarchical methods should be preferred. Summarizing, the correlation between indices and speed-up techniques is clearly visible.

6.2 Update Impact

For evaluating the update impact index, we use the Karlsruhe road network. The original data contains 13 different types of road categories. By applying different average speeds to each category we obtain different client profiles. The *linear*—denoted as travel times in the last section—profile is obtained by assigning average speeds of 10, 20, ..., 130 to the 13 categories, *slow/fast car* and *slow/fast truck* represent typical average speeds for these types of vehicles, while *distance* depicts the real distance, and *unit* assigns 1 to each edge. We also evaluate two synthetic metrics: *uniform* reassigns edge weights uniformly at random while *power law* uses a power law distribution for reassignments. The results of our indices and according speed-up techniques on those metrics not reported in Tab. 1 can be found in Tab. 4 in Appendix A.

Here, we test how much the routing structure is altered when switching from one metrics to another. The resulting figures can be found in Tab. 2. We observe small update impacts when switching within travel times metric (linear, fast and slow car, fast and slow truck). This coincides with the figures from

Table 2. Update impact of the Karlsruhe road network applying different metrics.

	distance	fast car	slow car	fast truck	slow truck	unit	power law	uniform
linear	0.22	0.052	0.045	0.044	0.049	0.29	0.33	0.30
distance		0.22	0.21	0.21	0.20	0.30	0.32	0.31
fast car			0.015	0.052	0.060	0.28	0.33	0.29
slow car				0.042	0.049	0.28	0.33	0.29
fast truck					0.014	0.28	0.32	0.29
slow truck						0.28	0.32	0.30
unit							0.30	0.19
power law								0.30

Tabs. 1 and 4 where all speed-up techniques show similar behavior on all tested travel times metrics. As expected, changing from slow car to fast car and from slow truck to fast truck hardly changes the routing structure yielding the smallest scores of our updated impact. The highest scores are achieved by switching to synthetic metrics. The reason for this is the random reassignment of *all* edges, while for other metrics, the 13 original categories are used. Thus, the routing structure is changed very significantly by such an update.

7 Outlook and Conclusion

In this work we establish a new methodology for evaluating speed-up techniques for DIJKSTRA's algorithm which can be adapted to overcome the dilemma of data dependency arising in the field of algorithm engineering in general. By analyzing the shortest-path structure in a network with simple indices we are able to predict the performance of the most prominent speed-up techniques. More precisely, we introduce two shortest-path indices, called the ReCo-Index and shortest-path entropy. These carefully designed measures capture highly relevant aspects of the shortest-path structure in a graph that are heuristically exploited by speed-up techniques. The feasibility of our theoretical approach is clearly confirmed by an extensive experimental evaluation with different types of networks: Our indices strongly correlate with speed-up techniques. More precisely, the two indices predict the performance of speed-up techniques and thus support the choice of technique for a given input. For dynamic scenarios, we introduce an index—called update impact—indicating how much the shortest-path structure of a network changes whenever the input is perturbed. This index assesses the feasibility of a dynamic update which we confirm by an experimental study. Finally, we present novel and efficient algorithms to compute all three indices.

For shortest-path indices, we see plenty of future work. While our indices predict the performance of ALT and hierarchical methods very well, they fail for Arc-Flags. Since Arc-Flags heavily rely on the employed partition, it may be worth focusing on quality measures for graph clusterings [26], e.g. modularity [27]. Such measures promise to be suitable for predicting the performance of partition based speed-up techniques. In addition, an index indicating the speed-up when switching from uni- to bidirectional algorithms may be interesting as well. The successfull application of our indices to shortest-path computation suggests their adaptation to related problems on graphs, e.g. fingerprints, visualizations or taxonomies. Finally, faster and better approximation algorithms for computing our indices are the next canonical step.

Summarizing, our indices predict the performance of speed-up techniques for DIJKSTRA's algorithm very well. Thus, they are capable of revealing weaknesses and further potential of current or new speed-up techniques to come.

Acknowledgments. We thank Dominik Schultes for providing the Highway Hierarchies code and his help on parameter settings, Marco Gaertler and Robert Görke for their valueable input, and Daniel Karch for implementing Arc-Flags.

References

1. Wagner, D., Willhalm, T.: Speed-Up Techniques for Shortest-Path Computations. In: 24th International Symposium on Theoretical Aspects of Computer Science (STACS). (2007) 23–36
2. Delling, D., Holzer, M., Müller, K., Schulz, F., Wagner, D.: High-Performance Multi-Level Graphs. In: 9th DIMACS Challenge on Shortest Paths. (2006)
3. Bast, H., Funke, S., Matijevic, D., Sanders, P., Schultes, D.: In Transit to Constant Time Shortest-Path Queries in Road Networks. In: Algorithm Engineering and Experiments (ALENEX). (2007) 46–59
4. 9th DIMACS Implementation Challenge: Shortest Paths. `http://www.dis.uniroma1.it/~challenge9/` (2006)
5. Goldberg, A.V., Harrelson, C.: Computing the shortest path: A^* meets graph theory. In: 16th ACM-SIAM Symposium on Discrete Algorithms. (2005) 156–165
6. Sanders, P., Schultes, D.: Engineering highway hierarchies. In: 14th European Symposium on Algorithms (ESA). (2006) 804–816
7. Goldberg, A., Kaplan, H., Werneck, R.: Reach for A*: Efficient Point-to-Point Shortest Path Algorithms. In: Algorithm Engineering and Experiments (ALENEX). (2006) 129–143
8. Sanders, P., Schultes, D.: Dynamic Highway-Node Routing. In: 6th Workshop on Experimental Algorithms (WEA). (2007) 66–79
9. Delling, D., Wagner, D.: Landmark-Based Routing in Dynamic Graphs. In: 6th Workshop on Experimental Algorithms (WEA). (2007) 52–65
10. Brandes, U., Erlebach, T., eds.: Network Analysis: Methodological Foundations. Volume 3418 of Lecture Notes in Computer Science. Springer-Verlag (2005)
11. Wagner, D., Willhalm, T., Zaroliagis, C.: Geometric containers for efficient shortest-path computation. ACM Journal of Experimental Algorithmics **10** (2005) 1–30
12. Holzer, M., Schulz, F., Willhalm, T.: Combining speed-up techniques for shortest-path computations. In: 3rd Workshop on Experimental and Efficient Algorithms (WEA). (2004) 269–284
13. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. McGraw-Hill (1990)
14. McEliece, R.J.: The theory of information and coding. 2. ed. edn. Cambridge Univ. Press (2002)
15. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association, Volume 58, pages 13-30 (1963)
16. Gutman, R.J.: Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In: Algorithm Engineering and Experiments (ALENEX), SIAM (2004) 100–111
17. Gini, C.: Measurement of inequality and incomes. the economic journal 31: 124-126 (2001)
18. Goldberg, A.V., Kaplan, H., Werneck, R.: Better Landmarks within Reach. In: 6th Workshop on Experimental Algorithms (WEA). (2007) 38–51
19. Brandes, U.: A Faster Algorithm for Betweenness Centrality. Journal of Mathematical Sociology **25** (2001) 163–177
20. Goldberg, A.V., Werneck, R.F.: An efficient external memory shortest path algorithm. In: Algorithm Engineering and Experimentation (ALENEX). (2005) 26–40
21. Hilger, M., Köhler, E., Möhring, R.H., Schilling, H.: Fast Point-to-Point Shortest Path Computation with Arc-Flags. In: 9th DIMACS Challenge on Shortest Paths. (2006)
22. METIS: A family of multilevel partinioning algorithms. `http://glaros.dtc.umn.edu/gkhome/views/metis/` (1995)
23. CAIDA: Cooperative Association for Internet Data Analysis. `http://www.caida.org/` (2001)
24. An, Y., Janssen, J., Milios, E.E.: Characterizing and mining the citation graph of the computer science literature. Knowl. Inf. Syst. **6** (2004) 664–678
25. DBLP: DataBase systems and Logic Programming. `http://dblp.uni-trier.de/` (2007)
26. Brandes, U., Gaertler, M., Wagner, D.: Experiments on Graph Clustering Algorithms. In: 11th European Symposium on Algorithms (ESA). (2003) 568–579
27. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E **69** (2004)

A Further Experiments

Table 3. Results for uni- and bidirectional speed-up techniques using small world graphs as inputs. *Type* indicates whether uni- or bidirectional search is used. The figures are based on 1 000 random queries. Query times are reported in milliseconds.

		router graph		coAuthors graph	
algorithm	type	#settled	query[ms]	#settled	query[ms]
Dijkstra	uni	28,244	29.79	25,293	20.29
	bi	147	0.28	137	0.17
ALT	uni	7,247	14.04	6,781	9.90
	bi	209	1.31	262	0.91
Arc-Flags	uni	1,917	5.10	4,345	8.55
	bi	26	0.09	44	0.11
RE	bi	142	0.30	108	0.19
REAL	uni	7,130	16.82	6,804	13.58
	bi	175	1.51	222	1.33
HH	bi	12,611	226.52	19,174	64.71

Table 4. Results of performance indices and speed-up techniques on the Karlsruhe road graph using different metrics. The figures for our shortest-path indices are based on exact computation. Results for speed-up techniques are based on 10 000 random queries.

INPUT	SP-INDEX		SPEED-UP TECHNIQUE											
			BiDij.	ALT		Arc-Flags		RE		REAL		HH	
metrics	ReCo	entropy	#settled	#settled	spd	#settled	spd	#settled	spd	#settled	spd	#settled	spd
fast car	0.71	0.14	44 489	1 670	26.6	254	175.2	1 251	35.6	343	129.9	341	130.5
slow car	0.72	0.14	51 201	2 136	24.0	272	188.0	1 396	36.7	413	123.9	367	139.5
fast truck	0.72	0.14	55 419	2 277	24.3	293	189.4	1 451	38.2	412	144.2	393	141.0
slow truck	0.72	0.14	59 299	2 376	25.0	301	197.1	1 542	38.5	430	138.0	412	143.9
power law	0.69	0.12	54 244	1 752	31.0	409	132.5	1 637	33.1	461	117.8	435	124.7
uniform	0.61	0.16	40 499	1 932	21.0	217	186.4	1 355	29.9	372	108.7	442	91.6

B Correctness of the Reach Sampling Algorithm

In this section, the shortest-paths tree T and the edge (u,v) are defined like in the pseudocode of the algorithm. A shortest path P is called *responsible* for the reach of an edge (u,v) if $reach_P(u,v) = reach(u,v)$.

Lemma 1. A shortest path responsible for the reach of (u,v) has to start at a node on T. *Proof:* If P is a shortest path on G containing (u,v) then the subpath of P with end node v is on a shortest-path tree on the reverse edge set that starts at v and contains (u,v).

Lemma 2. When computing the reach of (u,v) no node x on T has to be considered for which the distance on T from v is smaller or equal to the currently found lower bound for the reach of (u,v). *Proof:* The distance from v to x on T is the actual distance from x to v on G. The length of the suffix of a given path starting at a node x is $dist(x,v)$ at most, therefore the reach with respect to that path is also $dist(x,v)$ at most.

Lemma 3. In case (β) all nodes x on T with $dist(x,v) \leq dist(w,v)$ get blocked. *Proof:* In that case the reach of (u,v) is $dist(w,v)$ at least.

Lemma 4. The blocking in case (*not* β) is correct. *Proof:* That case implies case (α). Let $P = (w, \ldots, u, v, \ldots, z)$ be a maximal shortest path starting with w and containing (u, v) and $\tilde{P} = (x, \ldots, w, \ldots, u, v, \ldots, z, a)$ be a shortest path. Then would be $(w, \ldots, u, v, \ldots, z, a)$ also a shortest path starting at w and containing (u, v). Therefore P would not be maximal. In conclusion we know that the suffix of all shortest paths that contain w in the prefix and on which w is not the start node have length of $dist(w, v)$ at most and we therefore can omit them.

Theorem 1. The algorithm terminates. *Proof:* Every node v gets blocked or deleted after it is processed: In case (*not* β) is gets directly deleted. In case (β) we apply Lemma 3 and know that it also gets blocked.

Theorem 2. The algorithm is correct. *Proof:* By definition of reach we know that we can compute correct reach values by growing a full shortest-paths tree on each node on the graph. With Lemmata 1 to 4 we know that each restriction of the branch-and-bound algorithm will not change the computed reach value.

C Computation of Shortest-Path Entropy

Algorithm 2: COMPUTESHORTESTPATHENTROPY(G)

input : graph $G = (V, E)$
output: $P\{(v,w)|(u,v)\} := \mathbb{P}\{E(u,v) = (v,w)\}$ like in section 3.2

// forall $u, v, w \in V$: initialize $\sigma_{\bullet\bullet}(u,v)$ and $\sigma_{\bullet\bullet}(u,v,w)$ with zero
forall $s \in V$ **do**
    $\sigma_{sv} := 0$, $v \in V$
    $\sigma_{ss} := 1$
    $\sigma_{v\bullet} := 0$, $v \in V$
    $T :=$ grow shortest-path tree rooted at s
    when a node n is settled set $\sigma_{sn} := \sum_{\{(w,n) \in E : d(s,w) + len(w,n) = d(n)\}} \sigma_{sw}$
    // $d(n)$ represents the distance from s to n computed by the SP-tree
    // S returns nodes in order of non-increasing distance from s
    while *S not empty* **do**
        $v :=$ pop S
        forall $(v,w) \in E$ **do**
            if $d(v) + len(v,w) = d(w)$ **then** $\sigma_{v\bullet} := \sigma_{v\bullet} + \sigma_{w\bullet} + 1$
    forall $(u,v) \in E$ **do**
        $\sigma_{\bullet\bullet}(u,v) := \sigma_{\bullet\bullet}(u) + \sigma_{su} \cdot \sigma_{v\bullet}$
    forall $(u,v) \in E$ *with* $d(u) + len(u,v) = d(v)$ **do**
        forall $(v,w) \in E$ *with* $d(v) + len(v,w) = d(w)$ **do**
            $\sigma_{\bullet\bullet}(u,v,w) := \sigma_{\bullet\bullet}(u,v,w) + \sigma_{su} \cdot \sigma_{w\bullet}$
forall $(u,v) \in E$ **do**
    forall $(v,w) \in E$ **do** $P\{(v,w)|(u,v)\} := \sigma_{\bullet\bullet}(u,v,w) / \sigma_{\bullet\bullet}(u,v)$

The asymptotic runtime of the algorithm is $O(n(n \log n + m \cdot \Delta))$ where n denotes the number of nodes, m denotes the number od edges and Δ denotes the maximum outgoing degree over all nodes on the graph.