A New Perspective on Efficient and Dependable Vehicle Routing

Dieter Pfoser*
Institute for the Management of Information Systems
G. Mpakou 17
11524 Athens, Greece
pfoser@imis.athena-innovation.gr

Alexandros Efentakis
RA Computer Technology Institute
Davaki 10
11526 Athens, Greece
efedakis@cti.gr

Agnès Voisard
Fraunhofer ISST and FU Berlin
Steinplatz 2
10623 Berlin, Germany
Agnes.Voisard@isst.fraunhofer.de

Carola Wenk
Dept. of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249-1644, USA
carola@cs.utsa.edu

ABSTRACT

The essential elements of any navigation system are a shortest-path algorithm and a map dataset. When seen in the light of the basic requirement to such a system, to provide high quality navigation solutions fast, algorithms have to be efficient and road networks have to be up-to-date. The contribution of this work is two-fold. First, the HBA* algorithm, an efficient shortest-path algorithm is presented that mimics human driving behavior by exploiting road network hierarchies. HBA* is a fast algorithm that produces high quality routes. Second, in a thorough performance study dynamic travel times are introduced to replace the unreliable static speed types currently used in connection with road network datasets. Dynamic travel times are derived from large quantities of historic vehicle tracking data. The integrated result, fast algorithms using accurate data, is empirically evaluated using actual road network datasets and related dynamic travel time data.

Keywords

shortest-path computation, hierarchical road network, dynamic weights, speed profiles

1. INTRODUCTION

The essential components of a navigation system are (i) a shortest-path algorithm and (ii) a map dataset. The delivered solution can be the shortest in terms of either distance or time. We concentrate on the latter. In order to deliver high quality routing solutions to users fast, algorithms need to be efficient and road networks up-to-date. We introduce the HBA* algorithm, a *bidirectional* version of the A* algorithm that *utilizes road network hierarchies* to achieve faster computation times. The HBA* uses *hierarchical jumping*, a technique that favors the use of the higher category roads to reduce the overall search space and to significantly improve the running time of shortest-path computation. We show that provided the road network fulfills certain properties (connectedness), the HBA* is guaranteed to find a solution. The question that needs to be answered is how good the result will be when compared to an optimal shortest-path solution.

In relation to shortest-path computation, we can cite the following literature. Bidirectional heuristic search algorithms were introduced early to speed up computation in the general field of artificial intelligence [19, 22]. Many approaches have been proposed since then, among them a recent one based on the avoidance of repetitive searching usually induced by a bidirectional A* algorithm [27]. A comprehensive survey on the use of heuristics in the domain of transportation applications can be found in [11]. Introducing hierarchies in data structures as a divide and conquer strategy has been used for decades in data-intensive applications to speed up computing, especially in geospatial applications (map storage, map visualization, etc.). In the case of road datasets, it was introduced rather late (see e.g., [24, 5]). Virtual shortest-path-view was introduced in [21]. Hierarchical search has been also applied to the gaming context [1], by using hierarchical clusters of the game world and pre-computed paths between them to speed up routing. Closer related to our work, [15] uses knowledge on the road types to subdivide the road network and optimize shortest-path search.

*Work done when at RA Computer Technology Institute, Greece.

2. HIERARCHICAL ROUTING

In the following, we outline the basic principles behind shortest-path algorithms before discussing the properties of hierarchical road networks and how they can be exploited to improve algorithmic performance. The outcome of this discussion will be the HBA* shortest-path algorithm.

2.1 Algorithmic Solutions

A road network, made of links, is modeled as a directed graph $G = (V, E)$, whose vertices V represent intersections, and edges E represent links between intersections. Additionally, a real-valued weight function $w : E \rightarrow \mathbf{R}$ is given, mapping edges to weights. In the routing context, such weights typically correspond to speed types derived from road categories or based on average speed measurements. However, what is important is that such weights are static, i.e., once defined they are rarely changed.

Given a path $p = \langle v_0, v_1, \ldots, v_k \rangle$ in G, the weight of the path is the sum of the weights of its constituent edges $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$. The weight $\delta(u, v)$ of a shortest path between vertices u and v is defined as:

$$\delta(u, v) = \left\{ \begin{array}{l} min\{w(p) : \ p \text{ is a path from } u \text{ to } v\} \\ \infty \end{array} \right\} \qquad (1)$$

A shortest path from u to v is any simple path p with $w(p) = \delta(u, v)$ [7].

Assume a directed graph with non-negative edge weights $w(u, v) \geq 0$ is given. The *single source shortest path problem* of finding a shortest path between a source vertex s and a target vertex t can always be solved by applying a uniform-cost all-pairs shortest paths algorithm (also known as greedy or Dijkstra's algorithm [10]). This incremental algorithm maintains a set S of vertices whose final shortest path weights from source vertex s have already been determined. It also maintains a priority queue of all vertices $v \in V - S$, ordered by the shortest path estimate $d(v)$. Repeatedly, the vertex $u \in V - S$ with the *minimum shortest-path estimate* is extracted from the priority queue, added to S, and all edges (u, v) are relaxed. This means that it is tested whether the shortest path estimate for v can be improved by considering a path through u. If that is the case, the shortest-path estimate $d(v)$ is updated (which also affects the ordering in the priority queue). The algorithm terminates when the set $V - S$ is empty, i.e., when all edges have been relaxed, or when a designated target vertex has been extracted.

Although Dijkstra's algorithm is guaranteed to find the shortest path from s to any vertex u, it is an uninformed search algorithm that usually explores too many vertices that have no influence on shortest paths from s to t. This behavior can be improved by exploiting knowledge about the structure of shortest paths from s to t. In that context, informed search algorithms ("best-first search") have been proposed, among them pure heuristic-based algorithms and the A* algorithm [12], which combines Dijkstra's algorithm with a heuristic-based algorithm. The A* algorithm differs from Dijkstra's algorithm in that for the selection of $u \in V - S$, it uses the cost $d(u)$ of a shortest path from s to u, in combination with the *estimated cost to the goal*, $h(u, t)$. More precisely, the order of the priority queue storing $V - S$ (the "open list") is based on

$$f(u) = d(u) + h(u, t) \qquad (2)$$

The value of the *heuristic function* $h(u, t)$ has to be heuristically determined. h is called *admissible* if $h(u, t) \leq h^*(u, t)$ for all $u \in V$, where $h^*(u, t)$ is the cost of a shortest path from u to t. In other words, h is admissible if it never overestimates the real cost of reaching the target. It has been shown [16, 9] that for admissible h, the A* algorithm correctly computes the shortest paths. That means, when the algorithm terminates, $d(t)$ is the correct weight of a shortest path from s to t. For shortest path problems in the Euclidean plane, the straight-line (Euclidean) distance is admissible and easy to compute, and is therefore often used as the heuristic function.

Although the A* algorithm is optimal (cf. Section 2.1) in that, if it uses an admissible heuristic function it finds the shortest path, a point of criticism is still its efficiency. We introduce in the following the hierarchical, bidirectional A* (HBA*) algorithm to efficiently compute short path solutions. The HBA* is a *bidirectional algorithm exploiting road category metadata* to effectively reduce the size of the road network during the search.

In the following, we first discuss hierarchical road networks, then introduce the actual HBA* algorithm, and finally show some of its important properties.

2.2 Hierarchical Road Networks

Roadmap data available from vendors usually provides road category information for each road segment/link. Road categories include "Freeway", "Major Road", and "Local Road of Minor Importance". In this categorization, low numbers are assigned to higher road categories, i.e., the highest road category is "0: Freeway", and the lowest "8: Other Road". Also shown are the link-based speed types (static weights), i.e., the average speed that can be achieved in a road category. This table also gives the size of the respective road networks in number of links per road category.

This road category information gives rise to interpret the network as a *hierarchical road network*: Level L_i of the road network consists of all road segments of road categories $j \leq i$, including all nodes incident to those segments. Let $G = (V, E)$ be the whole road network with vertex/node set V and road segment/link/edge set E, and let $L_i = (V_i, E_i)$. Then $V_i \subseteq V_{i+1}$ and $E_i \subseteq E_{i+1}$ for all i, and $V = \cup_i V_i$ and $E = \cup_i E_i$.

Connectivity assumption: It is commonly assumed that each level $L_i = (V_i, E_i)$ in a hierarchical road network is *strongly connected*. This means that for every $u, v \in V_i$ there exists a path in L_i from u to v as well as a path from v to u (the two paths do not have to be the same). We call a hierarchical road network that fulfills this condition *strongly connected*. We will use this assumption in Section 2.4 to prove important properties about the proposed HBA* algorithm. To check whether the road networks used in our experiments (cf. Section 4) fulfill the above assumption, we use the JGraphT library [14] implementation of an algorithm presented in [7]. The algorithm is a special application of DFS with a running time of $O(V + E)$.

To best understand the significance of hierarchical road networks, consider the typical example of how roads of varying importance would typically be used in a *routing task by a human*. Figure 1 gives an example of driving from one neighborhood in Athens (Kallithea) to another (Moschato). The route length is 3.45km including varying road categories. The route starts at level 6 (gray), and continues on levels 5 (red), 3 (blue), 4 (green), 5 (red) and arrives at level 6 (gray). Such a route represents a typical behavior of a driver searching for a route between two locations: First, she searches for a major road connecting the two areas of interest, and then she finds access roads to those major roads [2].

The basic question that needs to be addressed is *how we can mimic such route finding behavior in shortest-path search algorithms*.

2.3 Hierarchical Networks and Shortest-Path Algorithms

Exploiting road network hierarchies in a shortest-path algorithm requires some re-thinking of the general algorithmic strategy. The A* algorithm computes a shortest path from a source to a target by expanding nodes and recording the respective path cost. Considering the example shown in Figure 1, ideally, shortest-path search should disregard lower category links during the middle portion of the route search, since they are unlikely to contribute to a better route solution. Note that in the example of Figure 1, the *spatiotem-*

Figure 1: A typical route for driving from one neighborhood to another

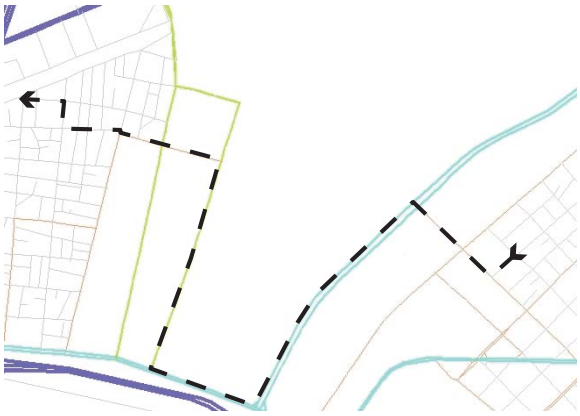


Figure 2: Road network hierarchy sequence

poral category sequence of road categories of the links along the path from the source to the target is $6-5-3-4-5-6$. The numbers in this category sequence are first monotonically decreasing, and then monotonically increasing. In other words, this sequence is *decreasing-increasing bitonic*, or short *bitonic*.

2.3.1 Hierarchical Jumping

In order to utilize the hierarchical road network, our HBA* algorithm will only consider paths that have a bitonic category sequence. The HBA* algorithm will consist of two separate A* algorithms that each compute shortest paths with a *monotone* category sequence.

Let's first assume that for every vertex $v \in V$, we would like to compute the shortest path from s to v among all paths with monotone increasing category sequence. The A* algorithm is iterative, and stores for every vertex v an implicit representation of a current shortest path from s to v with weight $d(v)$. We enforce the monotonicity constraint during the A* algorithm by storing for each vertex $v \in V$ its *current category* $cat(v)$, which is defined as the category of the edge of the current shortest path ending in v. During node expansion for a node u, only those edges (u, v) are considered for which $cat(u, v) \leq cat(u)$, where $cat(u, v)$ is the road category of this edge. The road category for v is then computed as $cat(v) \leftarrow cat(u, v)$. See Figure 4 for one iteration of this monotone A* algorithm. This simple modification of the A* algorithm ensures that only monotone sequences are considered. When $cat(u, v)$ is strictly less than $cat(u)$, this means that the routing algorithm jumped to a higher level in the hierarchical road network, and starts disregarding lower category links. Once having jumped to a higher category (i.e., a smaller number), the algorithm now stays at this level for this specific path that it explores. Note that the values of the current categories $cat(v), cat(v')$ for different vertices v, v' can be completely different, even if both vertices are incident to u via edges (u, v), (u, v'). This is because the current road category is *not* a global condition for all vertices in the open list, but a local condition for each vertex.

The advantage is that since we effectively reduce the overall size of the road network with each hierarchical jump, fewer nodes need to be subsequently explored and the performance of the algorithm in terms of memory consumption and computation speed is dramatically improved. Figure 2 illustrates the number of nodes existing for a route similar to Figure 1 when performing hierarchical jumps. During the middle portion of the search, the number of nodes that have to be evaluated is dramatically reduced. This is further evident when examining the number of nodes on a per-category basis of a road network. 70% and 50% of all nodes in the case of Athens and Vienna, respectively belong to the lowest category (local road of minor importance)! Thus, by eliminating this portion of the road network the performance of a shortest-path search will be considerably improved.

2.3.2 Hierarchical Bidirectional A* (HBA*)

We propose a new variant of a bidirectional A* algorithm to compute a shortest path from s to t that has a bitonic category sequence. This can be done in a quite straight-forward way by running two monotone A* algorithms in parallel: We simultaneously initiate a monotone increasing A* algorithm from s to t (the "front search"), as well as a monotone increasing A* algorithm from t to s (the "back search"). The two solutions will meet at some high level link in the "middle" of the computed route.

The HBA* algorithm is given in pseudo-code notation in Figures 3 and 4. Figure 3 initializes the respective data structures such as the open and the closed lists, path costs, and link categories. This function initiates the two respective A* searches and checks whether any search has terminated, upon which it reconstructs the resulting shortest path. The two respective searches are coordinated in that after execution of a single node expansion it is checked whether both algorithms still operate on the same link level. A search that has reached a higher level is halted until the other search has caught up.

One iteration of the monotone A* search algorithm is detailed in Figure 4. Each execution of this function only evaluates one node from the open list. The essential task of this algorithm is to extract the node with minimal $f(x)$ from the open list and expand it, i.e., check all neighboring links and place the current node on the closed list. This algorithm terminates once it finds a node from the closed list of the other search or reaches the target node (line 4). In line 6, a specific node is expanded by retrieving all neighboring nodes that can be reached through links that are at least of the category the current node was reached by. Note that for the back search, links have to be processed in the reverse direction. To provide a concise notation, this detail has been omitted in the pseudocode description of the algorithm.

The function StepCost(x, y) returns the travel time associated with the link that connects the two nodes x and y, and $category(x, y)$ is the category of the link (x, y). The heuristic function $h(x, y)$ needs to return the estimated travel time from node x to y. In our implementation, $h(x, y)$ is defined as the Euclidean distance between x and y, divided by the average speed of the network. The right

```
HBAStar(s, t)
    ▷ Initialize front search and back search
 1  CL_F, CL_B ← ∅ ▷ Closed lists
 2  OL_F, OL_B ← ∅ ▷ Open lists
 3  D[s], D[t] ← 0
 4  OL_F .insert(s, D[s] + h(s, t))
 5  OL_B .insert(t, D[t] + h(t, s))
 6  P[s], P[t] ← NULL ▷ Predecessor array, implicitly storing
                      ▷ the shortest paths tree
 7  CAT[s], CAT[t] ← ∞ ▷ Current category of nodes
 8  cat_F, cat_B ← ∞ ▷ Current category of search

    ▷ Start two-sided A* search
 9  m = NULL ▷ Node at which searches meet
10  while (front and back search are active)
11     if (cat_F ≥ cat_B and front search active)
12           m = AStar(s, t, OL_F, CL_F, CL_B, cat_F)
13     if (cat_B ≥ cat_F and back search active)
14           m = AStar(t, s, OL_B, CL_B, CL_F, cat_B)
15  ▷ Searches from both sides have terminated
16  if (m ≠ NULL)
17     return res = Path(s, m) ∘ Path(m, t)
18  else return NULL ▷ no path found
```

Figure 3: HBA* - control algorithm

```
AStar(s, t, OL, CL, CL', cat)
 1  if OL ≠ ∅
 2     x ← OL .removeMin() ▷ Node with smallest f-value
 3     CL .insert(x)
 4     if x = t or x ∈ CL'
 5           return x
 6     for each y ∈ Adj[x] with category(x, y) ≤ CAT[x]
           ▷ Link is of current category or better
 7           cost ← D[x] + StepCost(x, y)
 8           if (y ∉ OL and y ∉ CL) or (cost < D[y])
               ▷ Update y
 9              D[y] ← cost
10              CAT[y] ← category(x, y)
11              P[y] = x
12              cat ← min(cat, CAT[y])
13              if y ∈ OL
14                    OL .decreaseKey(y, D[y] + h(y, t))
15              else if y ∈ CL
16                    CL .remove(y)
17                    OL .insert(y, D[y] + h(y, t))
18              else
19                    OL .insert(y, D[y] + h(y, t))
20  else return NULL
```

Figure 4: One iteration of the monotone A* - search algorithm

choice of an average speed is critical since a small speed would overestimate the cost to the goal and thus eliminate valuable candidate solutions. Choosing a high speed underestimates the cost and thus keeps many candidate solutions, however possibly including too many unlikely routes that may inflate the size of the open list. For our setting, our choice of an ideal speed was 110km/h, which is less than the typical speed on highways but well above the speed encountered in inner-city routes. The function Path(s, m) extracts the computed shortest path from s to m using the predecessor array P. Proper concatenation of the two extracted paths from the front and the back searches (line 17 in Figure 3) yields a path with bitonic category sequence.

2.4 Algorithm Properties

In this section we show that given an admissible heuristic function and a strongly-connected hierarchical road network, the HBA* algorithm (see Figures 3 and 4) always terminates returning a bitonic path between source $s \in V$ and target $t \in V$, whose decreasing and increasing portions are each shortest.

Theorem 1. *If the HBA* algorithm uses an admissible heuristic function h, then for any $s, t \in V$, the algorithm computes a decreasing-increasing bitonic path from s to t, whose decreasing portion is shortest and whose increasing portion is shortest as well.*

Proof. The monotone A* algorithm (see Figure 4 for one iteration of it) by construction considers all monotone decreasing paths in G. The monotone A* algorithm basically equals the A* algorithm, with the only change being the category condition in line 6 and the addition of line 10 to update the current category of a node (see Figure 4); note that line 12 is only needed for coordinating the front search and back search. From the correctness and other related properties of the A* algorithm for admissible h [16, 9] follows that, at any time during the monotone A* algorithm with admissible h, if u is in the closed list then $d(u)$ equals the weight of a shortest monotone decreasing path from the source to $u \in V$. (If no such path exists, then the weight is considered to be ∞.)

The condition in line 4 of the monotone A* algorithm (see Figure 4) ensures that the HBA* algorithm stops with a found path iff a vertex m is found that is contained in both closed lists for the front and the back monotone A* searches. Hence, HBA* computes a path from s via m to t, such that the path from s to m is the shortest monotonically decreasing path from s to m, and the path from m to t is the shortest monotonically increasing path from m to t. □

Note that in theory it is possible to construct hierarchical road networks for which no bitonic path exists between two given vertices s and t. However, in Theorem 2 below we show that for hierarchical road networks that fulfill a reasonable connectivity assumption, which we introduced in Section 2.2, a bitonic path always exists for any choice of s and t, and hence HBA* always terminates finding a path.

Theorem 2. *If each level in a given hierarchical road network $G = (V, E)$ is strongly connected, then for any choice of $s, t \in V$ there exists a decreasing-increasing bitonic path from s to t in G.*

Proof. Let $L_0, \ldots, L_k$ be the levels of the hierarchical road network, with $k \geq 0$, and $L_i = (V_i, E_i)$. We show the claim by induction on k. Since L_0 is strongly connected, the category sequence of any path in L_0 between any two vertices $s, t \in V_0$ is constant $0, 0, \ldots, 0$, and hence trivially bitonic. Now assume the claim is true for k (inductive hypothesis), then it remains to show that the claim is true for $k + 1$. Let $s, t \in V_{k+1}$ be any two vertices. Remember that by definition, $V_k \subseteq V_{k+1}$. If both $s, t \in V_k$, then the claim

follows by the inductive hypothesis. Now assume $s \notin V_k$ or $t \notin V_k$ (or both), and let $\pi : s = v_0, v_1, \ldots, v_{l-1}, v_l = t$ be a path in V_{k+1} from s to t. If the category sequence for π is constant with category $k+1$, then it is also bitonic. Now assume the category sequence is not constant. Let (v_i, v_{i+1}) be the first edge in π with category less than $k+1$, and let (v_j, v_{j+1}) be the last edge in π with category less than $k+1$. This implies that $i <= j$, and $v_0, \ldots, v_i$ and $v_j, \ldots, v_l$, if non-empty, have constant category sequences with category $k+1$. (Note that one of these sequences may be empty, but not both.) By construction, $v_i, v_j \in V_k$, and by the inductive hypothesis there exists a path π' in L_k from v_i to v_j with bitonic category sequence, starting and ending with at most k. Therefore the concatenation $s = v_0, \ldots, v_i \circ \pi' \circ v_j, \ldots, v_l = t$ has a bitonic category sequence as well. □

Although it is not clear at first whether the front and the back search in HBA* algorithm have to meet, Theorem 2 shows that provided the hierarchical road network is strongly connected, HBA* is guaranteed to terminate with a found path. This *connectivity assumption* holds typically for any commercially available map dataset, i.e., connectivity at various levels of the road network hierarchy is a property guaranteed by the map data vendor. However, for all road network datasets used in our experiments it is certain that they fulfill this property (cf. Section 2.2).

Theorem 1 shows that if an admissible heuristic function is used, the algorithm is guaranteed to find a bitonic path that is optimal in the sense that the decreasing portion and the increasing portion are each optimal. Note that for road networks with travel time edge weights it is not obvious how to define a good heuristic function that is provably admissible (except for the trivial $h(u,t) = 0$). The conventionally used Euclidean distance does not yield a meaningful heuristic function in this case, since it does not encode time information. In our implementation we do however integrate the Euclidean distance, and compute $h(u,t)$ as the Euclidean distance divided by speed, see Section 2.3.2. There is however no provably correct value for the speed, as it could theoretically be arbitrarily large, and hence distance/speed arbitrarily small. For our applied setting, our choice of an ideal speed was 110km/h, which is less than the typical speed on highways but well above the speed encountered in inner-city routes.

Our HBA* algorithm, given an admissible heuristic function, computes a bitonic path with shortest decreasing and increasing portion but it does not necessarily compute the overall shortest bitonic path as it does not optimize over all possible split vertices. By running both the front search and the back search until all vertices have been discovered, an optimal split vertex and hence a shortest bitonic path can be computed. This, however, would not yield the desired speedup in runtime. Hence, we decided to opt for finding one bitonic path with shortest decreasing and increasing portion. The condition in lines 11 and 13 of the HBA* (see Figure 3), attempt to keep a balance between the decreasing and increasing portion of the computed path.

3. DATA

Road networks and related travel time data are an essential aspect for any meaningful route computation. In terms of road network, our experiments focus on the greater metropolitan areas of Athens, Greece and Vienna, Austria. The portion of the road network that was used has an extent of roughly 25×25km in each case.

To improve the quality of shortest-path solutions, one needs to carefully select the underlying weight database of the road network graph $DB(w(u,v))$ (cf. Section 2.1). Typically, weights in map data are static and correspond to link-based speed types, which are derived from the respective road category and its associated speed limit, or a speed type determined by costly road-side surveys.

In our experimentation, we also use a dynamic weight database in which the travel time associated with a link changes based on a temporal argument (speed profile). The idea is to derive dynamic weights from historical traffic assessment based on sensor measurements in the form of FCD. Using the causality between historical and current traffic conditions, weights in the form of *dynamic travel times* will replace static weights. To derive *dynamic travel time datasets* from collected FCD, map-matching algorithms are needed, as they relate GPS tracking data to the road network [3, 26]. The resulting travel times then need to be aggregated using a data warehouse architecture such as described in [17, 18].

In our case speed profiles have been compiled using FCD from a commercial vehicle fleet (Athens) and from a taxi fleet (Vienna). The size and nature of the dynamic travel time datasets used, is described in [18].

The second weight dataset used in the experiments comprises only *static travel times*. They are derived from speed information that is part of the road network dataset. In our case, speeds range from 90km/h for inner city highways to 15km/h for local roads.

4. EXPERIMENTAL EVALUATION

The objective of our experiments is (i) to compare the performance of the HBA* algorithm in terms of quality of result as well as (ii) to assess the respective impact of using dynamic travel times. To evaluate the HBA* algorithm we use a standard A* algorithm to benchmark its performance.

4.1 Tuning HBA*

The HBA* algorithm exploits an important aspect of the road network, its hierarchical structure, to improve its running time. In utilizing hierarchical jumping, the algorithm mimics human driving behavior, i.e., when given the choice, it selects higher category roads to reach a destination. On the other hand, hierarchical jumping, especially when used early on in shortest-path search may eliminate candidate solutions and provide suboptimal results. To address this potential issue, we introduce an *initialization buffer*. I.e., assume that we want to compute a shortest path from s to v, then the initialization buffer $I(\epsilon)$ around both s and v prevents the use of hierarchical jumping for all vertices
$u \in I(\epsilon) : \{dist(u,v) < \epsilon \vee dist(u,s) < \epsilon\}$.

Using this approach, the HBA* essentially postpones the choice of a higher category road in search for a potentially better alternative. As we shall see in the following experimentation, this modification improves the quality of the solutions by only minimally impacting the algorithmic cost.

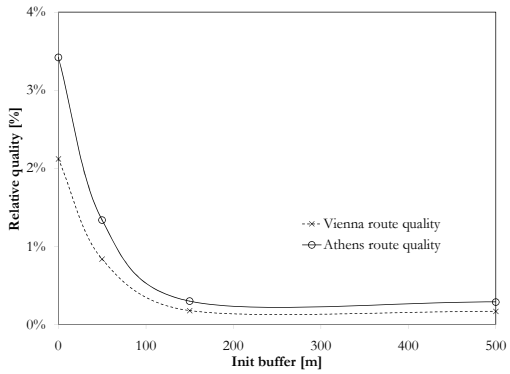
In our experimentation we utilized $\epsilon = [0, 50, 150, 500]m$.

4.2 Shortest Path Quality

To assess performance, the two algorithms are compared using a set of 150 shortest path problems, each uniformly distributed over the city area. The Euclidean distance between the nodes ranges from a couple of 100m to 20km.

Using the same heuristic function as defined in Section 2.3.2, Figure 5(a) shows that a plain HBA* ($\epsilon = 0$) in the case of *static weights* produces solutions that are on average 2% and 3.5% longer than those of the A* for Vienna and Athens, respectively.

The absolute shortest-path results for Athens (Vienna results are similar) in terms of travel time indicate that for 45% and 50% of the routes (Athens and Vienna), the two algorithms find the same solution. It is interesting that the plain HBA* algorithm performs this well, even though it considers only a subset of possible paths



(a) Quality and static weights (b) Cost and static weights

Figure 5: HBA* initialization buffer: shortest-path quality and running time

Figure 6: Running time, static weights, Athens

(namely, bitonic paths) and it heuristically stops immediately when the front and back searches meet. This shows that coordinating the front and back searches by roughly staying on the same level of the road network (lines 11-14 in Figure 3) yields surprisingly good results. The quality of the HBA* can be improved dramatically when using the initialization buffer. Figure 5(a) shows that for static weights and, e.g., $\epsilon = 150m$ the quality of the shortest-path solution is within 0.2% of that of the A* algorithm, with the gap between A* and HBA* closing with an increasing initialization buffer.

In the case of *dynamic weights* the quality disadvantage of the HBA* increases to 3.5% and 8% respectively. The main reason for this are the lower actual travel times measured for higher category roads.

4.3 Computation Cost

A major advantage of the HBA* algorithm is its improved running time due to the fact that it evaluates much fewer nodes of the road network graph when computing a shortest path.

It is evident from Figure **??** that the plain HBA* algorithm computes a shortest path 20 times faster than the A* algorithm (less than 5% of its running time). Figure **??** provides more detail for the Athens dataset. A typical run of the HBA* algorithm takes in the range of 0.0001s (short routes) to 0.01s (long routes), while the running time of the A* algorithm may be up to 3s.

4.4 Static vs. Dynamic Travel Times

Having two alternative shortest-path algorithms, each with its respective strength, the following section assesses qualitative improvements to algorithmic solutions *utilizing dynamic travel times*. As such, this section goes beyond evaluating the respective algorithms but assesses the potential of dynamic weights for improving routing solutions. In the specific experiments, in addition to static weights, two dynamic travel time datasets for 8h and 13h on Mondays were used. This selection was based on available travel time data (collected FCD) and expected impact on the routing solutions, i.e., common sense expectations could be that 8h constitutes rush hour, whereas 13h means smoother traffic.

Table 1: Qualitative improvements using dynamic travel times

Athens

	HBA*		A*	
	affected	imp.	affected	imp.
8h	55%	11.5%	65%	11.9%
13h	54%	11.1%	60%	10%

Vienna

	HBA*		A*	
	affected	imp.	affected	imp.
8h	53%	9.1%	59%	9.8%
13h	41%	6.1%	40%	7.1%

For the quintessential experiment with respect to dynamic travel times, i.e., *to assess the improvement of the route quality*, we compared dynamic weights to static weights (cf. Figure 6), by recomputing for the latter the time it would take to traverse those routes using the more accurate dynamic travel times! Table 1 gives the percentage of the shortest-path solutions that were improved and the average improvement in each case. For example, for Athens, using HBA* and the dynamic travel time dataset for 8h, 55% of the 150 shortest-path solutions were improved by an average of 11.5%. Examining the improvements in detail shows that the improvements may range from 27% to somewhat less than 1%.

The following observations can be made from these experiments:

- the shortest-path solutions for Athens benefit more from the dynamic travel time data - this can be attributed to greater fluctuations in traffic conditions for Athens, i.e., Athens exhibits greater fluctuations in its speed profiles than Vienna.

- a "rush-hour" effect can be observed when comparing 8h to 13h shortest path improvements. This effect is more evident in the Vienna data, e.g., in case of the A* algorithm yielding 9.1% vs. 6.1% improvement for 8h and 13h, respectively.

5. CONCLUSIONS AND FUTURE WORK

The HBA* algorithm computes shortest paths much more efficiently than the A* algorithm does. Utilizing an initialization buffer, (i) the quality of solutions is virtually identical to that of the A* algorithm and (ii) its running time is 10 times faster. While dynamic weights slightly affect the performance of the algorithms, they improve the quality of individual shortest-path solutions on average by 10%. Thus, when used, dynamic travel times represent a significant qualitative step for shortest-path solutions. Overall, combining dynamic travel times with the HBA* algorithm generates a *fast and accurate solution base* for routing and navigation applications.

Our ongoing and future work is in the area of developing an adequate data management techniques and improve the performance of the HBA* for dynamic travel times by further exploitation road network topologies.

Acknowledgment

The work of Dieter Pfoser and Alexandros Efentakis was partially supported by the TRACK&TRADE project, funded by the European Commission under contract number COOP-CT-2006-032823. Carola Wenk's work has been supported by the National Science Foundation grants NSF CCF-0628809 and NSF CAREER CCF-0643597.

6. REFERENCES

[1] A. Botea, M. Mueller and J. Schaeffer. Near Optimal Hierarchical Pathfinding. *J. of Game Development*, 1:28–35, 2004.

[2] P. Bovy and E. Stern. *Route Choice: Wayfinding in Transportation Networks*. Kluwer Academic Publishers, Dordrecht, 1990.

[3] S. Brakatsoulas, D. Pfoser, R. Sallas, and C. Wenk. On Map-Matching Vehicle Tracking Data. In *Proc. 31st VLDB conf.*, pages 853–864, 2005.

[4] E. Brockfeld, P. Wagner, and B. Passfeld. Validating travel times calculated on the basis of taxi Floating Car Data with test drives. In *Proc. ITS World Congress*, 2007.

[5] A. Car and A. U. Frank. General principles of hierarchical spatial reasoning - the case of wayfinding. In *Proc. 6th SDH symp.*, 1994.

[6] Cityrouter. [Online]. Available: http://www.cityrouter.net/

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001.

[8] Dash Inc. Dash express - traffic powered by the dash driver network. [Online]. Available http://blog.dash.net/2008/03/18/dash-express-traffic-powered-by-the-dash-driver-network

[9] R. Dechter and J. Pearl. Generalized Best-First Search Strategies and the Optimality of A*. *Journal of the ACM*, 32(3):505–536, 1985.

[10] E. W. Dikjstra. A Note on two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

[11] L. Fu, D. Sun, and L. R. Rilett. Heuristic shortest path algorithms for transportation applications: state of the art. *Comput. Oper. Res.*, 33(11):3324–3343, 2006.

[12] P. Hart, N. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. Syst. Science and Cybernetics*, 4(2):100–107, 1968.

[13] INRIX Inc. Company Website. [Online]. Available: http://www.inrix.com/

[14] JGraphT - a free Java Graph Library. [Online]. Available: http://jgrapht.sourceforge.net/

[15] B. Liu. Intelligent route finding: Combining knowledge and cases and an efficient search algorithm. In *Proc. 12th ECAI*, 1996.

[16] N. Nilsson. *Principles of Artificial Intelligence*. Tioga, Palo Alto, California, 1980.

[17] D. Pfoser, N. Tryfona, and A. Voisard. Dynamic Travel Time Maps - Enabling Efficient Navigation. In *Proc. 18th SSDBM conf.*, pages 369–378, 2006.

[18] D. Pfoser, S. Brakatsoulas, P. Brosch, M. Umlauft, N. Tryfona, and G. Tsironis. Dynamic Travel Time Provision for Road Networks. In *Proc. 16th ACM SIG SPATIAL conf.*, pages 475–478, 2008.

[19] I. Pohl. Bi-directional search. *Machine Intelligence*, 6, 1971.

[20] R.-P. Schaefer, K.-U. Thiessenhusen, and P. Wagner. A Traffic Information System by Means of Real-time Floating-car Data. In *Proc. ITS World Congress*, 2002.

[21] S. Shekhar, A. Fetterer, and B. Goyal. Materialization trade-offs in hierarchical shortest path algorithms. In Proc. 5th SSD smyp., pages 94–111, 1997.

[22] L. Sint and D. de Champeaux. An improved bidirectional heuristic search algorithm. *J. of the ACM*, 24(2), 1977.

[23] Tele Atlas. *Tele Atlas MultiNet Shapefile 4.3.1 Format Specifications*, 2005.

[24] S. Timpf, G. S. Volta, D. W. Pollock, and M. J. Egenhofer. A conceptual model of wayfinding using multiple levels of abstraction. In *Proc. Intl. Conference GIS - From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning*, 1992.

[25] TomTom. IQ Routes. Product page. [Online]. Available: http://www.tomtom.com/page/iq-routes

[26] C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. In *Proc. 18th SSDBM conf.*, pages 379–388, 2006.

[27] T. K. Whangbo. Efficient modified bidirectional A^* algorithm for optimal route-finding. In *Proc. 20th Int'l Conf. on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE)*, 2007.