Shortcut Removal On SHARC

Edith Brunel

September 6, 2009

Student Research Project
Universität Karlsruhe (TH), 76128 Karlsruhe, Germany
Supervised by D. Delling, Prof. Dr. Dorothea Wagner

Contents

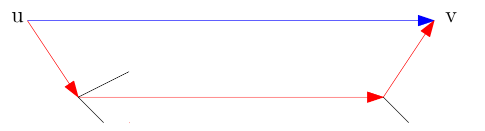
1 Introduction

Motivation. Shortest path computation between a given source s and target t is a classical problem in graph theory with a number of significant real-world applications, namely route planning in transportation networks, logistic planning and traffic simulation. The standard for this problem, Dijkstra's algorithm, although running in $\mathcal{O}(m + nlogn)$, does not achieve sufficiently fast query times for use on large datasets, such as continental-sized road networks. A number of speed-up techniques exploiting specific properties of such networks have been developed, many of them relying on extensive precomputation. Any such approach operates by preprocessing the network data during an *offline* phase, where information is precalculated and stored which later on helps to reduce the shortest path query times during the *online* phase, when the actual path-finding algorithm is executed. While the speed-up factors that can be achieved are enormous, space consumption for the precomputed data is generally also very high, which hinders the application on systems with limited storage space and memory, e.g. PDAs and other handheld devices. Some of the most successful techniques following this two-phase concept rely, amongst other methods, on enriching the graph with additional *shortcut* edges, that enable the query to traverse long paths while only having to relax a single edge. This student thesis attempts to reduce space overhead for one such speed-up technique in particular, SHARC-Routing [BD09], by identifying and removing unnecessary shortcuts from the preprocessing.

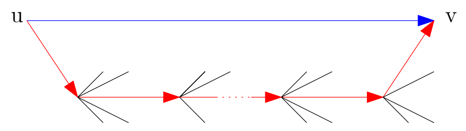
Related Work. An overview on shortest path speed-up methods in general can be found in Engineering Route Planning Algorithms [DSSW09]. SHARC-Routing is a unidirectional approach based on the combination of Arc-Flags [HKMS09, Lau04] with techniques from Highway Hierarchies [SS06], which utilizes arc-flag information and shortcuts, inserted into the graph during several preprocessing steps, to greatly reduce the eventual query time. Another student thesis, Arc-Flag Compression by A. Gemsa [Gem08], dealt with the corresponding problem of minimizing space overhead on SHARC by reducing the number of retained unique arc-flag vectors via probabilistic methods, i.e., hash functions, and various importance measures for the individual flags. The paper Mobile Route Planning by P. Sanders, D. Schultes and C. Vetter [SSV08] demonstrated compression techniques for the related Contraction Hierarchies routing, an entirely shortcut-based approach, in particular with regard to application on mobile devices.

Our Contribution. This student thesis presents a thorough examination of the space reduction attainable by removing nonessential shortcuts on a SHARC preprocessing while maintaining good query performance. This might be especially interesting for the time-dependent variant of SHARC [Del09], where space-intensive edge weight functions make storing additional shortcuts particularly expensive.

The main idea here is, that not all shortcuts added during SHARC preprocessing are of equal importance. Shortcuts which skip only one or two nodes of low degree, or have almost no arc-flags set, are arguably less crucial to the SHARC query than shortcuts representing long paths with many set flags (c.f. figure 1).



(a) A shortcut (u, v) representing a very short path with low degree nodes. It can be removed from the preprocessing without much of an impact on query performance.



(b) Removing shortcuts like this one, however, which represent long paths with a lot of high degree nodes, will result in a severe increase of the search space.

Figure 1: Two shortcuts in comparison.

By focusing on such unimportant shortcuts for removal, it is possible to keep the incurred search space increase to a minimum. Since removing shortcuts from the preprocessing can be done without any alteration to the actual SHARC query algorithm, the additional computational effort introduced by this method of space reduction is comparatively small.

We present a detailed experimental evaluation on how a shortcut's specific attributes, such as their head and tail node level, flag information and the path they represent in the graph, affect its value for the SHARC query. It turns out that, following an order of removal based solely on these properties, about 30-40% of all shortcuts can be discarded without any noticeable impact on query times.

Outline. Section 2 settles some necessary preliminaries, in particular the fundamentals of SHARC routing and the introduction of basic terms and definitions. A proof for correctness of the shortcut removal routine is given in section 3. The main subject of this work are a number of weighting metrics assessing a shortcut's significance for the SHARC query. These are detailed in section 4, grouped by the shortcut properties they are based on, namely level of head and tail node, set arc-flags, and represented path. Subsequently, several favourable combinations of these strategies are discussed. Section 5 provides an overview of the experiments conducted and test results. A summary concludes this thesis in section 6.

2 Preliminaries

Let $G = (V, E)$ be a simple, directed graph with node set V and edge set E. An edge is denoted as $(u, v) \in E$ with head node u and tail node v in V.

Multi-Level Arc-Flags Let $C = \{C_0, C_1, ..., C_k\}$ be a family of sets in V. C is called a partition of V, if each $v \in V$ is contained in exactly one C_i. The basic idea behind arc-flags, as detailed in [HKMS09] and [Lau04], is to apply such a partition to the graph and precompute arc-flags for each edge that indicate for which target cells the edge is relevant in a shortest path query. A Dijkstra query then only has to relax edges that have the corresponding flag for the cell the target node inhabits set, which greatly reduces the search space. Note that as the search approaches its destination, more and more edges are likely to have the relevant flag set, and once the target cell has been reached the query has to take into account all edges again. This results in a distinctive *coning* of the search space.

The multi-level arc-flag algorithm expands on the basic concept by employing a multi-level graph partition instead, i.e., a family of partitions $\{C^0, C^1, ..., C^{L-1}\}$, such that $\forall_{l<L-1}\ C_i^l \in C^l\ \exists C_j^{l+1} \in C^{l+1} : C_i^l \subseteq C_j^{l+1}$, where L denotes the number of partition levels. The query algorithm then considers lower level arc-flags when the top level target cell has been reached, which counteracts the search space coning and makes speed-up possible for the latter part of the query. Figure 2 gives an example of such a multi-level arc-flag vector for the number of levels $L = 4$, with 8 top level cells and 4 cells on all lower levels.

level	3	2	1	0
flags	0 1 0 1 0 0 0 0	1 1 0 0	0 1 1 0	0 0 1 0

Figure 2: The multi-level arc-flag vector for an edge in a 4-level hierarchy.

Contraction SHARC adopts contraction from Highway Hierarchies, as introduced in [SS06]. The graph is contracted by bypassing nodes iteratively to reduce the hop count of shortest paths and thereby speed up the query. When a node v is removed, for each pair of edges (u, v) and (v, w) that is discarded a shortcut (u, w) is introduced. Note that not every such shortcut may be necessary, as it is possible that a shorter path from u to w already exists in the graph. If such a *witness path* is found the shortcut can safely be discarded.

2.1 SHARC-Routing

By and large, the SHARC preprocessing routine is organized as follows:

To start off with, the input graph is trimmed down to its maximal node induced subgraph of minimum node degree 2 (the 2-core). Any stripped nodes (1-shell nodes) have their arc-flags assigned directly, and are reattached after the last step of preprocessing. A multilevel partition is applied to the remainder.

Subsequently, shortcuts are inserted into the graph as nodes are bypassed in L iterations of a contraction routine, L being the number of partition levels. During each iteration step of this routine, nodes are discarded until no more pass the bypass criterion $\#shortcut \leq c \cdot (deg_{in}(n) + deg_{out}(n))$, where $\#shortcut$ is the prospective number of shortcuts to be inserted in case node n is removed, and c a tunable parameter. Also, a node is never bypassed if any shortcut with a hop number greater than $h := 10$ would have to be added, or if any of its neighbouring nodes is not contained in the same partition cell. The last iteration additionally adds *boundary shortcuts* between boundary nodes, i.e., nodes with at least one neighbour in a different cell.

During contraction, arc-flags are set tentatively for the introduced shortcuts and all edges on their respective represented paths. For every shortcut (u, v) the edges it skips have all flags set to true, except their first passed edge, i. e., the edge with tail node u. Such edges are only used in queries targeting their own cell and therefore have just this one flag set. The shortcut itself has the flag corresponding to its own cell set to false. A small example is given in figure 3.

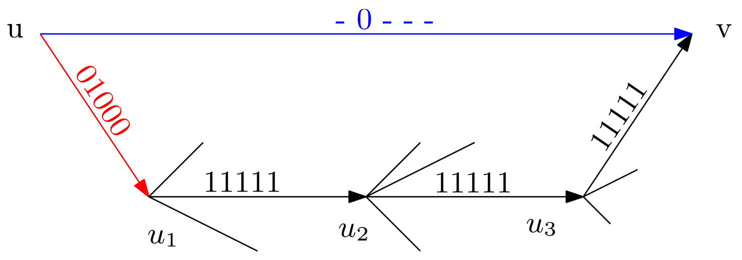


Figure 3: The first edge on shortcut (u, v)'s represented path has only the flag corresponding to its own cell set to true, while for the other edges on the path all flags are set. On the shortcut itself, the flag for its own cell is set to false.

A trailing edge reduction routine picks out any edges with no set arc-flags, which are redundant since they are not part of any shortest path, and eliminates them. Lastly, a refinement step enhances the arc-flags assigned by the contraction routine.

Note that this is a very abridged summary meant only for the introduction of the basic premise and terms this thesis is based on. For a more detailed explanation refer to [BD09].

2.2 Definitions

Node Level. The level $l(u)$ of a node u is defined by the step of the contraction routine it is bypassed in. 1-shell nodes are assigned level 0, nodes bypassed during step i of the contraction routine have level i . The maximum level $L+1$ is assumed by nodes which are never bypassed.

Arc Flag Vector. Given an edge (u, v), let $AF(u, v)$ denote the n-bit arc-flag vector of said edge and $F_i(u, v), i \in \{1, ..., n\}$ represent the individual flags.

$$F_i(u,v) = \begin{cases} 1, & \text{if the flag is set} \\ 0, & \text{otherwise} \end{cases},$$

where a higher value of i corresponds to a higher flag level $l(F_i(u, v))$. The level of an arc-flag is defined as the partition level its corresponding cell belongs to and ranges from 1 to L. $|AF(u, v)|$ denotes the total number of set flags.

Represented Path. The shortcut path $(u =: u_0, u_1, ..., u_k := v)$ of length k currently represented by a shortcut (u, v) in the graph is denoted as $P_{(u,v)}$. It may contain other shortcuts that have not been removed yet, as opposed to the completely expanded path. A small example for the represented path of a shortcut is given in figure 4.

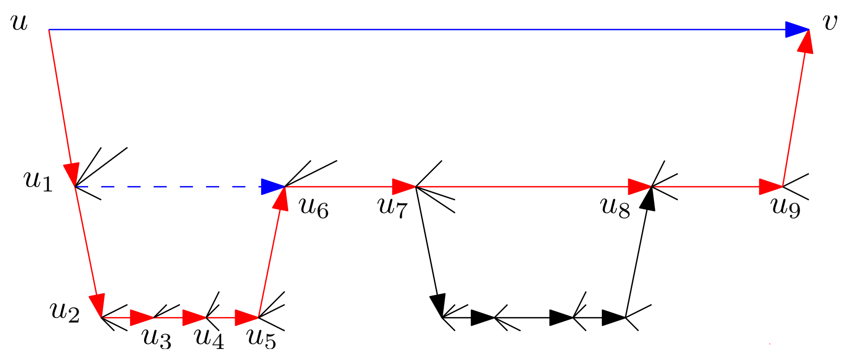


Figure 4: The currently represented path of shortcut (u, v) is marked in red. Shortcut (u_1, u_6) has already been removed, while shortcut (u_7, u_8) still remains in the graph, and is therefore part of (u, v)'s represented path.

3 Removing Shortcuts

As mentioned in the preliminaries, every shortcut represents a unique path in the graph, and when it is discarded the query can simply be rerouted along this way. A shortcut can be safely removed from the preprocessing, if all its set arc-flags are propagated to the first edge of their currently represented path. That this is sufficient to maintain correctness of the SHARC query is proven by the following theorem.

Theorem 3.1. *Given an arbitrary shortcut (u, v) with arc-flag vector $AF(u, v)$ and represented path $P_{(u,v)} = (u =: u_0, u_1, ..., u_k := v)$. Propagating all arc-flags set on (u, v) to the first passed edge (u, u_1) via $AF(u, u_1) = AF(u, u_1) \vee AF(u, v)$ allows removal of the shortcut from the preprocessing without affecting correctness of SHARC.*

Proof. Consider any shortest path from source s to target t that contains (u, v). When (u, v) is removed, the path $(s, ..., u, u_1, .., t)$ can be traversed by the SHARC query instead if the arc-flags for t are set on all path edges. For the subpath $(s, .., u)$ this is given by definition, since SHARC is correct and the flags have not changed for any of its edges. The same holds true for the subpath $(u_1, .., t)$, because SHARC does not permit negative edge weights and (u, v) therefore can't be part of it. Lastly, the correct flags are set for (u, u_1) as well, as these must have been set on (u, v) and therefore have been inherited by (u, u_1) during propagation. □

An example for the flag propagation is given in figure 5. This principle was introduced in [BD09], where a stripped version of SHARC was presented that disposed of shortcuts entirely. However, stripping all shortcuts from the preprocessing results in a significant coning of the search space, and consequently slows down the query considerably.

The main focus in the following is therefore on finding a better trade-off between space consumption and query performance by estimating a shortcut's importance for the eventual query and removing the ones deemed unimportant first. This will be the topic of the next chapter.

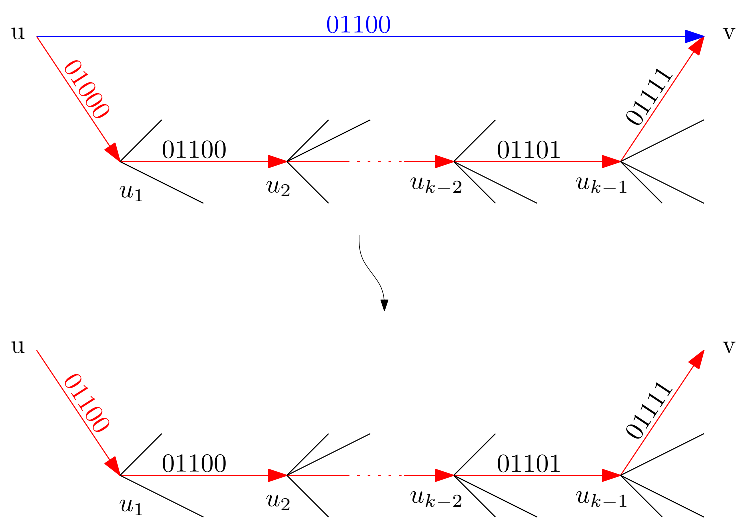


Figure 5: After the shortcut (u, v) is discarded, all its set arc-flags are propagated to the first edge on its represented path, (u, u_1).

4 Removal Strategies

This section introduces various heuristics to assess the impact a shortcut's removal from the graph might have on the SHARC query. All of these are based on specific shortcut properties, in particular the level of its head and tail node, set arc-flags and represented path. Since the removal of a shortcut might affect the properties of any shortcuts remaining in the graph, the order of removal is relevant. Shortcuts are therefore first assigned an evaluation value by the metric functions and then inserted into a priority queue, which allows for correct updating of the remaining keys whenever a shortcut is discarded. Upon removal of a shortcut from the graph, its set arc-flags are inherited by the first edge on its current represented path, which suffices to keep queries correct (c.f. section 3). The metrics presented in the following are organized into three large categories according to the abovementioned shortcut attributes they operate on, i.e., level information, arc-flags and represented path.

4.1 Level Information

A shortcut's evaluation value is defined by the level of its head or tail node, respectively, i.e., by the functions $level_{start\ node}(u, v) = l(u)$ and $level_{end\ node}(u, v) = l(v)$ for a shortcut (u, v). High level shortcuts can be considered more important for the query, as they are introduced into the hierarchy at a later point and might therefore be used more frequently, especially during long range queries. They may bridge a longer distance as well.

4.2 Arc-Flags

The four evaluation strategies presented in this subsection all take into account a shortcut's arc-flag vector. Since arc-flags are responsible for the major part of the speed-up achieved by SHARC, the number and level of a shortcut's set arc flags might give a good indication of its importance for the query.

4.2.1 Number Of Set Flags

This measure counts the number of set arc-flags on the shortcut. Each set flag is rated at a value of one, the sum over all set flags on a shortcut makes up its final value. It is denominated as

$$num_{set\ flags} = |AF(u,v)| = \sum_{i=1}^{n} F_i(u,v)$$

A shortcut with many set flags might be relevant for more queries than one with barely any flags set. Note that a shortcut's value may have to be updated whenever it happens to be the first passed edge of a shortcut removed at an earlier point, as the propagation might set additional flags. This applies for all subsequent flag-based metrics as well.

4.2.2 Level Of Set Flags

A shortcut is assigned the maximal level of any of its set flags as key value, more specifically:

$$max\ level_{set\ flags} = \max_{i\in\{1,..,n\}} (l(F_i(u,v)))$$

Set high level flags indicate a shortcut may skip a long distance. Also, a SHARC query usually climbs up to the top level of the hierarchy very quickly, and only descends once it has reached the target supercell. Hence, shortcuts with no set top level flags might not be relevant for as many queries.

4.2.3 Flag Cost Function

The flag cost function $cost_{set\ flags}(u,v)$ is a combination of the previous two metrics. Each flag $F_i(u,v)$ is attributed a function value $cost_{level}(l(F_i(u,v)))$ based on it's level, with the intention that high level flags are of greater importance. The final flag cost value for a shortcut is then comprised of their sum:

$$cost_{set\ flags}(u,v) = \sum_{i=1}^{n} cost_{level}(l(F_i(u,v)))$$

Figure 6 gives an example for $cost_{set\ flags}$ with the level cost function $cost_{level} = 2^i$.

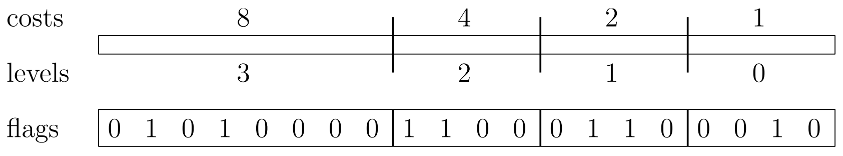


Figure 6: An example for the flag cost function with $cost_{level} = 2^i$. The shortcut's final value is $cost_{set\ flags}(u,v) = 2*8+2*4+2*2+1*1 = 29$.

Region Distance. Instead of relying solely on the level of a set flag for evaluation, it might be beneficial for the $cost_{level}$ function to also incorporate a measure for the relative distance of a set flag's corresponding cell from the cell the shortcut's start node inhabits. Set flags for distant cells might indicate a shortcut's importance for long range queries in particular. Since the computation of exact cell distances is rather complex, this measure has to settle for an approximation. For each top level cell a representing node is picked at random and distances between these representing nodes serve as an estimate. Furthermore, the region distance information is only used in the weighting of highest level flags, since these are considered the most important and usually make up the major part of an arc-flag vector.

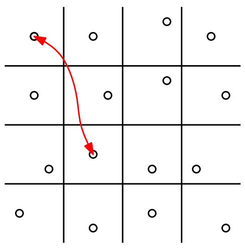


Figure 7: Region Distance Approximation.

4.3 Represented Path

This final batch of removal strategies evaluates the path $P_{(u,v)} = (u =: u_0, u_1, ..., u_k := v)$ represented by a shortcut (u, v) in the graph. A shortcut's represented path might provide a precise estimate on the search space coning its removal entails, based on path's length and the degree of the nodes it contains.

4.3.1 HopCount

A shortcut's hop number is defined as the length of its represented path. The corresponding evaluation function is denoted as $num_{hops}(u, v)$. Shortcuts with a high hop count might reduce the number of hops on shortest paths, and thereby speed up the query. Note that whenever a shortcut is removed, the evaluation value of all shortcuts whose represented path it was part of will increase, and consequently has to be updated.

4.3.2 Max Degree Of Nodes on Path

This metric values the maximal degree $max_{path\ degree} = \max_{i \in \{1,...,k-1\}}(d(u_i))$ of any node u_i on the represented path, excluding start and end node. A large node degree on the path indicates the shortest path query might cone more in case the shortcut is discarded, as it usually has to consider all outgoing edges of the nodes it reaches.

Correct updates for this term introduce some computational overhead, since a remaining shortcut's key in the queue might

- increase, if a removed shortcut was part of its represented path
- decrease, if a removed shortcut was incident to any of its path nodes, excepting start and end node

Therefore, on large graphs, a 'lazy' variant, which only updates shortcut values on the fly when they are removed from the queue, might be preferable. While this function can correct the queue key only in case of an increase, and is therefore no longer correct, it speeds things up significantly.

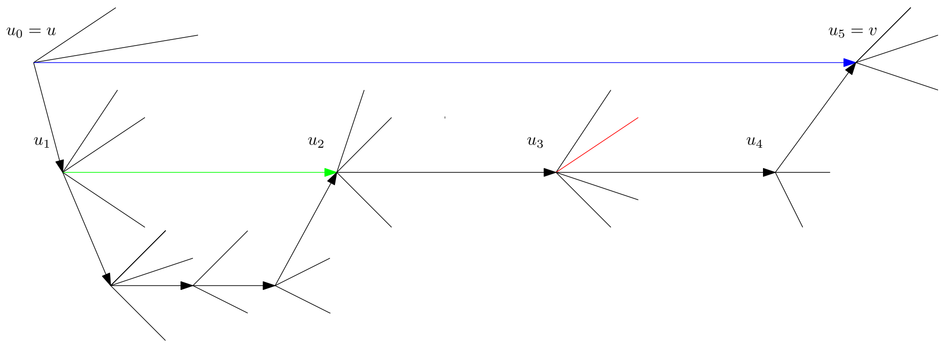


Figure 8: An example for $max_{path\ degree}$ updates. Removal of the shortcut marked in green will entail an increase of shorcut (u, v)'s key, while discarding the red edge will result in a key decrease.

4.3.3 Summed Degree Of Nodes on Path

This metric gives a rough indication of the search space increase induced by a shortcut's removal, and combines the two preceding measures as follows: For each node u_i on the represented path, excepting start and end node, the node degree $d(u_i)$ is computed. The key value is then determined by $sum_{path\ degree} = \sum_{i=1}^{k-1} d(u_i)$, the sum over all degrees. For updating, consequently, the same as for the $max_{path\ degree}$ function applies.

4.3.4 Search Space Coning Coefficient

Given a shortcut (u, v) with arc-flag vector $AF(u, v)$, number of set flags $|AF(u, v)|$ and represented path $P_{(u,v)} = (u =: u_0, u_1, ..., u_k := v)$ of length

k, the search space coning coefficient is defined as:

$$sscc(u,v) = \sum_{i=1}^{k-1} \Big(\sum_{\substack{(u_i,w)\in E \\ w \neq u_{i+1}}} cost_{flags}(AF(u_i,w) \wedge AF(u,w)) \Big),$$

where $cost_{flags}$ denotes the flag cost function introduced in 4.2.3. The intention here is that when a shortcut is discarded, a SHARC query will only consider alternate edges which have some of the same flags set as the shortcut. This metric therefore provides a more exact estimate on the search space increase than the simpler $sum_{path\ degree}$ function, which does not consider arc-flags at all.

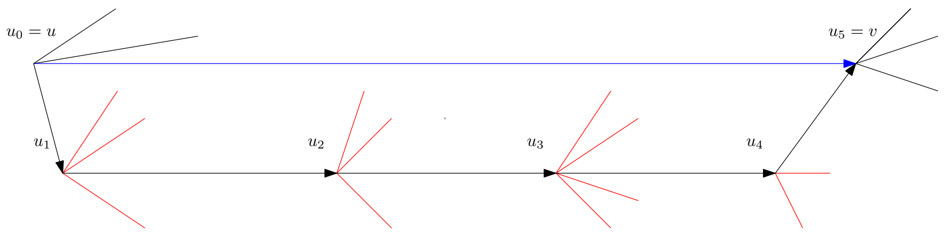


Figure 9: An example for the sscc metric. Only the edges marked red and the arc-flags set on shortcut (u,v) influence the computation of the evaluation value $sscc(u,v)$.

However, calculation and updates are more complex. A shortcut's queue value may

- increase, if

  a removed shortcut was part of its represented path, or

  a removed shortcut's first passed edge was incident to any of its path nodes, or

  a removed shortcut's first passed edge was the shortcut itself.

- decrease, if a removed shortcut was incident to any of its path nodes

For 'lazy' updates the same drawbacks and advantages as before apply.

4.4 Combinations

As each group of introduced metrics rate disjoint aspects of a shortcut, a weighted combination of the best performing functions in each category might prove advantageous. While the most straightforward approach is a simple linear combination of the individual factors, scaling and computation time pose significant problems. Therefore, an alternate solution is introduced in the following, which might be preferable, especially for the more computation-intensive metrics.

4.4.1 Weighting Function

Scale and range of the individual metrics's evaluation values differ widely. Moreover, no upper bound is known beforehand for any of the path based measures, since these increase (and decrease) unpredictably as more and more incidental shortcuts are discarded. This holds for the lazy and correct update strategies alike. Hence, scaling the individual function's results down to a common domain for a weighted evaluation function to operate on brings about considerable computational overhead, as well as unavoidable inaccuracies in the end result.

4.4.2 Precomputed Orders

The abovementioned drawbacks suggest a different approach, namely precomputing an order of removal on the set of shortcuts for each basic metric, and subsequently weighting on these orders instead of directly on the functions. However, this method does not allow for any reasonable updates of the combinate's key value at all, which is its predominant drawback. For each individual order, at least, correct updates are possible, especially since they only have to be computed once on each graph. This method is consequently significantly faster for the more complex metrics.

5 Experimental Results

This section provides an experimental survey on the performance of all introduced shortcut removal strategies, individually and in combination.

Setup. All experiments were conducted using one core of an AMD Opteron 2218 clocked at 2.6 GHz, on a testing system with 16 GB of RAM running SUSE Linux 11.1. The implementation of the shortcut removal routine was written in C++ and compiled with gcc 4.3.2, using optimization level 3.

Methodology. A metric's quality is judged based on the resulting increase in the amount of nodes settled by the SHARC query (i.e., the number of nodes removed from the Dijkstra's priority queue), and is normalised over the basis of the unmodified preprocessing. All results are depicted on a logarithmic scale, with the percentage of removed shortcuts on the x-axis and incurred scale factor of settled nodes on the y-axis.
Rating based on the average number of settled nodes during multiple queries, instead of their actual runtime, ensures the results are hardware-independent. This average is computed over 10000 random shortest path queries, which are executed at a fixed step of every 5% removed shortcuts (1% where stated).

Input. The standard input for all comparative tests is a SHARC preprocessing for the road network of Western Europe. More specifically, the generous variant, i.e., with arc-flags computation on all levels. For more details, in particular with regard to the differences between economical and generous SHARC, refer to [BD09]. Additional information on all studied preprocessings is listed in Table 1, in particular the number of shortcuts contained and the partitioning scheme.

preprocessing	#nodes	#edges	#shortcuts	partitioning
gen. SHARC Europe	18 010 173	53 311 151	12 168 696	4-4-4-4-8-104
eco. SHARC Europe	18 010 173	53 365 407	12 222 952	4-4-4-4-8-104
gen. SHARC Germany	4 692 091	13 414 319	2 722 029	4-4-4-4-112
eco. SHARC Germany	4 692 091	13 426 252	2 733 962	4-4-4-4-112

Table 1: SHARC preprocessing properties.

5.1 Individual Metrics

The following gives an overview on the relative performance of all metrics per category, as well as an account on which ones are relevant for combinations.

5.1.1 Level Based

By trend, it appears more beneficial to remove 'descending' shortcuts, i.e., those leading from high level to low level nodes, before touching the 'ascending' ones, since a long range SHARC query usually ascends quickly to a high level in the hierarchy and descends only very late. However, weighting the end node level more heavily than the start node level actually has no reliable positive effect (c.f. figure 10), as this makes it possible for shortcuts 'ascending' to a high level to be removed ahead of one 'descending' from an even higher level.

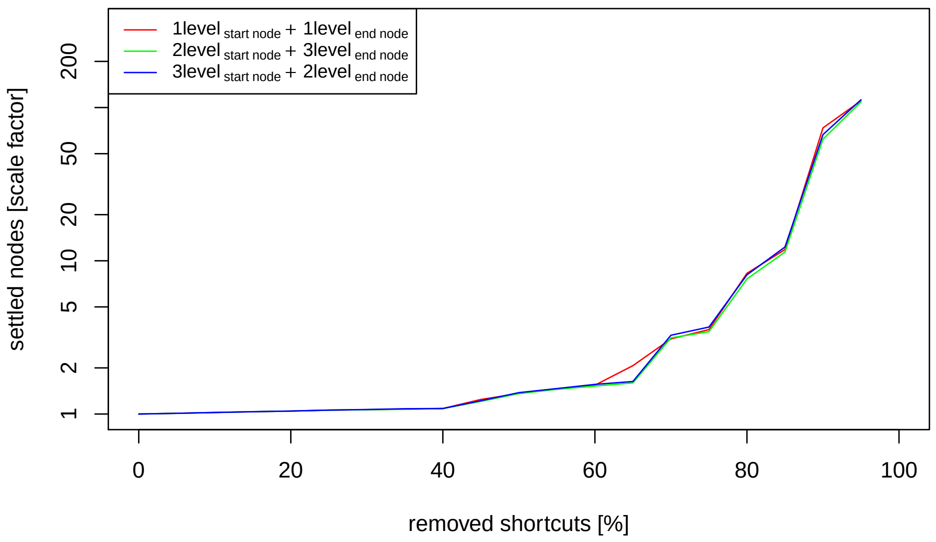


Figure 10: Weighting of start and end node level.

Therefore, in combinations, start and end node level are weighted with equal factors and subsequently treated as a singular shortcut level metric. On the generous SHARC preprocessing of Western Europe, this metric by itself allows for the removal of about 35% of all shortcuts without any impact on query performance, and of about 65% at an increase of settled nodes by only factor two.

Results for other preprocessings are given in table 2. While the overall trend remains the same, discarding shortcuts from the economical variant of SHARC generally impairs query performance to a much higher degree than is the case for the generous preprocessings. This may be mostly due the fact that the arc-flags set by economical SHARC are less accurate, and consequently the contribution of shortcuts to the speed-up is comparatively larger.

preprocessing	removed shortcuts					
	0%	20%	40%	60%	70%	80%
gen. SHARC Europe	818	855	888	1 261	2 529	6 772
eco. SHARC Europe	945	1 059	1 173	2 068	5 162	15 850
gen. SHARC Germany	544	606	652	1 235	3 186	5 725
eco. SHARC Germany	623	832	949	2 392	7 170	14 213

Table 2: Settled nodes for the node level metric on different preprocessings.

5.1.2 Flag Based

In Figure 11, the effect of different $cost_{level}$ functions on the flag cost metric is displayed. Overall, rating the flag level l exponentially provides the best results.

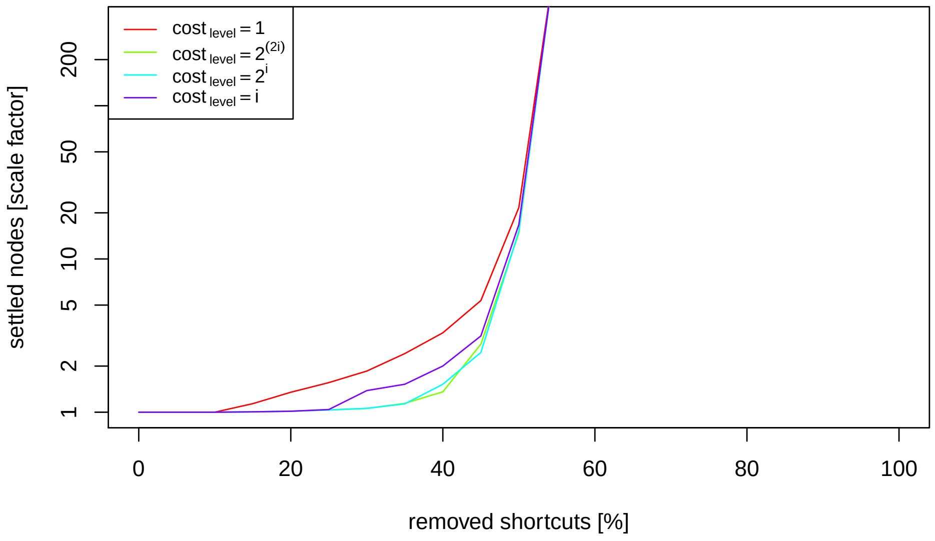


Figure 11: Different $cost_{level}$ functions for the flag cost metric.

The exponential function with base two, $cost_{level}(l) = 2^l$, is subsequently used as a norm (in particular for the *sscc* metric), since faster growing exponential functions, e.g. $cost_{level}(l) = 3^l$ and $cost_{level}(l) = 2^{2l}$, offer no noteworthy improvement.

Including the region distance approximation in the $cost_{level}$ function can further improve the flag cost metric's results slightly, as can be seen in table 3 for $cost_{level}(l) = 2^l$, and distance values of top level flags scaled to the range $[0,\ cost_{level}(L) - 1]$.

preprocessing		removed shortcuts					
		0%	20%	30%	40%	50%	60%
gen. SHARC Eur.	basic	818	830	867	1 245	12 438	1 495 730
	dist.	818	830	866	1 174	12 426	1 439 640

Table 3: $cost_{set\ flags}$ with and without region information.

Figure 12 compares all three flag based metrics on a logarithmic scale. The $max\ level_{set\ flags}$ metric actually surpasses the $cost_{set\ flags}$ function (with $cost_{level}(l) = 2^l$ using region distance estimate) after about half the shortcuts have been removed. In the relevant section, however, the flag cost metric clearly outperforms both its incorporated functions.

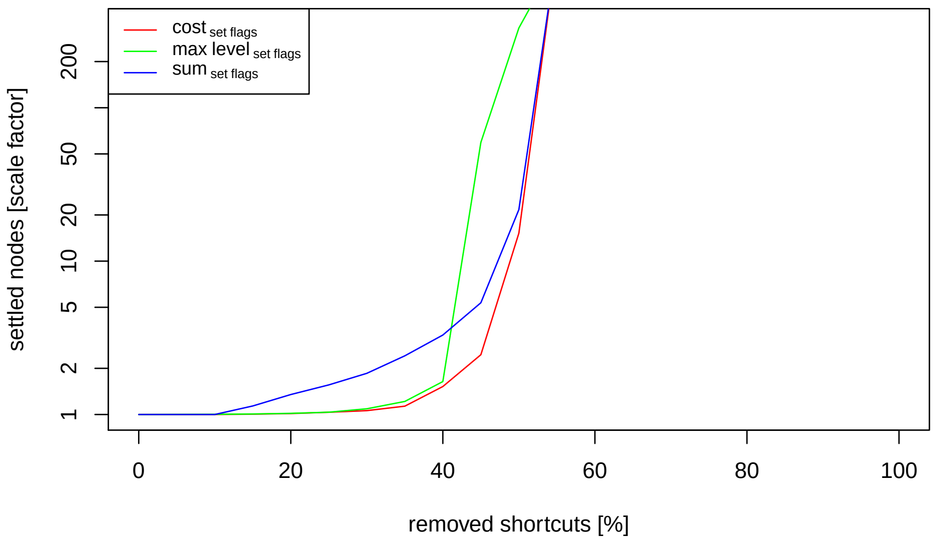


Figure 12: The three flag based metrics in comparison.

On the whole, when using the $cost_{set\ flags}$ metric about 40-45% of all shortcuts can be discarded at an increase in settled nodes by factor two. Table 4 shows the performance of the $cost_{set\ flags}$ metric on all preprocessings, which is somewhat worse for Germany than for Europe, and deteriorates notably for the economical variants. In particular on Germany, where the removal of only 30% of the shortcuts already doubles settled nodes for the query.

preprocessing	removed shortcuts					
	0%	20%	30%	40%	50%	60%
gen. SHARC Europe	818	830	866	1 174	12 426	1 439 640
eco. SHARC Europe	945	1 076	1 323	2 263	14 489	2 165 150
gen. SHARC Germany	544	563	679	1 229	61 036	445 827
eco. SHARC Germany	623	858	1 313	2 245	75 343	731 919

Table 4: $cost_{set\ flags}$ on different preprocessings.

These results are significantly worse than for the node level metric, which allowed for the removal of an additional 15% of the shortcuts at the same penalty.

5.1.3 Path Based

It turns out that the 'lazy' update strategy actually achieves better results than the correct one for the $sum_{path\ degree}$ metric and, to a much lesser degree, for the *sscc* function (c.f. Figure 13). This may be largely due to the fact that the removal of an edge incident to one of a shortcut's path nodes may lead to a decrease of this shortcut's key value for both functions, when it actually indicates an increase of the search space in case the shortcut is discarded. Since the 'lazy' variant only acknowledges key increases and ignores decreases completely, it amends this somewhat. The effect is further mitigated for the *sscc* metric, since all flags of a shortcut are propagated to its first passed edge when it is removed, and the decrease in value is therefore considerably smaller (i.e., the original $cost_{set\ flags}$ value of this first passed edge, which should not have many flags set).
Note that the region distance approximation is not applied for the $cost_{set\ flags}$ function used by the *sscc* metric, as it has no noticeable impact on the end results and slows down computation.

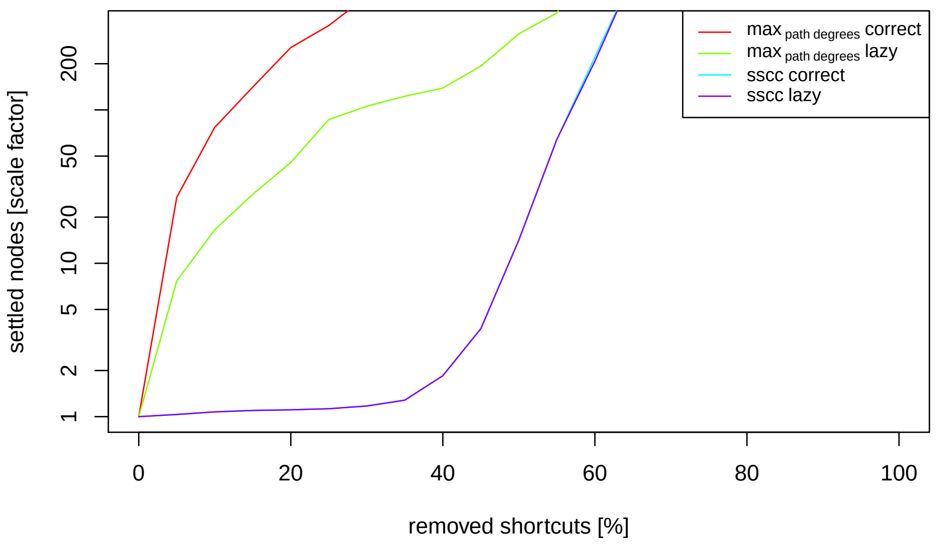


Figure 13: Lazy and correct updates for $sum_{path\ degree}$ and the search space coning coefficient in comparison.

A comparison of all path based metrics is given in Figure 14. As is to be expected, the advanced *sscc* metric (using level weight function $cost_{level} = 2^i$ without region distance approximation and lazy update strategy) outperforms the simpler path based measures by far. Also, it shows some improvement over the $cost_{set\ flags}$ metric, albeit not in the relevant section, as it only allows for the removal of about 35-40% of the shortcuts before the average amount of settled nodes increases by factor two.

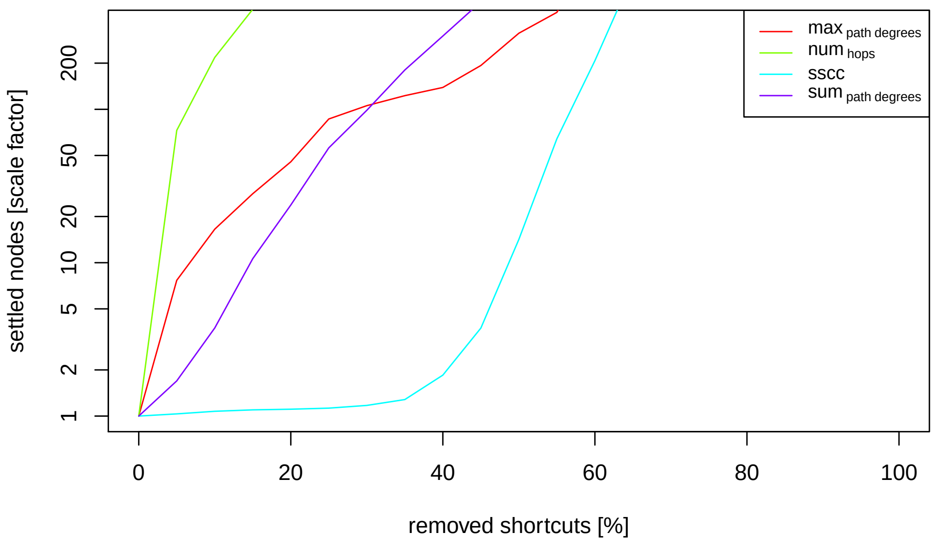


Figure 14: All path based metrics on a large scale

As is shown in table 5, the performance of the search space coning coefficient on other preprocessings is similar, although notably worse for the economical variants. The results are slightly better for Europe than for Germany.

preprocessing	removed shortcuts					
	0%	20%	30%	40%	50%	60%
gen. SHARC Europe	818	906	960	1 512	11 603	169 657
eco. SHARC Europe	945	1 158	1 450	3 006	19 233	516 761
gen. SHARC Germany	544	616	773	2 149	14 352	96 194
eco. SHARC Germany	623	906	1 414	3 422	31 135	217 813

Table 5: The *sscc* metric on different preprocessings.

5.1.4 Comparison

As can be seen from figure 15, the shortcut level metric outperforms all other measures by far, while also being the simplest and least computation intensive. It generally allows for the removal of about 15-20% more shortcuts than the best performing flag and path based metrics.

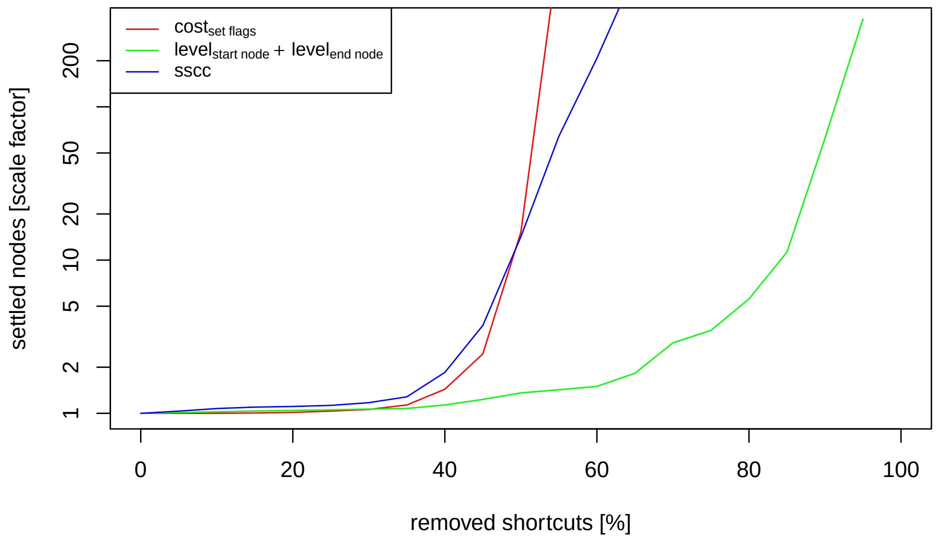


Figure 15: The best metrics of each category

It is also notable that, while the search space coning coefficient does manage to surpass it eventually, the flag cost function delivers arguably better results in the most relevant section.

5.2 Combinations

Since the node level metric showed the best performance, all of the combinations presented herein are aimed at improving this heuristic even further.

5.2.1 Node Level & Flag Cost

As stated before, the flag cost metric with $cost_{level}(l) = 2^l$ and region distance information for top level flags is used. Figure 16 shows the best results for a weighted combination of the precomputed orders for $cost_{set\ flags}$ and the node level metric. The optimum is reached by a ratio of about 4:1, and manages to improve the shortcut level order metric somewhat.

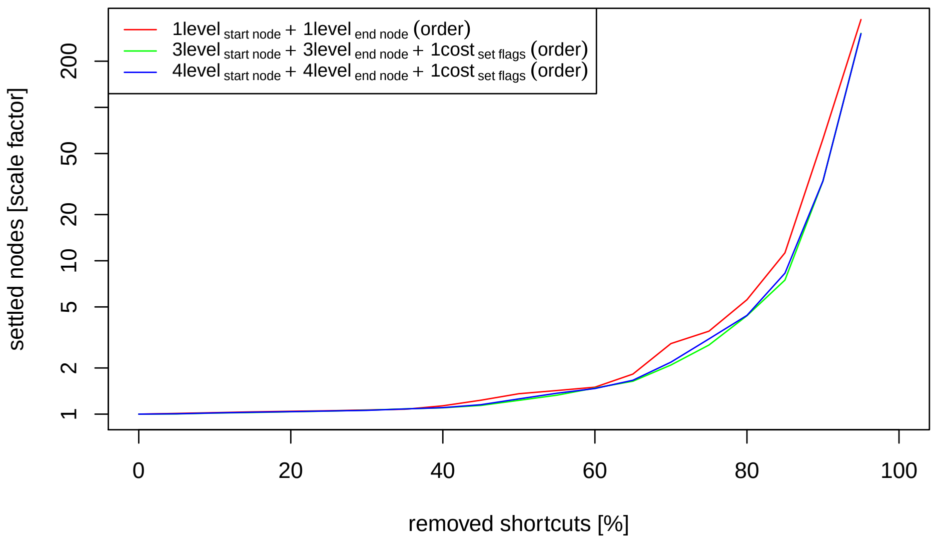


Figure 16: Order based combination of shortcut level and flag cost metric

For direct weighting, the domains of the $level_{start\ node}$ and $level_{end\ node}$ metrics are each scaled to the maximum of the $cost_{setflags}$ function. The results (c.f. figure 17) are more impressive than for the order based combination. At a ratio of 5:1, the weighted evaluation function performs much better than the shortcut level metric by itself, and allows for the removal of more than 70% of all shortcuts before an increase of settled nodes by factor 2.

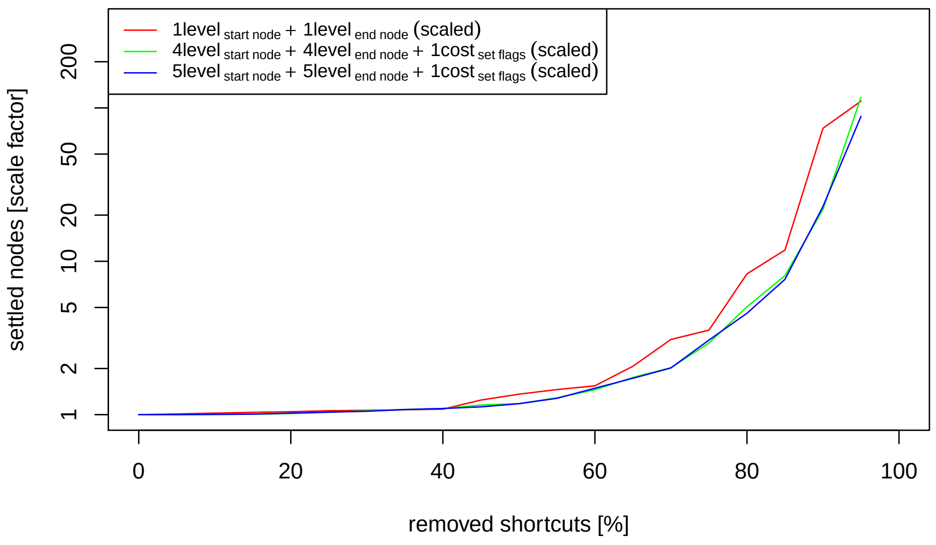


Figure 17: Direct combination of shortcut level and flag cost metric

Table 6 lists the results for order based and direct combination of the node level and flag cost metrics on other preprocessings. The improvement is notable for both the economical and generous variants alike (c.f. section 5.1.1, table 2 for the results achieved by the node level metric on its own), although the performance on the preprocessing for Germany is significantly worse than on Europe.

preprocessing		removed shortcuts					
		0%	20%	40%	60%	70%	80%
gen. SHARC Eur.	scaled (5:1)	818	837	897	1 216	1 649	3 748
	order (4:1)	818	849	902	1 201	1 785	3 600
eco. SHARC Eur.	scaled (5:1)	945	1 033	1 328	2 500	3 483	10 073
	order (4:1)	945	1 051	1 242	1 990	3 844	10 018
gen. SHARC Ger.	scaled (5:1)	544	570	647	1 108	1 927	4 786
	order (4:1)	544	601	654	1 135	1 874	4 814
eco. SHARC Ger.	scaled (5:1)	623	778	931	2 152	5 209	10 034
	order (4:1)	623	826	1 001	2 232	4 888	10 378

Table 6: Node level and flagcost combination results on different preprocessings.

5.2.2 Node Level & Search Space Coning Coefficient

Order based combination of the node level and sscc metrics did not turn out well in comparison (c.f. figure 18). Factoring in the *sscc* order leads only to a very slight improvement in the most interesting section, which is, however, barely noticeable on a large scale.

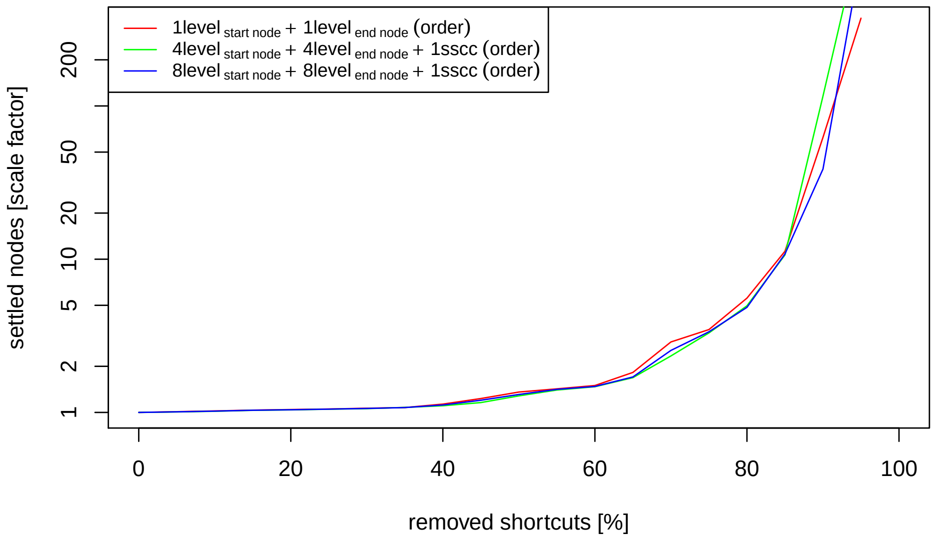


Figure 18: Order based combination of shortcut level and sscc metric

The results grow drastically worse over the 85-90% marks, where the combinate's performance declines more rapidly then the node level metric's on its own.

With direct combination slightly better results can be achieved (c.f. figure 19). The node level metrics' domains are here scaled to a precomputed maximum of the search space coning coefficient. The improvement is only notable after settled nodes have already increased by factor five, however, and therefore does not rival the results achieved by combination with the flag cost function.

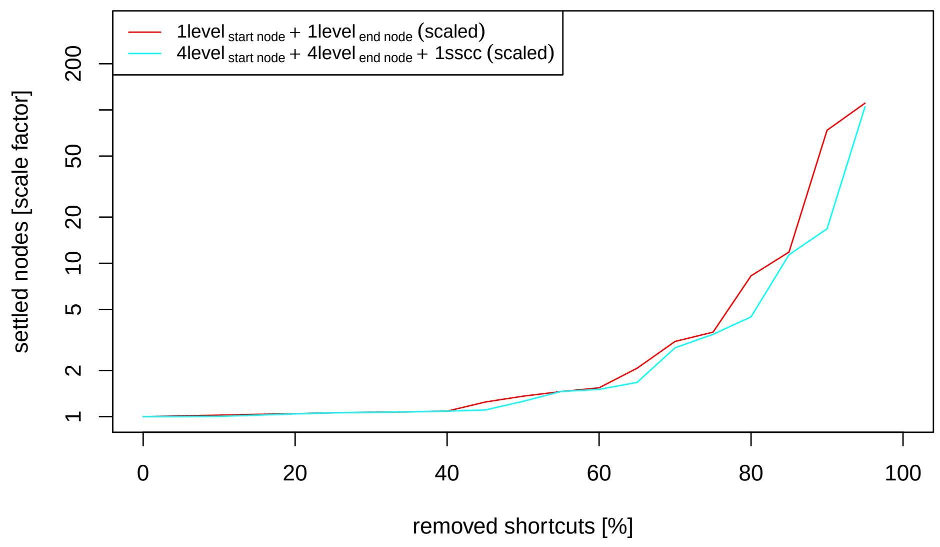


Figure 19: Direct combination of shortcut level and sscc metric

5.3 Best Results

As mentioned before, the best performance of any heuristic on its own is delivered by the shortcut level metric. Using this metric, about 30-40% of all shortcuts can be discarded without any notable loss in query performance, and about 60-65% at an increase in settled nodes by factor two. A weighted combination of the node level metric with the flag cost function improves on this, and generally allows for an additional 5% of the shortcuts to be removed at the same penalties.

6 Conclusion

This student research project evaluated different strategies of determining a shortcut's relative importance for the eventual SHARC query, in order to minimize space consumption of a SHARC preprocessing while retaining fast query times. The presented heuristics exploited three specific shortcut properties, i.e., node level information, set arc flags and represented path. While being the least computation intensive, the node level information based metric surpassed the best performing heuristics in the other two categories by far. A weighted combination with the predominant flag based metric improved even further on these results, while barely increasing the computational effort. This combination allows for the removal of about 60-70% of all shortcuts at the relatively small penalty of doubling the average amount of nodes settled by a SHARC query on almost all tested SHARC preprocessings.

Future Work. Within the scope of this student thesis experimental evaluation was only done for the static variant of SHARC. Since time-dependent SHARC (c.f. [Del09]) operates on complex edge-weight functions, shortcuts here make up a comparatively larger part of the space overhead. Therefore, it might be especially interesting to see how well the metrics presented herein can be applied in a time-dependent scenario.

Further space reduction for SHARC might be achieved by employing a more space-efficient representation of the graph data itself, such as the one introduced in 'An Experimental Analysis of a Compact Graph Representation' by Daniel K. Blandford, Guy E. Blelloch and Ian A. Kash (c.f. [BBK04]).

References

[BBK04] Daniel K. Blandford, Guy E. Blelloch, and Ian A. Kash, *An Experimental Analysis of a Compact Graph Representation*, Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04), SIAM, 2004, pp. 49–61.

[BD09] Reinhard Bauer and Daniel Delling, *SHARC: Fast and Robust Unidirectional Routing*, ACM Journal of Experimental Algorithmics **14** (2009), 2.4–2.29, Special Section on Selected Papers from ALENEX 2008.

[Del09] Daniel Delling, *Time-Dependent SHARC-Routing*, Algorithmica (2009), Special Issue: European Symposium on Algorithms 2008.

[DSSW09] Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner, *Engineering Route Planning Algorithms*, Algorithmics of Large and Complex Networks (Jürgen Lerner, Dorothea Wagner, and Katharina A. Zweig, eds.), Lecture Notes in Computer Science, vol. 5515, Springer, 2009, pp. 117–139.

[Gem08] Andreas Gemsa, *Arc-Flag Compression*, 2008, Student Reseach Project.

[HKMS09] Moritz Hilger, Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling, *Fast Point-to-Point Shortest Path Computations with Arc-Flags*, The Shortest Path Problem: Ninth DIMACS Implementation Challenge (Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, eds.), DIMACS Book, vol. 74, American Mathematical Society, 2009, pp. 41–72.

[Lau04] Ulrich Lauther, *An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background*, Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung, vol. 22, IfGI prints, 2004, pp. 219–230.

[SS06] Peter Sanders and Dominik Schultes, *Engineering Highway Hierarchies*, Proceedings of the 14th Annual European Symposium on Algorithms (ESA'06), Lecture Notes in Computer Science, vol. 4168, Springer, 2006, pp. 804–816.

[SSV08] Peter Sanders, Dominik Schultes, and Christian Vetter, *Mobile Route Planning*, Proceedings of the 16th Annual European Symposium on Algorithms (ESA'08), Lecture Notes in Computer Science, vol. 5193, Springer, September 2008, pp. 732–743.