Engineering Multi-Level Overlay Graphs for Shortest-Path Queries

MARTIN HOLZER, FRANK SCHULZ, and DOROTHEA WAGNER
University of Karlsruhe

An overlay graph of a given graph $G = (V, E)$ on a subset $S \subseteq V$ is a graph with vertex set S and edges corresponding to shortest paths in G. In particular, we consider variations of the multi-level overlay graph used in [Schulz et al. 2002] to speed up shortest-path computation. In this work, we follow up and present several vertex selection criteria, along with two general strategies of applying these criteria, to determine a subset S of a graph's vertices. The main contribution is a systematic experimental study where we investigate the impact of selection criteria and strategies on multi-level overlay graphs and the resulting speed-up achieved for shortest-path computation: Depending on selection strategy and graph type, a centrality index criterion, selection based on planar separators, and vertex degree turned out to perform best.

Categories and Subject Descriptors: G.2.2 [**Graph Theory**]: Graph algorithms

General Terms: Algorithms, Experimentation, Measurement, Performance

Additional Key Words and Phrases: Dijkstra's algorithm, hierarchical, multi-level, overlay graph, preprocessing, shortest path, speed-up technique, vertex selection

1. INTRODUCTION

Given a graph and a subset of its vertices, an *overlay graph* describes a topology defined on this subset, where edges correspond to paths in the underlying graph[1]. E.g., consider as base graph the internet graph representing connections between hosts and as overlay graph the topology of a peer-to-peer network. Depending on the application, overlay graphs are demanded to fulfill certain requirements, such as high connectivity and reliability in the case of peer-to-peer networks. Another use case requires that shortest-path lengths in the original graph carry over to the overlay graph.

Following the *multi-level (graph) approach*, or *multi-level technique*, introduced in [Schulz et al. 2002], a method to speed up exact single-pair shortest-path computation, we restrict ourselves to overlay graphs preserving shortest-path lengths.

Authors' address: Universität Karlsruhe (TH), Fakultät für Informatik, Postfach 69 80, 76128 Karlsruhe, Germany. Email: {mholzer,wagner}@ira.uka.de, frank.schulz@ptv.de.

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

[1]We use the term "overlay graph" in this general sense, while sometimes it is used only in a specific context, e.g., associated with certain properties like uniform vertex degree.

With the multi-level approach, one or more levels of overlay graphs which inherit shortest-path lengths from the base graph are constructed. Then a shortest-path computation takes place in a graph consisting basically of one of the overlay graphs and some additional edges.

We now further refine this approach by introducing a simple procedure to compute overlay graphs that preserve shortest paths—also referred to by *shortest-path overlay graphs*—and additionally have a *minimal* number of edges. Analogously to [Schulz et al. 2002], we can iterate the process of determining shortest-path overlay graphs to make use of a whole *hierarchy* of such graphs for shortest-path computation. In other words, the difference between multi-level graphs from [Schulz et al. 2002] and (a hierarchy of) shortest-path overlay graphs due to this work is that the latter may contain fewer edges and thus tend to perform even better when used for answering shortest-path queries. For the sake of convenience, however, and since here we are dealing only with shortest-path overlay graphs in the new (i.e., minimal) sense, we will often call (hierarchies of) shortest-path overlay graphs also multi-level overlay graphs, or multi-level graphs for short, unless any ambiguity may arise.

It is important to note that multi-level graphs as just described (both minimal and non-minimal ones) additionally have edges passing between different levels of a given hierarchy; we therefore call them *extended*. In this work, we also introduce a *basic* variant, which renounces those additional edges.

In the theoretic part, we identify a class of graphs for which the multi-level approach works provably well. We give bounds on the size of a corresponding multi-level graph and the asymptotic size of the search space when the multi-level graph is used for shortest-path computation. According to these results, it is crucial for the effectivity of our technique that the sets of selected vertices be rather small and—informally speaking—decompose the graph in a balanced way. Moreover, we show how to construct random graphs that allow for a decomposition into balanced components through few vertices.

In [Schulz et al. 2000], it is shown that the multi-level approach with *one* additional level can be successfully applied to public-transportation networks; [Schulz et al. 2002] then provides a generalization to *multiple* levels. In both studies, application-specific information is used to determine the vertex subsets. In this work, however, we focus on getting along without information from the application underlying the graph. To this end, we give general criteria and two overall strategies for selecting the vertices upon which an overlay graph is to be built.

In an extensive experimental study we investigate the impact of the choice of parameters involved in our model—first of all selection criterion/strategy—on the quality of the resulting multi-level graphs by measuring their performance when used for shortest-path computation (average *speed-up*, i.e., the number of edges 'visited' during shortest-path search with Dijkstra's algorithm compared to the multi-level technique). To underpin the significance of our experiments, we further consider different real-world and generated graphs. It turns out that with a *global* selection strategy, vertex degree is a good criterion. The best results, however, are achieved with a *recursive* strategy based on either centrality index or planar-separator criteria.

Structuring. The paper is organized as follows. The rest of this section is devoted to classifying our technique in the context of related shortest-path approaches. In Section 2, shortest-path overlay graphs are formally defined and a construction algorithm is given; we further contrast the basic and extended variants of multi-level graphs. Their application to speed up shortest-path computation is shown in Section 3, where we first need to define an auxiliary data structure employed. In Section 4, we theoretically analyze multi-level graphs of some regular structure and introduce a model for random graphs that allow for such regular-shaped multi-level graphs. Section 5 first presents the different ways of obtaining vertex subsets, the graphs used with our experiments, and prestudies taking a closer look at some of the selection criteria, followed by the main empiric study. We conclude our work in Section 6 by summarizing the obtained results.

1.1 Related Work

There are numerous approaches to speed up single-pair shortest-path computation (cf. [Willhalm and Wagner 2007] for a survey), where most of them improve on Dijkstra's algorithm [Dijkstra 1959]. A few speed-up techniques can be applied immediately, e.g., goal-directed and bidirectional search [Ahuja et al. 1993]. However, much better speed-up factors are reached when some precomputed information can be used. Since in most scenarios both time and storage requirements inhibit advance computation of all shortest paths, such approaches represent a trade-off between precomputational effort and resulting average speed-up. Also, many combinations of speed-up techniques have proven successful [Delling et al. 2007; Goldberg et al. 2006; Holzer et al. 2004; Schulz et al. 2000].

Preprocessed information can be employed to guide the shortest-path search toward the target [Goldberg and Harrelson 2005], or to prune the search space at vertices that are known not to form part of a shortest path requested [Gutman 2004; Lauther 2004; Möhring et al. 2005; Wagner and Willhalm 2003]. Another usage, also followed in this paper, is precomputing an auxiliary graph in which shortest-path calculations take place (mostly, the original graph is enriched with additional edges corresponding to shortest paths). We now discuss several approaches of the latter kind and point out their relationship to ours.

Hierarchical Encoded Path Views. In [Jing et al. 1998], shortest-path computation in the context of navigation systems is studied, where there is also the need for efficiently maintaining shortest paths themselves, or path views—as opposed to the mere distance. The input graph is fragmented into connected components using spatial information, and the 'border vertices' thus obtained play a similar role as our selected vertices. Roughly speaking, the auxiliary graph is made up of partial graphs each of which keeps all-pairs shortest-path information for one fragment.

HiTi Graphs. HiTi graphs introduced in [Jung and Pramanik 2002] are applied in the area of car navigation. The basic difference to our approach is that the input graph is decomposed into connected components by edge instead of vertex separators. For each component, a complete graph on the 'boundary vertices' stores the shortest-path lengths between these vertices (to this end, also shortest-path overlay graphs would be appropriate).

Highway Hierarchies. In [Sanders and Schultes 2005; 2006], the auxiliary graph is computed by first determining a neighborhood of each vertex. Then a level of so-called highway edges is introduced, i.e., edges (v, w) for some shortest path $(u, \ldots, v, w, \ldots, x)$ such that v is not in the neighborhood of x and w not in that of u. This level, which can be regarded as a (not necessarily minimal) shortest-path overlay graph, is compressed such that low-degree vertices are removed, and iteratively further levels can be constructed. Shortest-path computation for a given pair of vertices starts a bidirectional search in the input graph, and switches to edges of higher levels as the distance to source and target, respectively, grows. This approach is successfully applied to road graphs.

High-Performance Multi-Level Graphs. In [Delling et al. 2007], our multi-level technique is further developed in that greater emphasis is placed on preprocessing; it is also closely related to Hierarchical Encoded Path Views. This variation relies on the extended version of multi-level graphs, with one crucial difference to the definition in the present work: roughly, an edge is introduced between each pair of selected vertices adjacent to different components (according to the decomposition through a given set of vertices). This modification leads to a large increase in the number of edges, but permits during shortest-path search to require only a constant number of 'hops.' Another distinct feature is that many partial graphs, or search space parts, are kept rather than one whole multi-level graph. This allows for individual optimization of each part, which was shown to reduce the overall amount of edges with road graphs considerably.

Transit Node Routing. A similar idea is exploited in [Bast et al. 2007], which was developed independently of our approach: One basic observation is that from a fixed vertex, virtually all long-distance shortest paths leave a local area around it through a small number of 'important' vertices, called transit nodes. An exhaustive precomputation determines the distances from vertices to their local transit nodes (and vice versa) as well as between all pairs of transit nodes and stores them in the form of tables. A shortest-path search then only requires a few table lookups.

Dynamic Aspects. Finally, we want to mention two papers that propose related procedures for dynamic settings. In [Bauer 2006], our multi-level approach is revisited in the realm of edge updates: It is shown basically that only those parts of the multi-level graph have to be recomputed which belong to components (due to the decomposition through selected vertices) affected by an update.

A very recent study [Schultes and Sanders 2007] considers basic multi-level graphs as defined in the work at hand but with selected vertices taken from a Highway Hierarchies precomputation. The most conspicuous difference is that large parts of the input graph may not be decomposed into different components any more. This leads to an alteration of the search algorithm: The search starts in the original graph; when a certain portion of the local area around the source (and the target, respectively) has been settled, the next-higher level of the hierarchy of overlay graphs is factorized into the search. Unlike with our algorithm, edges of both the original and overlay graphs may hence be considered simultaneously.

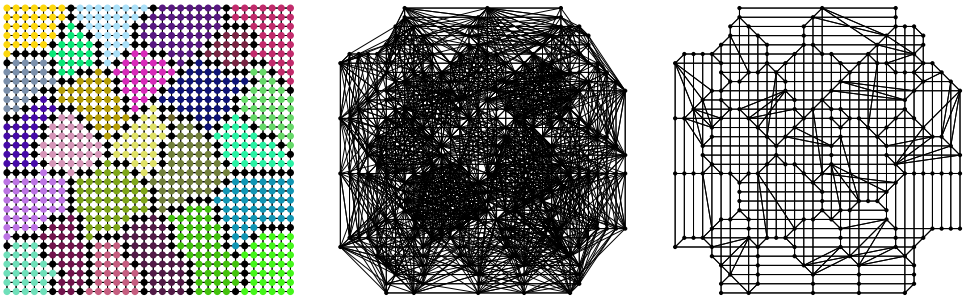

Fig. 1. From left to right: decomposition of a 32×32 grid graph through a separator of 208 vertices (black dots); a belonging non-minimal shortest-path overlay graph (according to [Schulz et al. 2002]) with 1994 edges; the corresponding minimal shortest-path overlay graph with 538 edges.

2. OVERLAY GRAPHS

In this section, we define shortest-path overlay graphs, by which we mean overlay graphs that inherit shortest-path lengths and have a minimal number of edges, and sketch a construction algorithm. Iterative application yields a hierarchy of shortest-path overlay graphs, or *basic multi-level graph*; by adding some further edges, we obtain the *extended* variant, corresponding to the multi-level graph from [Schulz et al. 2002] and [Holzer 2003], except that now minimality is guaranteed.

Notation. For the remainder of this work, let $G = (V, E)$ be a directed, connected graph with a positive edge length ℓ_e for each edge $e \in E$. Unless stated otherwise, n denotes the number of vertices and m the number of edges in G.

2.1 Shortest-Path Overlay Graph

For a subset $S \subseteq V$ we seek a graph G' with vertex set S and shortest-path lengths inherited from G: for each pair of vertices $u, v \in S$, shortest u-v-paths have equal length in G and G'. Additionally, we want G' to contain a minimal number of edges (to keep the search space the smallest possible when G' is used for shortest-path computation). We formally define shortest-path overlay graphs and prove that the definition meets the above requirements; finally, we outline an algorithm `min-overlay` that constructs shortest-path overlay graphs.

Definition 2.1. Given a graph $G = (V, E)$ and a subset $S \subseteq V$, the *shortest-path overlay graph* $G' := (S, E')$ is defined as follows: for each $(u, v) \in S \times S$, there is an edge (u, v) in E' if and only if for *every* shortest u-v-path in G no internal vertex belongs to S (internal vertices are all vertices on the path except for u and v). The length of (u, v) is set to the shortest-u-v-path length in G.

Note that in [Schulz et al. 2002] a different condition for $(u, v) \in E'$ is used: For each vertex $u \in S$, a shortest-path tree T_u is computed. Then an edge (u, v) is added to E' if the u-v-path in T_u contains no internal vertex in S—however, there may exist another shortest u-v-path in G with an internal vertex in S so that G' contains redundant edges. Figure 1 depicts two overlay graphs of a grid, one computed by the procedure suggested in [Schulz et al. 2002], the other being the minimal overlay graph computed by the subsequent procedure `min-overlay`.

THEOREM 2.2. *Given a graph $G = (V, E)$. A shortest-path overlay graph $G' = (S, E')$ of G according to Definition 2.1 inherits shortest-path lengths from G, and the number $|E'|$ of edges is minimal among all graphs with vertex set S and inherited shortest-path lengths. Moreover, G' is the unique overlay graph under these constraints.*

PROOF. We first show that for every $s, t \in S$, shortest s-t-paths in G and in G' are of equal length. Let p be such a shortest s-t-path in G. Consider the subpaths $p_1, \ldots, p_k$ of p divided at all vertices in S (i.e., the first and the last vertex of each p_i are in S and no internal vertex of p_i belongs to S). For each subpath $p_i = (w_1, \ldots, w_l)$, one of two cases occurs: either every other shortest path from w_1 to w_l in G also has no internal vertex in S, so there is an edge (w_1, w_l) in E'. Or there is a shortest path from w_1 to w_l in G via some vertex $x \in S$, in which case we replace p_i with two subpaths $p_i' = (w_1, \ldots, x)$ and $p_i'' = (x, \ldots, w_l)$; since this can happen only a finite number of times, we get a division of p into subpaths each of which has a corresponding edge in the overlay graph G'. Hence, there is also an s-t-path in G' and by construction, the lengths correspond.

To prove minimality and uniqueness of G', assume that there is an overlay graph $G'' = (S, E'')$ with shortest-path lengths inherited from G. Further, let (u, v) be an edge in E' but not in E'', and $(u = w_1, \ldots, w_k = v)$ be a shortest u-v-path in G'' (it holds that $k > 2$ because $(u, v) \notin E''$). Since subpaths of shortest paths are also shortest, each (w_i, w_{i+1}) corresponds to a shortest w_i-w_{i+1}-path of equal length in G. Hence, there must be a shortest path $(u = w_1, \ldots, w_2, \ldots, w_{k-1}, \ldots, w_k = v)$ in G, where some internal vertices are in S, in contradiction to $(u, v) \in E'$. □

We now jot down the construction algorithm `min-overlay`, which strongly relies on Definition 2.1.

Procedure `min-overlay`(G, ℓ, S)

For each vertex $u \in S$, run Dijkstra's algorithm on G with pairs (ℓ_e, σ_e) as edge weights, where $\sigma_e := -1$ if the tail of edge e belongs to $S \setminus \{u\}$, and $\sigma_e := 0$ otherwise. Addition is done pairwise, and the order is lexicographic. The result of Dijkstra's algorithm are distance labels (ℓ_v, σ_v) at the vertices, where $(\ell_u, \sigma_u) := (0, 0)$ in the beginning. For each $v \in S \setminus \{u\}$ we introduce an edge (u, v) in E' with length ℓ_v if and only if $\sigma_v = 0$.

The algorithm can be implemented with cost in $\mathcal{O}(|S| \cdot (|E| + |V| \log |V|))$ using Fibonacci heaps. Note that the Dijkstra search can be terminated when $\sigma_v < 0$ for all vertices v in the queue since for vertices w not yet labeled at that point in time, it cannot hold true that $\sigma_w = 0$ (this heuristically improves the running time).

2.2 Basic Multi-Level Graph

By iteratively applying the `min-overlay` procedure with a sequence of subsets $S_1 \supseteq S_2 \supseteq \ldots \supseteq S_l$ of V we obtain a hierarchy $G_i = (S_i, E_i)$ of shortest-path overlay graphs (for some $l \geq 1$). Together with $G_0 = (V_0, E_0) := G$, we call this collection of shortest-path overlay graphs, also denoted by $\mathcal{M}(G; S_1, \ldots, S_l)$, a *basic multi-level graph* of G with $l + 1$ levels. We also refer to G_i as *level i*, and call a vertex v a *level-i vertex* if i is the highest index such that $v \in S_i$.

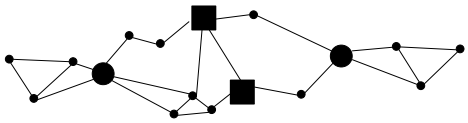

Fig. 2. Sample graph with vertex selections S_1 (big disks and squares) and S_2 (squares). Edge lengths are uniform. For the sake of clarity, edge directions are not reflected in the drawing.

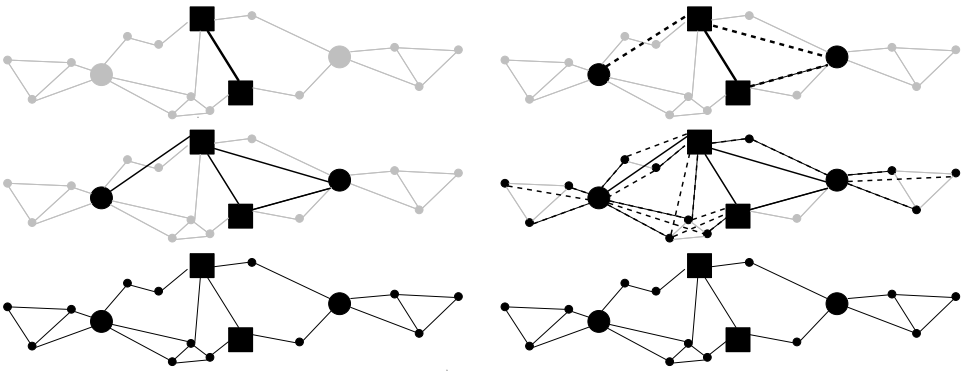

Fig. 3. Different levels (0 to 2, from bottom to top) of the basic (left) and extended (right) multi-level graph belonging to the graph from Figure 2. Level edges are drawn solid, up- and downward edges are dashed; edge weights are not respected any more.

2.3 Extended Multi-Level Graph

The definition of *extended multi-level graph*s corresponds to that of the basic variant except that each level $i \geq 1$ contains two additional sets of edges: *upward edges*, U_i, from vertices in $S_{i-1} \setminus S_i$ to vertices in S_i, and *downward edges*, D_i, from vertices in S_i to vertices in $S_{i-1} \setminus S_i$. For each edge in U_i and D_i, an analogous condition to that from Definition 2.1—viz., the respective path in G_{i-1} must not contain an internal vertex from S_{i-1}—is fulfilled. The edges in E_i—where E_0 is included—are also called *level edges*.

Thus, an extended multi-level graph with $l + 1$ levels is the collection of the enriched shortest-path overlay graphs $G_i = (V_i, E_i \cup U_i \cup D_i)$ together with $G_0 := G$. We also use $\overline{\mathcal{M}}(G; S_1, \ldots, S_l)$ as a notation. Extended multi-level graphs are equal to *multi-level graphs* as introduced in [Schulz et al. 2002] except that now, due to the altered condition for edges to be included in the sets E_i, U_i, and D_i, minimality of these edge sets is guaranteed.

The procedure `min-overlay` can be extended to construct also downward and upward edges: In the last step, where new edges are added, we now consider also vertices $v \in V \setminus S$ and introduce a downward edge (u, v) if and only if $\sigma_v = 0$. To construct upward edges, we run Dijkstra's algorithm also from vertices $u' \notin S$ and introduce an edge (u', v') to a vertex $v' \in S$ if and only if $\sigma_{v'} = 0$.

Figure 2 shows a sample graph with a sequence of selected vertices of length 2 and Figure 3 the belonging basic and extended multi-level graphs.

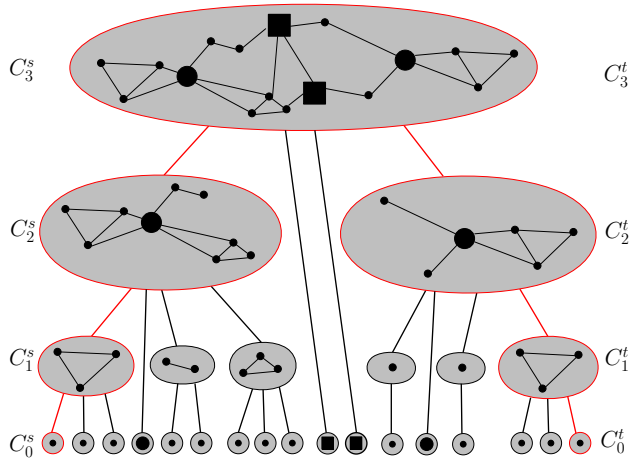

Fig. 4. Tree of connected components to the sample graph from Figure 2, with given vertices s and t. The C_0^s-C_0^t-path is marked in red ($k = k' = 1$ and $L_{st} = 3$).

3. SHORTEST-PATH SEARCH

In this section we show how to use multi-level graphs to speed up single-pair shortest-path algorithms. Depending on the given source and target vertices, a subgraph of the multi-level graph is determined on which a shortest-path search is run. Since our approach essentially defines another graph used as input for the actual computation, any shortest-path algorithm can be used to perform the search. This is also the reason for which combinations with other speed-up techniques that do or do not rely on precomputed information are feasible (cf. [Holzer et al. 2004]).

We first revisit the definition of an auxiliary data structure called *tree of connected components*, which is used in [Schulz et al. 2002] to extract a suitable subgraph of an extended multi-level graph. We demonstrate here only how to obtain, for a given query (s, t), a subgraph of a *basic* multi-level graph in which the length of a shortest s-t-path remains unchanged, and refer the reader to [Schulz et al. 2002] for further details (in fact, both variants behave quite similarly). We want to point out that computation of the subgraph and shortest-path search can be performed in one pass, i.e., the subgraph is determined 'on the fly'; a sketch of such a routine is given at the end of this section.

Tree of Connected Components. The subsequent definitions and employed notation are illustrated in Figure 4.

For $1 \leq i \leq l$, consider the subgraph of G induced by $V \setminus S_i$ (we also use the rather informal term of a *decomposition* of G). The set of connected components associated with level i is then denoted by $\mathcal{C}_i$, and for a vertex $v \in V \setminus S_i$ let C_i^v denote the component in $\mathcal{C}_i$ that contains v. For each component in $\mathcal{C}_1 \cup \ldots \cup \mathcal{C}_l$, there is a vertex in the tree; additionally, there is a root C_{l+1} corresponding to G and for every vertex $v \in V$ a leaf C_0^v (our parlance does not distinguish between a connected component and its belonging tree vertex, if context is unambiguous).

The parent of a vertex in the tree is determined as follows. For every component

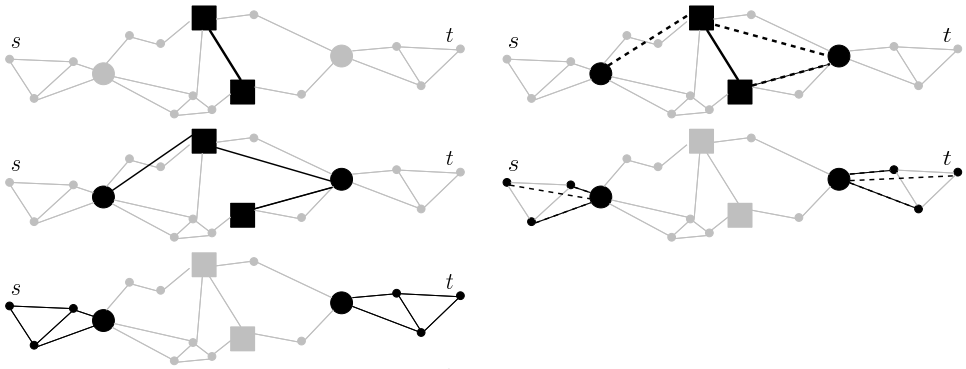

Fig. 5. For given vertices s and t, the subgraphs $\mathcal{M}_{st}$ (left) and $\overline{\mathcal{M}}_{st}$ (right) of the basic and extended multi-level graphs from Figure 3.

$C_i^v \in \mathcal{C}_i$ with $1 \leq i \leq l$ and v being an arbitrary vertex of that component, its parent is C_{i+1}^v (note that component C_{l+1} contains every vertex in V). For a leaf C_0^v, let j be the largest index such that $v \in S_j$, or $j := 0$ if $v \notin S_1$; then the parent of C_0^v is C_{j+1}^v (the smallest level where v is contained in a non-singular connected component is $j+1$).

Definition of the Subgraph. For given vertices s and t, consider the C_0^s-C_0^t-path $(C_0^s, C_k^s, C_{k+1}^s, \ldots, C_{L_{st}}^s = C_{L_{st}}^t, \ldots, C_{k'+1}^t, C_{k'}^t, C_0^t)$ in the component tree, where L_{st} is the smallest index with $C_{L_{st}}^s = C_{L_{st}}^t$ (i.e., this component is the lowest common ancestor of C_0^s and C_0^t) and k and k' are the levels of the parents of C_0^s and C_0^t, respectively. This path induces a subgraph $\mathcal{M}_{st} = (V_{st}, E_{st})$ of the basic multi-level graph $\mathcal{M}(G; S_1, \ldots, S_l)$: E_{st} contains, for each component C_i^x on the path ($x \in \{s,t\}$, $0 < i < L_{st}$), all edges in E_{i-1} incident with a vertex in C_i^x, as well as all level edges $E_{L_{st}-1}$; the vertex set V_{st} is induced by E_{st}. Figure 5 shows $\mathcal{M}_{st}$ for our sample graph, and the subgraph $\overline{\mathcal{M}}_{st}$ of the extended multi-level graph (cf. [Schulz et al. 2002] for the definition) for reference.

Shortest-Path Search. We now describe how to search for a shortest s-t-path in $\mathcal{M}_{st}$ using the multi-level graph $\mathcal{M}$ (as mentioned above, $\mathcal{M}_{st}$ need not be extracted explicitly). We start from s at level $k-1$ of $\mathcal{M}$, using edges in E_{k-1}. When a vertex in S_k is scanned, only outgoing edges in E_k are taken into account, for vertices in S_{k+1}, only edges in E_{k+1} and so on. At level $L_{st}-1$, all edges in $E_{L_{st}-1}$ can be visited. From a vertex in $S_{L_{st}-1}$ incident with component $C_{L_{st}-1}^t$, we 'descend the hierarchy', considering edges in $E_{L_{st}-2}$, and so on until we reach t at level $k'-1$.

By definition of the component tree, any s-t-path must leave component C_k^s. Hence, it suffices to maintain level-$(k-1)$ edges incident with vertices in C_k^s. The remaining part of the shortest path can then be found from level-k vertices using higher-level edges. The same argument applies iteratively for higher levels, and symmetrically for components around t. At level L_{st}, vertices s and t belong to the same component, so all level-L_{st-1} edges are required. Summarizing, we can state (the strict proof is analogous to that in [Schulz et al. 2002]):

LEMMA 3.1. *The lengths of shortest s-t-paths in G and in $\mathcal{M}_{st}$ are equal.*

4. REGULAR MULTI-LEVEL GRAPHS

One basic assumption for multi-level graphs to speed up shortest-path computation is that the multi-level subgraphs are small compared to the original graph. In general, this is not necessarily true; clearly, a bad example would be if a set of vertices did not decompose the input graph at all.

However, for graphs that allow for some 'regular' decomposition we are able to prove, for any (s,t)-query, a bound on the number of edges in the multi-level subgraph, which we will show to depend crucially on the index L_{st} from the previous section. As this number is mainly determined by the number of level-$(L_{st}-1)$ edges and the sets E_i get sparser as i increases, we are also interested in the probability that for a random query at least a given level L_{st} in the component tree is reached.

Note that the subsequent results refer to the *extended* version of multi-level graphs, but can be carried over to the basic variant easily. For the sake of conciseness, we only outline the main results, and refer the reader to [Holzer 2003; Schulz 2005] otherwise: proofs, which contain rather lengthy but straightforward calculations, can be found there.

For experimental purposes, we also wish for graphs that permit a regular decomposition in the specified sense. A class of graphs that meet the theoretical results is therefore described in the second part of this section.

4.1 Theoretical Analysis

After fixing some notation, we formally coin the notion of a regular decomposition, which will serve as an assumption for all of the following considerations: First we bound the number of edges in a regular multi-level graph, i.e., a multi-level graph exhibiting a regular decomposition, then we note the probability that the highest level L_{st} on the path in the component tree is at least some given value, and finally provide an upper bound on the size of a multi-level subgraph.

Notation. By $E_G(S)$ we denote the edge set induced by vertex set S in graph G. Furthermore, given a decomposition of G (with the same notation as in the previous section), the maximal number of selected vertices adjacent to any vertex of any component in $\bigcup_{i=1}^{l} \mathcal{C}_i$ be marked by a.

Definition 4.1. A decomposition of a graph G through vertex sets $S_1, \ldots, S_l$ into connected components $\bigcup_{i=1}^{l} \mathcal{C}_i$ is called *regular* if the number $\sum_{i=1}^{l} |\mathcal{C}_i|$ of all components is at most n and the difference in the sets of edges induced by a consecutive pair of vertex sets is at least halved for two consecutive pairs:

$$|E_G(S_i) \setminus E_G(S_{i+1})| \leq |E_G(S_{i-1}) \setminus E_G(S_i)|/2 \quad (1 \leq i < l).$$

The size of the multi-level graph can be bounded as follows.

Lemma 4.2. *Under the assumption of a regular decomposition, the total number of additional edges in the multi-level graph* $\overline{\mathcal{M}}(G; S_1, \ldots, S_l)$ *is at most*

$$m + (a^2 + a)n.$$

Let $(s,t) \in V \times V$ be a query selected uniformly at random. We want to give the probability that the level L_{st} of the lowest common ancestor of C_0^s and C_0^t in the component tree is at least L $(1 \leq L \leq l+1)$.

LEMMA 4.3. *Under the assumption of a regular decomposition, the probability that for any vertices s and t the index L_{st} is at least some L amounts to*

$$\frac{2|S_{L-1}|n - |S_{L-1}|^2 + (c-1)(n - |S_{L-1}|)^2/c}{n^2},$$

if we suppose that all components in $\mathcal{C}_{L-1}$ have the same size c.

Let us further assume that c and the number $|S_l|$ of vertices in the smallest subset are constant. It follows that the probability with which the C_0^s-C_0^t-path leads via the highest level l converges to $(c-1)/c$ with $n \to \infty$. Loosely speaking, with an apt decomposition we can asymptotically expect to answer almost all queries by taking into account the top level of the multi-level graph.

Finally, we are able to give a bound on the number of edges of the subgraph $\overline{\mathcal{M}}_{st}$.

LEMMA 4.4. *Under the assumption of a regular decomposition, the total number of edges in the subgraph $\overline{\mathcal{M}}_{st}$ is bounded by*

$$2(a + (L_{st} - 2)a^2) + |E_{L_{st}-1}|.$$

It turns out that a and $|E_{L_{st}-1}|$ are the crucial parameters to the size of $\overline{\mathcal{M}}_{st}$. If a can be considered a small constant, together with the above results we get that the search space (number of edges in $\overline{\mathcal{M}}_{st}$) is asymptotically dominated by the number $|E_l|$ of edges at the highest level.

4.2 Component-Induced Random Graphs

Motivated by the above results, we now define a random graph model such that a regular decomposition is possible, *component-induced graph*s, which depend on the following parameters:

—the number l' of construction levels,

—the numbers n' and m' of new vertices and edges, respectively,

—the number c' of new components per level, and

—the number a' of adjacent vertices per component.

Construction of a component-induced graph is roughly done as follows (cf. [Schulz 2005] for further information). At top level, compute a classic Erdös-Réyni random graph with n' vertices and m' edges selected uniformly at random from all possible edges (cf. [Bollobás 1985]); if it is not connected, repeat this step. The remaining $l' - 1$ levels of the hierarchy are constructed in a recursive fashion: For the connected graph/component currently considered, introduce c' new connected components with n' vertices and m' edges each. From each of these components, a' (not necessarily distinct) vertices are picked and an edge from each of these to some randomly selected vertex in the current component is introduced.

A regular decomposition of a component-induced graph can be obtained by including in S_i the vertices generated at levels greater than or equal to $i + 1$ with $1 \leq i < l'$. An example of a component-induced graph with three construction levels is provided in Figure 6.

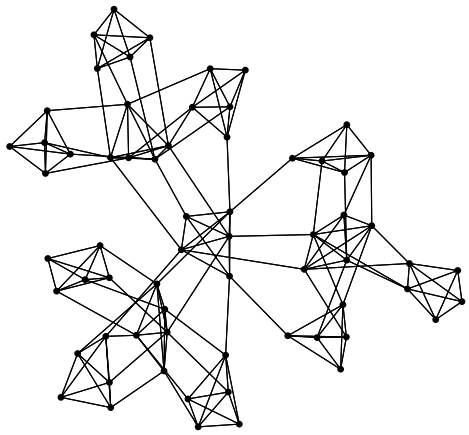

Fig. 6. Sample component-induced graph with parameters $l' = 3$, $n' = 5$, $m' = 10$, $c' = 3$, $a' = 4$.

5. EXPERIMENTAL ANALYSIS

In this section we give several criteria, along with two general strategies, of selecting vertices of a graph to construct a multi-level graph, and introduce four graph classes investigated in the subsequent experimental study. In a preparatory analysis we compare for given graphs the multi-level graphs computed both by the `min-overlay` procedure and according to the definition in [Schulz et al. 2002]. In another prestudy we focus on two of the selection criteria that are special in some sense, *betweenness approximation* and *planar separator*.

The main results show the impact of diverse combinations of graph class and selection criterion and strategy on multi-level graphs as well as their performance when applied for shortest-path search. Further experiments contrast basic and extended multi-level graphs with different numbers of levels.

Our code is written in C++, based on the LEDA library [Näher and Mehlhorn 1999], and compiled with the GNU compiler (version 3.3); as underlying shortest-path routine we use Dijkstra's algorithm. The experiments were carried out on several 64-bit AMD Opteron machines, clocked at roughly 2 GHz, with 4 or 8 GB of main memory.

5.1 Selecting Vertices

In the following, we present a variety of *criteria* of how to determine a subset of a graph's vertex set to construct a multi-level graph. All criteria—except for *planar separator*—can be applied using one of two different selection *strategie*s, *global* or *recursive* (the planar-separator criterion can be applied only in a recursive manner).

5.1.1 *Selection Criteria.* We propose nine criteria: one random (RND) criterion; two criteria related to vertex degree, degree (DEG) and percentage (PCT); one related to graph cores (COR) [Brandes and Erlebach 2005]; four coming from *centrality indexes*, reach (RCH) [Gutman 2004], closeness (CLO), betweenness (BET), and betweenness approximation (BAP) [Brandes and Erlebach 2005]; and one involving a *planar-separator* algorithm (PLS) [Holzer et al. 2005].

Random (RND). Vertices are selected uniformly at random.

Degree (DEG). Vertices with the highest degrees are selected.

Percentage (PCT). We consider for each vertex v its *percentage value*, which is the share of v's adjacent vertices that have smaller degree than v in all adjacent vertices (an isolated vertex is assigned -1). Vertices with the highest percentage values are then selected.

Core (COR). A graph's k-core (for an integer k) is the maximal subgraph such that all vertices in that subgraph have degree at least k. The core number of a vertex is defined to be the maximum k such that this vertex belongs to the k-core. Vertices with the highest core numbers are selected.

Reach (RCH). Reach $r(v, p)$ of a vertex v on a path p is defined to be the minimum of the lengths of p's subpaths with respect to v. Reach of v is then the maximum of all values $r(v, p)$ where p is a shortest path over v, thus denoting the greatest distance of v to the nearer of the end-vertices over all shortest paths containing v. Vertices with the greatest reach values are selected.

Closeness (CLO). Closeness of v is defined as $1 \big/ \sum_{t \in V} d(v, t)$, letting $d(v, t)$ denote the distance from v to t (with $1/0 := 0$). Intuitively speaking, a vertex with great closeness has short distances to most of the other vertices. Vertices with the largest closeness values are selected.

Betweenness (BET). Betweenness of v is defined to be $\sum_{s,t \in V} \sigma(s, t \,|\, v) \big/ \sigma(s, t)$, where $\sigma(s, t)$ stands for the number of shortest paths from s to t and $\sigma(s, t \,|\, v)$ for the number of shortest paths from s to t that contain v as an *internal* vertex (with $0/0 := 0$). Betweenness reflects how important a vertex is to shortest paths. Vertices with the greatest betweenness values are selected.

Betweenness Approximation (BAP). Betweenness can be approximated through random sampling, by not taking into account all pairs (s, t) in $V \times V$ but only in $V' \times V'$, for a subset $V' \subseteq V$ of size $(\log n)/\varepsilon^2$ (with an appropriate choice of ε). The goal is to obtain 'good enough' betweenness values in much shorter time than required for computation of the exact values (cf. Section 5.4.1). The probability of an error larger than $\varepsilon n(n-2)$ is at most $1/n$.

Planar Separator (PLS). This criterion makes use of a planar-separator algorithm, which is—informally speaking—to divide a planar graph into two parts as much balanced as possible by removing a small set of vertices. The vertices returned by this procedure are then taken as selected.

We use the heuristic suggested in [Holzer et al. 2005], which is based on the Planar-Separator Theorem by Lipton and Tarjan. In order to employ this criterion also for non-planar graphs, we first planarize the graph by introducing new vertices at crossings (we presume a fixed embedding). Then the planar-separator algorithm is applied to the planarized—auxiliary—graph. Finally, separator vertices for the original graph are taken over from the auxiliary graph, and conflicts with edges that connect two vertices of different components are resolved by declaring one of the end-vertices a separator vertex (cf. Section 5.4.2).

Figure 7 illustrates the different criteria with a sample graph.

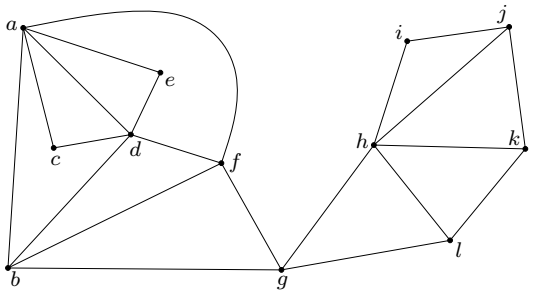

Fig. 7. Highest-priority vertices in a sample graph with unit edge length according to the different selection criteria: DEG: a, d, h; PCT: h; COR: a, b, d, f; RCH: b, f, g; CLO: g; BET: g; PLS: $\{f, g\}$ (for instance).

	n	m
ci	1000	10480
	2000	21460
del	1000	5000
	10000	50000

	n	m
road	995	2470
	9968	25648
	19463	49692
	49625	125018
	99529	252390
	199739	501948
	299790	771418
	399558	1030802
	499604	1283236

	n	m
lrail	999	2534
	1650	4574
	2239	6452
	2348	8458
	4553	15866
	6848	19276
srail	2070	6880
	10795	35996
	12070	39966
	14335	51126

Table I. Sizes of the graphs used in our experiments.

5.1.2 *Selection Strategies.* With each criterion except PLS, we propose two different ways of selecting vertices. The first, called *global strategy*, is to compute, according to the criterion specified, the priority of all vertices in the graph and to pick from amongst them a desired number with the highest priority values.

With the second, referred to by *recursive strategy*, a maximum component size has to be specified. Recursively, for each connected component bigger than that threshold, the vertices are sorted according to the given criterion; one by one, the vertices with the highest priority values are selected until either the component splits or the number of non-selected vertices in this component falls below the threshold. Since with PLS, there is no priority value in the proper sense associated with the vertices—vertices either are or are not contained in the separator set—this criterion can be used in an expedient way only with the recursive strategy, slightly modified in that all separator vertices are selected at once.

5.2 Graph Classes

With our experiments we take into account four types of graphs, two randomly generated and two taken from real world. All graphs are connected and bidirected, i.e., as the case may be, each edge has been replaced with two directed edges, one in either direction. For each of the subsequent graph classes we provide a short key, which can be further specified by the number of vertices to denote a concrete instance. Table I provides a synopsis of the graph sizes.

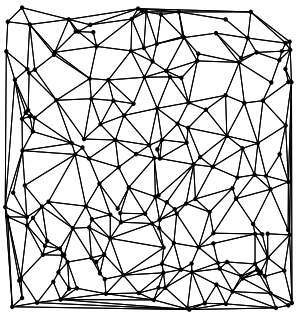

Fig. 8. Sample planar Delaunay graph.

Component-Induced Graphs (`ci`*).* Due to construction (cf. Section 4.2), these random graphs exhibit some regular hierarchical structure and are rather dense compared to the other classes. Edge lengths are chosen at random.

Planar Delaunay Graphs (`del`*).* Planar Delaunay graphs are graphs with vertices randomly spread over a unit square for which the Delaunay triangulation is computed. Then edges are deleted at random until a given number is reached. We chose the number of edges such that density ranges somewhere between the values for the `ci` and real-world graphs. Edge lengths correspond to Euclidian distances.

Road Graphs (`road`*).* By road graphs we denote subgraphs of the German road network.[2] These graphs are comparatively sparse. The length of an edge is the length of the corresponding road section—not the straight-line distance—with a granularity of 10 meters. For our experimental study, we use road graphs with up to roughly 500 000 vertices.

Railway Graphs (`rail`*).* Railway[3] graphs are condensed networks reflecting train connections[2] (cf. [Schulz et al. 2000]): vertices stand for railway stations, and there exists an edge between two vertices if there is a non-stop connection between the respective stations. As opposed to `road` graphs, which are almost planar, specimens of this class can have quite some edges spanning a major distance. Graphs representing long-distance traffic within some European countries (`lrail`) or local/short-distance transportation of several German regions (`srail`) are provided. The length of an edge is assigned the average travel time of all trains that contribute to this edge. The `lrail` graphs contain up to almost 7000 vertices, while there are approximately twice as many in the largest `srail` graph.

Sample instances of these graph classes can be found in Figures 6, 8, and 9, respectively. The generator for the `ci` and `del` graphs is available on-line [Borgi et al. 2005].

[2]We are grateful to the companies PTV AG, Karlsruhe, and HaCon, Hannover, for providing us with road and railway data, respectively, for scientific purposes.

[3]Note that terms like *railway*, *train*, etc. here comprise also other means of public transportation, such as trams, local buses and so on.

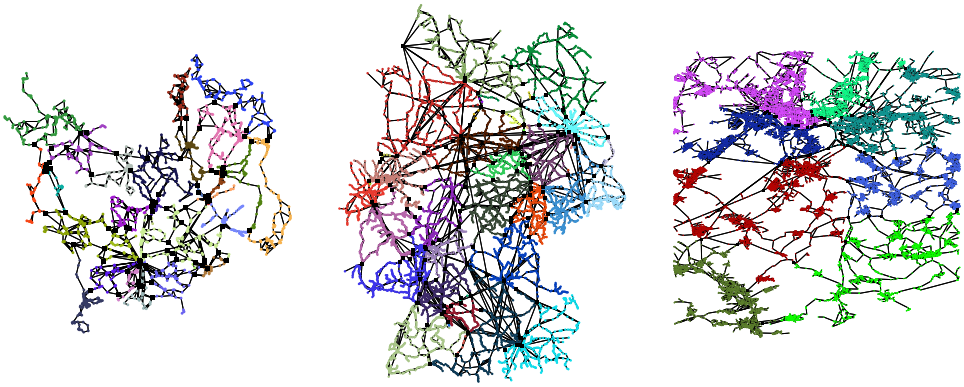

Fig. 9. Recursive decompositions of `srail2070` (local bus service network in central Germany), `lrail6848` (railway network of Germany), and `road19463` (road network of Karlsruhe and surrounding area) by PLS. The black squares mark the separator vertices, different components are indicated by colors.

	unit lengths		genuine lengths	
graph	opt	blowup	opt	blowup
`ci2000`	7.0	1.21	6.9	1.00
`del10000`	31.1	2.95	35.4	1.00
`road19463`	12.6	1.26	15.8	1.01
`rail6848`	5.9	1.37	8.8	1.00

Table II. Comparison of multi-level graphs computed through `min-overlay` and according to [Schulz et al. 2002]. *opt* denotes the number of edges divided by the number of vertices obtained with `min-overlay`, while *blowup* indicates the multiplicative factor indicating how many edges in relation are constructed using the non-minimal method. We distinguish the case of unit (left) and genuine (right) edge lengths.

5.3 Shortest-Path Overlay Graphs

In Theorem 2.2 we proved that the procedure `min-overlay` yields shortest-path overlay graphs with a minimal number of edges. To get an idea of the amount of edges that can be saved when switching from the definition in [Schulz et al. 2002][4] to the minimal variant given in this work, we compare for one graph of each class the sizes of (extended) multi-level graphs obtained by either procedure (induced by one subset of vertices each, determined with PLS). Moreover, we provide two different kinds of edge lengths: genuine ones, as described above, and unit lengths.

The outcome is depicted in Table II: *opt* denotes the number of edges divided by the number of vertices in the minimal overlay graph and *blowup* the quotient of the numbers of edges obtained with each procedure. Under the use of unit lengths, there exist many paths of equal length, resulting in a comparatively big blowup (almost 3 for the `del` graph) while with genuine lengths, there is practically none.

[4]The procedure in [Schulz et al. 2002] differs from `min-overlay` in one fundamental respect: When two shortest paths of equal length are encountered, one of them is picked arbitrarily to be included in the multi-level graph. This may result in different potential multi-level graphs so for our comparison, we consider one with a maximal number of edges.

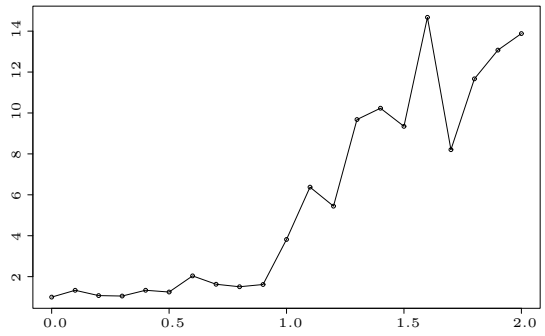

Fig. 10. Quotient of the average numbers of edges visited during shortest-path computation with multi-level graphs based on BAP and BET. The abscissa denotes the parameter ε, governing the random sampling. As input graph, `road49463` is used.

The latter observation is owed to the fact that edge lengths are Euclidean lengths or mean travel times, represented by double-values, so it is rather unlikely that different paths between two vertices have equal length. If actual, integer-valued travel times were used for the `rail` graph instead, we would expect a blowup factor lying between the two given in the table.

5.4 Special Selection Criteria

Most of the criteria described in Section 5.1.1 are uniquely determined. However, BAP involves parameter-dependent random sampling, where an appropriate choice of the parameter for our purposes cannot be given offhand. Moreover, PLS applied to non-planar graphs requires planarization and retranslation steps, where effects on the size of the selection set remain quite unclear. These issues are highlighted in the following prestudies.

5.4.1 *Betweenness Approximation.* To assess the quality of betweenness approximation and to earmark a parameter setting for practical application, we determine for different choices of ε vertex selection sets and investigate the average search space size of belonging multi-level graphs: For a fixed maximum component size, we construct extended multi-level graphs of `road49463`, induced by both exact and approximated betweenness, and with each of these graphs, answer a series of shortest-path queries. We then evaluate the number of visited edges, i.e., scanned by the shortest-path algorithm.

Figure 10 shows the ratio of the numbers of visited edges with approximated and exact betweenness values, for choices of ε of up to 2: the larger this ratio, the smaller is the speed-up achieved with BAP. We observe that with increasing ε, performance of the multi-level approach at first worsens only very slowly but for $\varepsilon \geq 1$, slumps dramatically.

For subsequent experiments with BAP, we want to play safe by setting ε to a value of only 0.2. Nevertheless, this leads to a drastically reduced preprocessing time of about 0.5 % of that needed to compute exact betweenness. Further tests with other `road` graphs confirmed this choice of ε as appropriate.

graph	n^*	$\lvert S^*\rvert$	$\lvert S\rvert$	$\lvert S_{\text{opt}}\rvert$
`road19463`	479	165	196	177
`road49625`	1060	336	410	382
`road99529`	2591	773	941	855
`road199739`	3754	2176	2636	2315
`road299790`	8025	3399	4104	3628
`lrail1650`	645	22	38	16
`lrail2239`	1830	68	107	55
`lrail2348`	3458	154	132	58
`lrail4553`	11447	605	412	164
`lrail6848`	3169	183	399	164

Table III. Application of the planar-separator algorithm to non-planar graphs (`road` and `lrail`). The following measurements (average values with different maximum component sizes) are reflected: number n^* of crossings in the input graph/planarization vertices, size $|S^*|$ of the separator for the planarized graph, size $|S|$ of the retranslated separator, and size $|S_{\text{opt}}|$ of the separator for the input graph after removal of redundant vertices.

5.4.2 *Planar Separator.* As mentioned in Section 5.1.1, our planar-separator algorithm can be applied also to non-planar graphs: Planarize the input graph by introducing a vertex for each crossing, run the separation algorithm, and retranslate the separator found to the original graph by possibly including further vertices in the separator set. Afterwards, a simple procedure can be used to optimize the separator set by sorting out 'redundant' vertices. The main issues that we want to cover in this prestudy are: the number of crossings in the input graph (and thus the number n^* of planarization vertices) and the size $|S_{\text{opt}}|$ of the separator set induced for the original graph. We respect `road` and `lrail` graphs, with maximum component sizes of 500 and 1000.

The results are depicted in Table III, showing for each graph the number of crossings as well as the separator sizes (for the planarized graph, the original graph, and after optimization). As alluded above, the `road` graphs are already almost planar, which is not true for the `rail` graphs. This is underpinned by the values for n^*, ranging around 2 percent of the number of original vertices for `road` graphs, but between 39 and an enormous 251 percent for `lrail`. The optimized separator sets S_{opt} are quite small for both classes, however, consisting of roughly 1 percent and up to 3.6 percent of vertices in the input graph, respectively. This outcome suggests for further experiments that decomposition by PLS at least of our real-world graphs through a 'reasonable' number of vertices is possible.

One alternative to the planar-separator algorithm is the graph-partitioning tool METIS [Karypis 2005], which computes balanced edge partitions rather than vertex separations (from an edge partition, a vertex separator can be derived by a simple greedy heuristic). In [Holzer et al. 2005], it is shown that separators obtained through METIS are of almost the same quality (with respect to both separator size and component balance) as those received by our planar-separator algorithm. Preliminary experiments corroborate that this observation carries over to the multi-level approach in that performance with selected vertices determined via METIS is slightly worse than with PLS-computed selections.

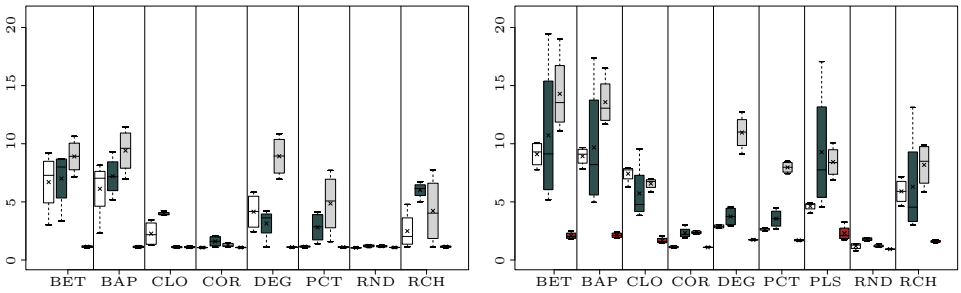

Fig. 11. Speed-up in terms of visited edges with global (left) and recursive (right) decomposition. The x-axis denotes the selection criterion; graphs considered (from left to right within each criterion): `ci1000` (white), `road995` (dark-gray), `lrail999` (light-gray), and `del1000` (brown). The boxplots span the average speed-up values with different choices for the number of selected vertices (3, 5, 8, and 10 percent) and the maximum component size (3, 5, 10, and 20 percent), respectively. The horizontal line within a box denotes the median, the cross marks the mean value.

5.5 Multi-Level Approach

This section contains a computational study in which we investigate the performance of the multi-level approach with different parameters: We consider diverse combinations of graph class, selection criterion and strategy, and number of levels, as well as contrast the basic and extended versions. Our experiments are divided into three sections: first, we focus on extended multi-level graphs and explore different combinations of settings, but restrict ourselves to only one additional level; then we compare basic multi-level graphs to the extended variant; finally, we factor multiple levels into our experiments.

As most important measure for *speed-up* we use the quotient of the number of edges visited by Dijkstra's algorithm over that number visited by the search routine of the multi-level approach. This parameter is implementation- and machine-independent, and turned out to be closely related to CPU time.

5.5.1 *Selection Criteria.* We explore all combinations of graph type, selection criterion, and number of selected vertices or maximum component size, respectively (due to the large number of combinations, we have to settle for rather small graphs). According to these results, we pick in a second pass the most promising parameter settings to run them with a series of larger graphs, where we take a closer look at the influence of the maximum component size.

Small Graphs. We take into account one graph of each type with about 1000 vertices: `ci1000`, `del1000`, `road995`, and `lrail999`. With the global strategy, we choose for the number of selected vertices 3, 5, 8, and 10 percent of the number of vertices in the graph and with recursive decomposition, the maximum component size is set to 3, 5, 10, and 20 percent, which in some preliminary runs turned out to be representative values. For each of these combinations, we run 1000 queries selected at random and compute the averages of the resulting speed-up values.

Figure 11 presents average speed-up in the form of standard boxplots: Each box spans, for one criterion and one graph, the range obtained with the four choices for the number of selected vertices and the maximum component size, respectively.

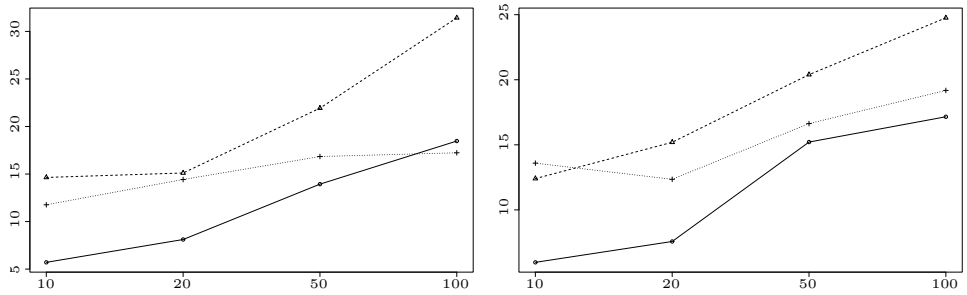

Fig. 12. Average speed-up with PLS (left) and recursive BAP (right) for larger `road` graphs. The x-axis denotes the number of vertices in the input graph (in thousands). Maximum component sizes: 1 (solid), 10 (dashed), and 20 (dotted) percent.

Overall, the highest speed-up values are obtained with recursive BET/BAP (there is hardly any difference between these two criteria; cf. Section 5.4.1), which work very well for all graphs but `del1000`; a factor of almost 20 can be achieved with `road995`. The second-best criterion, of similar quality, turns out to be PLS, followed by recursive RCH. The DEG criterion suitably decomposes `lrail999`, which can be explained by the large range of vertex degrees compared to other graphs. Some of the other criteria work only slightly better than selection by RND. In general, with recursive decomposition higher speed-ups are attainable.

As for graph classes, `del` constitutes a hard instance, whereas for the real-world and `ci` graphs the approach is well-suited. For `ci1000`, recursive BET/BAP yields a maximal speed-up of around 10, which quite corresponds to the value obtained with a multi-level graph induced by the vertices selected during construction of the input graph. Analyzing for global and recursive decomposition the variability of the component sizes and the number of selected vertices in the multi-level graphs, respectively, a strong correlation to the speed-up values becomes evident: the smaller component size variance and selection set, the better the speed-up. Note that these results parallel our deliberations on regular decomposition in Section 4.

Medium-Size Graphs. With a series of somewhat larger real-world graphs and the most promising criteria identified in the previous paragraph, we again run random queries with different maximum component sizes. We use recursive BAP and PLS for the `road` as well as recursive DEG and PLS for the `rail` graphs.

Figure 12 shows the speed-up for `road` graphs with up to 100 000 vertices. For the largest graph, a speed-up factor of 31 is reachable. Here, the PLS criterion is clearly superior to BAP, with which the maximal speed-up is 22. For the maximum component size, 10 percent turns out to be the best choice. Concerning preprocessing times, decomposition of `road99529` takes several minutes with PLS, but around two hours with BAP. Times for the construction of the multi-level graph, of well over half an hour, are similar for both criteria.

According to these findings, we settle for a maximum component size of 10 percent for `rail` graphs, but distinguish between long- and short-distance networks (cf. Figure 13, left). Interestingly, for `lrail` graphs, DEG works better than PLS while for `srail`, speed-up with DEG is not very pronounced, whereas PLS yields much higher factors.

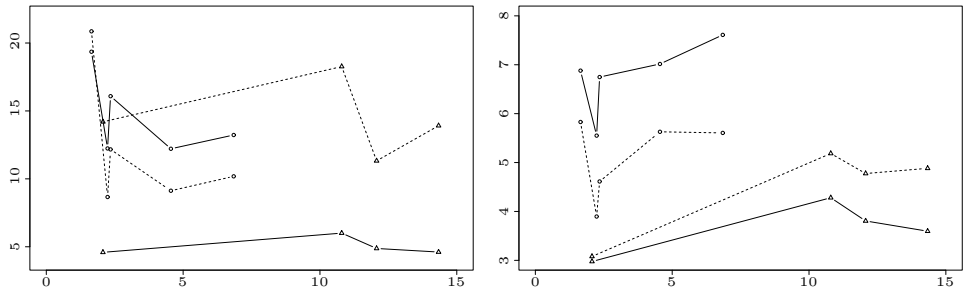

Fig. 13. Average speed-up with extended (left) and basic (right) multi-level graphs of `rail` graphs. The x-axis denotes the number of vertices in the input graph (in thousands). Maximum component sizes: 10 percent for the extended and 1 percent for the basic multi-level graphs; selection criteria: recursive DEG (solid) and PLS (dashed); graph classes: `lrail` (circle) and `srail` (triangle).

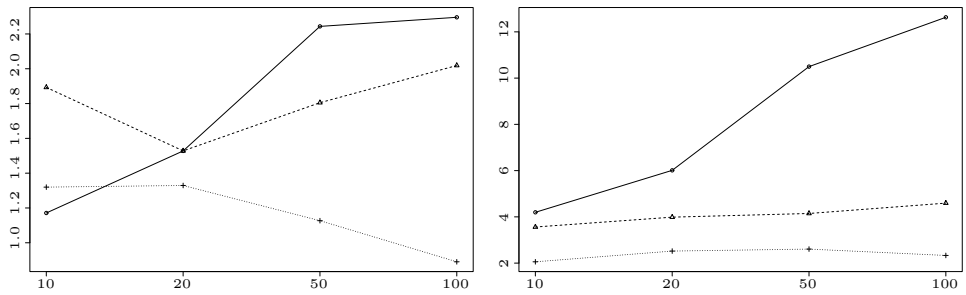

Fig. 14. Relative average speed-up with extended (left) and basic (right) multi-level graphs of `road` graphs. The x-axis denotes the number of vertices in the input graph (in thousands). Maximum component sizes: 1 (solid), 10 (dashed), and 20 (dotted) percent; selection criterion is PLS.

5.5.2 *Basic Multi-Level Graphs.* The main feature that distinguishes basic from extended multi-level graphs is that no upward and downward edges are maintained (cf. Section 2.2). Hence, one may expect more edges that have to be visited during a search and thus less speed-up; at the same time, the overhead of storing additional edges is smaller. This notion is captured by *relative speed-up*, which is defined as speed-up divided by graph expansion, where *graph expansion* is the quotient of the numbers of edges in the multi-level and the original graph. We run experiments with the same medium-sized `road` and `rail` graphs as above.

Figure 14 depicts relative speed-up for the `road` graphs with both extended and basic multi-level graphs and the PLS criterion. The best value observed with the extended version is about 2.3, but well over 12 with the basic. Graph expansion of basic multi-level graphs is only slightly greater than 1, so the right-hand diagram shows at the same time virtually pure speed-up and can thus be perfectly compared to Figure 12: best speed-up is obtained with a maximum component size of 1 percent, in contrast to 10 percent for the extended version. This suggests that the maximum component size should be chosen smaller for basic multi-level graphs (with respect to relative speed-up, however, 1 percent seems to be advantageous for both variants). Finally, the right-hand diagram in Figure 13 reflects speed-up with basic multi-level graphs of `rail` graphs.

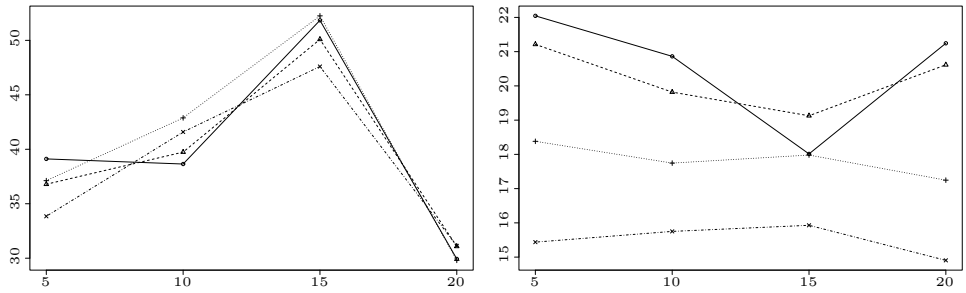

Fig. 15. Average speed-up with extended (left) and basic (right) multi-level graphs of `road99529` with two additional levels. The x-axis denotes the maximum component size (in thousands of vertices) for level 1; maximum component sizes for level 2: 300 (solid), 500 (dashed), 1000 (dotted), and 1500 (dotted-dashed) vertices; separation criterion is PLS.

5.5.3 *Multiple Levels.* The last series of experiments is devoted to the question of how much speed-up can be gained by introducing more than one level, where we use medium-sized and large `road` (cf. Table I) as well as larger `ci` graphs.

Road Graphs. We consider three-level extended and basic multi-level graphs of `road99529`, based on different combinations of vertex selections through PLS. Figure 15 shows that speed-up can be increased from 31 with one level (cf. Figure 12) to well over 50 (with maximum component sizes of 15 and 3 to 5 percent) and from about 13 (cf. Figure 14) to 22 (with 5 and .3 percent) using extended and basic multi-level graphs, respectively. A similar behavior could be observed with even larger `road` graphs: not only could speed-up be improved by introducing another level; also, with extended multi-level graphs, graph expansion could be reduced drastically while for basic multi-level graphs, it exhibited as still negligible.

Component-Induced Graphs. Last, we want to refer to [Holzer 2003] for an experimental study investigating component-induced graphs with up to 100 000 vertices and belonging extended multi-level graphs with up to five additional levels, where vertices used during construction of the input graphs are selected. The main outcome is that with increasing graph size, speed-up scales to a factor of approximately 1000. These results further confirm the intuition and theoretical results from Section 4 and demonstrate the importance of a fairly regular decomposition.

5.5.4 *Summary.* We want to conclude our empiric study by extracting from the above experiments some principal insights, which can be seen as a guideline to choosing good parameter settings for multi-level shortest-path computation. Concerning selection criterion, we would recommend PLS or recursive BAP[5], or recursive DEG for real-world graphs with a great variance of vertex degrees. Storage capacity permitting, the extended variant should be favored over the basic (where the latter allows for unbeatable relative speed-up), with a maximum component size of around 10 percent of the vertices in the input graph. For graphs with more than 1000 vertices, employing two or even more levels should be considered.

[5]In our experiments, BAP may have incurred greater preprocessing times, but this defect could be overcome by a more courageous choice of ε, with little loss in speed-up.

6. CONCLUSION

In this survey, we reviewed the multi-level technique for shortest-path computation, improved the definition of shortest-path overlay/multi-level graphs and introduced a new—the basic—variant, as well as provided several criteria along with two general strategies to select vertices in a graph for constructing multi-level graphs. The theory part provides some common considerations regarding the speed-up that can be achieved given a regular decomposition of the input graph. In an extensive experimental study, both variations of multi-level graphs, induced by the different selection criteria and strategies, were tested with various random and real-world graphs with respect to speed-up when used for answering shortest-path queries.

As for the results, the recursive strategy performed better than the global one, and the planar-separator and betweenness criteria clearly outdid the others; further, betweenness was shown to be approximated efficiently. Also, the degree criterion sped up query times with long-distance rail graphs. Comparing the two variants in terms of mere speed-up, extended multi-level graphs were superior to basic ones; however, the latter turned out to be more efficient when the sizes of the multi-level graphs were taken into account.

ACKNOWLEDGMENTS

The authors would like to thank Imen Borgi, Sebastian Knopp, and Andrea Schumm for their support in parts of the implementation work. Special thanks go also to the two anonymous referees for their thorough corrections and numerous suggestions, which helped greatly to enhance the quality of this work.

REFERENCES

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall.

Bast, H., Funke, S., Matijevic, D., Sanders, P., and Schultes, D. 2007. Transit to constant shortest-path queries in road networks. In *Proc. 9th Workshop on Algorithm Engineering and Experiments*. SIAM, 46–59.

Bauer, R. 2006. Dynamic speed-up techniques for Dijkstra's algorithm. M.S. thesis, Universität Karlsruhe (TH), Fakultät für Informatik.
`http://i11www.ira.uka.de/teaching/theses/files/da-rbauer-06.pdf`.

Bollobás, B. 1985. *Random Graphs*. London Academic Press.

Borgi, I., Graf, J., Holzer, M., Schulz, F., and Willhalm, T. 2005. A graph generator.
`http://i11www.ira.uka.de/resources/graphgenerator.php`.

Brandes, U. and Erlebach, T., Eds. 2005. *Network Analysis*. LNCS, vol. 3418. Springer.

Delling, D., Holzer, M., Müller, K., Schulz, F., and Wagner, D. 2007. High-performance multi-level graphs. In *Proc. Workshop on DIMACS Shortest-Path Challenge*. To appear.
`http://i11www.ira.uka.de/members/mholzer/publications/pdf/dhmsw-hpmlg-06.pdf`.

Delling, D., Sanders, P., Schultes, D., and Wagner, D. 2007. Highway hierarchies star. In *Proc. Workshop on DIMACS Shortest-Path Challenge*. To appear.
`http://i11www.ira.uka.de/members/delling/files/dssw-hhs-06.pdf`.

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

Goldberg, A. and Harrelson, C. 2005. Computing the shortest path: A* search meets graph theory. In *Proc. 16th Symposium on Discrete Algorithms*. SIAM, 156–165.

Goldberg, A., Kaplan, H., and Werneck, R. 2006. Reach for A*: Efficient point-to-point

shortest path algorithms. In *Proc. 8th Workshop on Algorithm Engineering and Experiments*. SIAM, 129–143.

Gutman, R. 2004. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *Proc. 6th Workshop on Algorithm Engineering and Experiments*. SIAM, 100–111.

Holzer, M. 2003. Hierarchical speed-up techniques for shortest-path algorithms. M.S. thesis, Universität Konstanz, Fachbereich Informatik und Informationswissenschaft. `http://www.ub.uni-konstanz.de/kops/volltexte/2003/1038/`.

Holzer, M., Prasinos, G., Schulz, F., Wagner, D., and Zaroliagis, C. 2005. Engineering planar separator algorithms. In *Proc. 13th European Symposium on Algorithms*. LNCS, vol. 3669. Springer, 628–639.

Holzer, M., Schulz, F., and Willhalm, T. 2004. Combining speed-up techniques for shortest-path computations. In *Proc. 3rd Workshop on Experimental and Efficient Algorithms*. LNCS, vol. 3059. Springer, 269–284.

Jing, N., Huang, Y.-W., and Rundensteiner, E. A. 1998. Hierarchical encoded path views for path query processing: An optimal model and its performance evaluation. *IEEE Trans. Knowledge and Data Engineering 10,* 3.

Jung, S. and Pramanik, S. 2002. An efficient path computation model for hierarchically structured topographical road maps. *IEEE Trans. Knowledge and Data Engineering 14,* 5.

Karypis, G. 2005. METIS. `http://www-users.cs.umn.edu/~karypis/metis`.

Lauther, U. 2004. An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. In *Geoinformation und Mobilität – von der Forschung zur praktischen Anwendung*. Vol. 22. IfGI prints, Institut für Geoinformatik, Münster, 219–230.

Möhring, R. H., Schilling, H., Schütz, B., Wagner, D., and Willhalm, T. 2005. Partitioning graphs to speed up Dijkstra's algorithm. In *Proc. 4th Workshop on Experimental and Efficient Algorithms*. 189–202.

Näher, S. and Mehlhorn, K. 1999. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press. `http://www.algorithmic-solutions.com`.

Sanders, P. and Schultes, D. 2005. Highway hierarchies hasten exact shortest path queries. In *Proc. 17th European Symposium on Algorithms*.

Sanders, P. and Schultes, D. 2006. Engineering highway hierarchies. In *Proc. 14th European Symposium on Algorithms*. LNCS, vol. 4168. Springer, 804–816.

Schultes, D. and Sanders, P. 2007. Dynamic highway-node routing. In *Proc. 6th Workshop on Experimental and Efficient Algorithms*. LNCS. Springer, 66–79.

Schulz, F. 2005. Timetable information and shortest paths. Ph.D. thesis, Universität Karlsruhe (TH), Fakultät für Informatik.

Schulz, F., Wagner, D., and Weihe, K. 2000. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *J. Experimental Algorithmics 5,* 12.

Schulz, F., Wagner, D., and Zaroliagis, C. 2002. Using multi-level graphs for timetable information in railway systems. In *Proc. 4th Workshop on Algorithm Engineering and Experiments*. LNCS, vol. 2409. Springer, 43–59.

Wagner, D. and Willhalm, T. 2003. Geometric speed-up techniques for finding shortest paths in large sparse graphs. In *Proc. 11th European Symposium on Algorithms*. LNCS, vol. 2832. Springer, 776–787.

Willhalm, T. and Wagner, D. 2007. Shortest path speedup techniques. In *Algorithmic Methods for Railway Optimization*. LNCS, vol. 4359. Springer.