Using turn restrictions for faster route planning with partitioned road networks

Int. Conf. Integr. Navigation Systems
St Petersburg

Using Turn Restrictions for Faster Route Planning with Partitioned Road Networks [#]

Ingrid Flinsenberg[*]
Embedded Systems Institute – Eindhoven University of Technology,
Laplace Building 0.10, P.O. Box 513, 5600 MB Eindhoven, The Netherlands, E-mail: I.Flinsenberg@tue.nl

Abstract

Key words: route planning, turn restrictions, graph partitioning, car navigation systems

We investigate the planning of optimum routes in large real-world road networks, by using a graph partitioning approach. Very specific for real-world road networks is the presence of turn restrictions. We consider the problem of planning optimum routes in partitioned graphs, where the original graph, and thus the created reduced graph, contains turn restrictions. We discuss the consequences of the presence of turn restrictions for the creation of reduced graphs. Then we create a new reduced graph based on the original graph partition. When constructing this new reduced graph, we use the presence of turn restrictions in the original graph to our advantage. We develop a new planning algorithm to plan with the new reduced graph and the original graph partition. We compare the number of function evaluations of this algorithm for the old and new reduced graph, for several planning criteria on several real-world road networks. We conclude that our algorithm performs significantly better for the new reduced graph than for the old one. Our algorithm applied to the old reduced graph is, in turn, faster than the generalized $A*$-algorithm for unpartitioned graphs.

Introduction

An increasing number of car manufacturers are offering a navigation system as one of the extra features of their cars. The key components of such a navigation system are positioning, guidance and route planning. In this paper, we focus on the route planning functionality of a car navigation system. Users of such systems are becoming increasingly demanding. Because a driver does not like to wait for his route, the route planning process has to be very fast. Furthermore, the size of road networks available on CD or DVD is increasing. CDs or DVDs containing the road network of entire Europe are now becoming available. Also, when a driver asks the system for the fastest route to his destination, he expects the system to present the absolutely fastest route. So optimum routes have to be planned on very large road networks, in very little time. Because of the increasing size of available road networks and higher demands on route quality and planning speed, planning optimum routes in little time is a continuing challenge for companies developing car navigation systems.

Very specific for road networks is the presence of turn restrictions. Turn restrictions can be modeled by costs on adjacent edges, so that certain turns can be given an additional cost. Turn restrictions have been studied recently by Schmid [10], Szeider [11] and Winter [12]. Schmid [10] discusses forbidden turns or equivalently, turn restrictions with infinite costs. He presents several algorithms and graph reformulations for planning optimum routes if forbidden turns occur in a road network. Winter [12] considers turn costs on pairs of adjacent edges. He constructs the line graph of the road network, and proves that optimum routes in the original road network can be planned by applying a standard shortest-path algorithm to the line graph. Szeider [11] considers the problem of determining whether a simple path exists between two nodes in a road network with turn restrictions, where a simple path is a route in which each node appears at most once. He proves that this problem is NP-complete for real-world road networks.

For a car-navigation system, a standard Dijkstra-like algorithm [1], [4] is not fast enough to plan optimum routes in large real-world road networks. Because of the high demands on planning speed, the route planning process has to be speeded up, which can be done by pre-processing the roadgraph. Flinsenberg [3] and Jung and Pramanik [6] describe a cell-partitioning approach to speed up the planning process. They divide the roadgraph into a number of disjunct subgraphs, called cells, that are connected by a boundary graph. The planning process is speeded up, by reducing the graph that is needed to plan the optimum route. Kim, Yoo and Cha [7] discuss handling real-time data in combination with cell-partitions. They add all edges that could be subject to real-time data to the boundary graph. Henzinger et al [5] use the graph separators of Lipton and Tarjan [8] to achieve a faster route planning process. Their approach gives efficient theoretic bounds on the running time of the

[#] This research was performed in co-operation with Siemens VDO Automotive in Eindhoven, The Netherlands.
[*] M.A.,Ph.D.-student.

algorithm, but we do not consider this approach to be practically feasible. Ertl [2] creates an implicit hierarchy by creating a 'radius' for every edge. Only if the distance from the start node or destination to the end node of an edge is smaller than the radius, the edge is evaluated when planning an optimum route between the two nodes.

This paper is organized as follows. In Section 2, we discuss planning optimum routes with turn restrictions. In Section 3, we introduce a cell-partition, and in Section 4 we discuss the consequences for creating cell-partitions for real-world road networks with turn restrictions. We discuss how turn restrictions can be used to achieve faster route planning by creating a new boundary graph in Section 5, and in Section 6, we give a new planning algorithm for planning with this boundary graph. We compare the number of evaluated edges of our planning algorithm when the old and new boundary graph are used in Section 7. First, we introduce some basic notation.

1. Basic notation

A road network can be represented as a graph, in which the edges represent the road segments, and the nodes represent the junctions. Since there may exist parallel and circular roads, we do not exclude parallel edges or loops. Because one-way roads have to be modeled as well, every edge in a road network is directed. A road network can thus be represented by a directed multi-graph. Very specific for road networks is the presence of turn restrictions. Turn restrictions can be modeled by costs on adjacent edges, so that certain turns can be given an additional cost. Such costs can be used to represent forbidden turns, but also to increase the cost of using a short-cut instead of the main road.

For an edge e from node u to node v, let $\delta_1(e)$ denote start node of the edge, and $\delta_2(e)$ the end node of the edge, i.e. $\delta_1(e)=u$ and $\delta_2(e)=v$. Formally, we define the *roadgraph* as a tuple $G=(N,E,w,r)$, where N denotes the set of nodes, E the set of edges, $w(e)$ the non-negative cost associated with edge e, and $r(e_1,e_2)$ the cost associated with edges e_1 and e_2 with $\delta_2(e_1)=\delta_1(e_2)$. The cost $r(e_1,e_2)$ is called a *rule*, and is used to model the additional cost of making a 'turn' from edge e_1 to edge e_2. If a turn is not allowed, for example due to a turn restriction, we call the turn *forbidden*, and we set $r(e_1,e_2)=\infty$. A *route* in a roadgraph is a sequence of adjacent edges, $p=(e_1,\ldots,e_k)$, where $e_i\in E$ and $\delta_2(e_i)=\delta_1(e_{i+1})$ for $i=1,\ldots,k-1$. The cost of route p, the *route-cost*, is equal to $c(p)=\sum_{i=1}^{k}w(e_i)+\sum_{i=1}^{k-1}r(e_i,e_{i+1})$. If a route uses two adjacent edges that are not allowed because of a forbidden turn, the route-cost equals infinity. A route with finite cost is called a *feasible route*. A route with minimum cost from *start node* s to *destination node* d is called a *minimum cost route* or an *optimum route*.

2. Planning optimum routes in a roadgraph with turn restrictions

Because forbidden turns may be present in a roadgraph, an optimum route may contain a node more than once. Consider the example in Figure 1, and assume that $r(e_1,e_2)=\infty$. The optimum route from node 1 to node 6 is the route through all six nodes, which passes through node 2 twice. Note that this cyclic route is the only feasible route from node 1 to node 6.

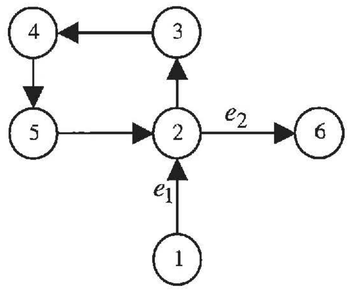

Figure 1: An optimum route with a cycle.

Because an optimum route may contain a node more than once, the standard $A*$-algorithm [4] cannot be used to determine the optimum route. However, an optimum route does not contain an edge more than once. In order to plan optimum routes in a graph with rules, a modified $A*$-algorithm can be used that evaluates edges instead of nodes. Schmid [10] and Winter [12] consider turn restrictions on pairs of adjacent edges. Both papers present polynomial-time algorithms that construct optimum routes that may contain nodes more than once. Winter [12] uses that an optimum route does not contain an edge more than once by constructing the line graph of the roadgraph to plan optimum routes.

3. Cell-partition

Since a driver does not like to wait for his route to be planned, the planning of optimum routes in a roadgraph has to be done very fast. The route planning process can be speeded up, by pre-processing the roadgraph, as was shown by Flinsenberg [3] and Jung and Pramanik [6]. They divide the roadgraph into a number of disjunct subgraphs called *cells* in order to achieve faster route planning. Let $G = (N, E, w, r)$ be a roadgraph, then a *cell-partition* is a set $\{C_1,\ldots,C_k\}$ where $C_i = (N_i, E_i, w, r)$ is a roadgraph induced by the nodes N_i, such that $N_i \cap N_j = \emptyset$, for every $i \neq j$, and $\bigcup_{i=1}^{k} N_i = N$. The edge costs and rules of *cell* C_i are equal to the edge costs and rules of those edges and rules of G that are completely contained in cell C_i. The edges of cell C_i are called *internal edges*, the nodes that only have adjacent nodes in C_i are called *internal nodes*, and nodes that also have adjacent nodes outside cell C_i are called *boundary nodes*. The *boundary graph* B of a cell-partition $\{C_1,\ldots,C_k\}$ is defined by $B = (N_B, E_B, w, r)$ with N_B the collection of *boundary nodes* and E_B the collection of *boundary edges* with $E_B = E \setminus \{\bigcup_{i=1}^{k} E_i\}$. The edge costs are given by the costs of the corresponding edges in G, and the rules are given by the rules in G formulated on pairs of edges of which both edges are contained in B. After the roadgraph has been partitioned into a number of cells, we compute the optimum route cost between every pair of boundary nodes of a single cell. The optimum routes are represented by edges that we add to the boundary graph. These edges are called *route edges*, and the set of route edges of cell C_i is denoted by A_i. Specifically, we add two directed edges between every pair of boundary nodes of a single cell. The set of route edges of a single cell thus forms a directed clique. We denote the graph formed by the set of route edges by K_i, so $K_i = (N_B \cap N_i, A_i, w, r)$ for each cell C_i. The total set of route edges is denoted by $E_A = \bigcup_i A_i$. Define $A = (N_B, E_A, w, r)$. The cost of a route edge between two boundary nodes of cell C_i is equal to the minimum route cost in this cell from the start node to the end node of the edge. For a roadgraph without turn restrictions, A does not contain rules. Finally, we define the *searchgraph* G_S for planning an optimum route between start node s and destination d as $G_S = C_s \cup C_d \cup B \cup A \setminus (K_s \cup K_d)$, where C_i is the cell containing node i, and K_i the graph formed by the set of route edges of cell C_i. Flinsenberg [3] shows that planning a route in searchgraph G_S is much faster than in the entire roadgraph. An example of a cell-partition and the boundary graph including route edges are given in Figure 2 and Figure 3 respectively. A cell is represented by the graph contained in an oval. Note that the boundary nodes are black and the internal nodes are white.

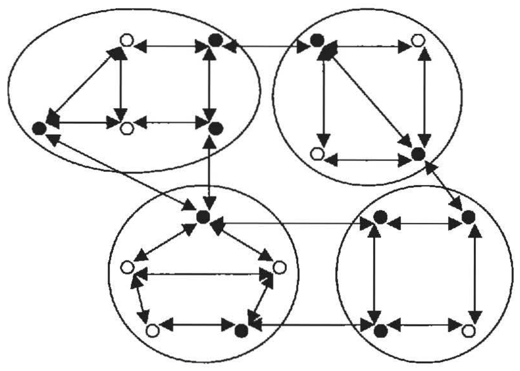
Figure 2: A cell-partition.

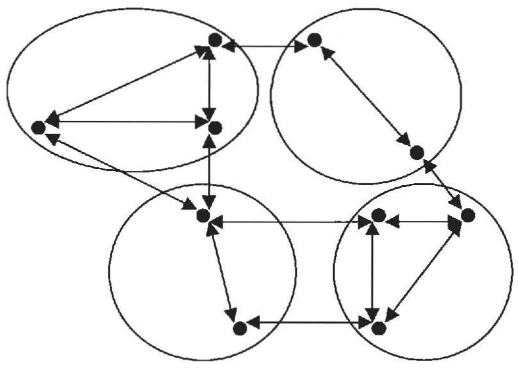
Figure 3: Boundary graph and route edges.

4. Cell-partitions of roadgraphs with turn restrictions

We now elaborate on the use of cell-partitions when turn restrictions are present. In Figure 4 the numbers next to the edges represent both the edge cost and the edge number. A cell is given by the sub-graph induced by the nodes contained in an oval. Assume the combination of edges 6,1 corresponds to a forbidden turn, i.e. $r(6,1) = \infty$. This means that the optimum route from node 4 to node 9 is the route $(6,3,4,13)$, with cost 26. The optimum route from node 5 to node 8 is the route $(1,2)$, so the searchgraph contains an edge from node 5 to node 8 with cost 3. As a result, a standard route-planning algorithm such as Dijkstra's algorithm [1], planning the optimum route from node 4 to 9 in the searchgraph, finds a route with cost 22. So in this case, planning an optimum route with the searchgraph does not lead to a route with the correct minimum route cost. The problems arise because the optimum route between two nodes contains a route different from the optimum route between two boundary nodes. This is caused by the turn restrictions imposed on a pair of edges, for which not both edges are contained in a single cell or in the boundary graph.

Now assume that the turn from edge 6 to edge 7 is also forbidden, i.e. $r(6,7) = \infty$. As a result the optimum route from node 4 to node 2, is the (cyclic) route $(6,3,5,1,7)$ with cost 22. The searchgraph consists of the cell that contains node 4, the boundary graph and the route edges, i.e. $G_S = C_4 \cup B \cup A \setminus K_4$. As a result, G_S does not contain nodes 6 and 7, and a route-planning algorithm can only find a route via node 8. We can solve this problem, by adding a boundary edge between every boundary node and itself, with as cost the minimum cost of a path from the boundary node to itself containing at least one edge. However, to overcome all difficulties with turn restrictions in the cell-partition, we have to modify the partition itself. This is practically feasible because the number of rules is small compared to the number of edges and nodes in a real-world roadgraph. We modify the cell-partition so that there are no rules between internal edges and boundary edges, or between two boundary edges. As a result, every rule of the roadgraph is completely contained in a single cell.

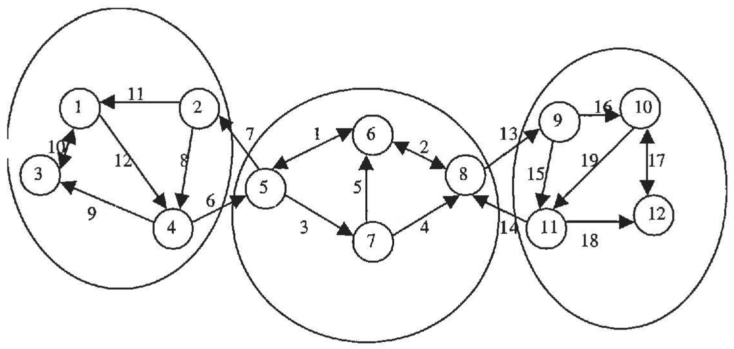

Figure 4: Turn restrictions in a cell-partition.

We also have to introduce rules between route edges in order to guarantee that optimum routes can be planned. Consider Figure 5 and assume there is a turn restriction between edges e_1 and e_2, i.e. $r(e_1, e_2) = \infty$. The routes from node 1 to node 2 and from node 2 to node 3 are feasible, but the route from node 1 to node 3 is infeasible. As a result, we need to create turn restrictions between the route edges that start and end in node 2, to prevent the route from node 1 via node 2 to node 3 from becoming feasible.

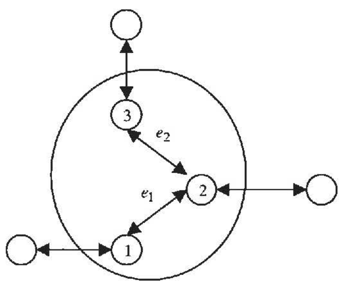

Figure 5: Turn restrictions between route edges.

Therefore, we also introduce rules $r(e_1, e_2) = \infty$, for all route edges e_1 and e_2, such that $\delta_2(e_1) = \delta_1(e_2)$. Due to these rules, a feasible route never contains two adjacent route edges. Let A be an algorithm suitable for planning optimum routes in a graph with turn restrictions. Such an algorithm exists, as was shown by

Schmid [10] and Winter [12]. With the resulting searchgraph, we can plan optimum routes using Algorithm A, as formulated by Lemma 4.1, where $\{C_1,\ldots,C_k\}$ is a cell-partition of roadgraph G.

Lemma 4.1

If $\bigcup_{i=1}^{k}\{r(e_1,e_2)\,|\,r(e_1,e_2)>0, e_1,e_2\in C_i\}=\{r(e_1,e_2)\,|\,r(e_1,e_2)>0, e_1,e_2\in G\}$, *then an optimum route between start node s and destination node d in roadgraph G with cost c can be planned using algorithm A, if and only if an optimum route between start node s and destination node d in G_S with cost c can be planned using Algorithm A.*

5. Using turn restrictions for faster route planning

As stated in Section 3, Flinsenberg [3] creates two directed edges between all pairs of boundary nodes of a cell with as cost the minimum path cost within that cell. This leads to a quadratic number of edges between these nodes. Specifically $n(n-1)$ route edges are created for a cell with n boundary nodes. In this section, we show that this can be reduced to a linear number of edges, specifically to $2n$ route edges for a cell with n boundary nodes. First, we introduce the structure of the new boundary graph, then we demonstrate that using this structure, the optimum route costs between every pair of boundary nodes remains the same. Finally, we argue how planning with this new structure relates to planning with the old 'clique graph'.

To create a clique graph, an edge is added between every pair of boundary nodes of a cell, which leads to a graph as in Figure 6. The $n(n-1)$ route edges can be reduced to $2n$ route edges by storing the optimum route costs in a *star graph*. We create a star graph by adding a *dummy node*, and two directed edges between the dummy node and every boundary node, see Figure 7. This leads to $2n$ route edges. Note that for the clique graph we had n boundary nodes per cell, and for the star graph we have $n+1$ (boundary) nodes per cell.

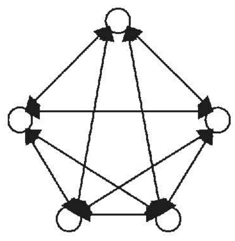
Figure 6: Clique graph.

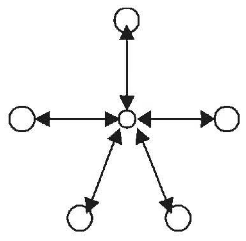
Figure 7: Star graph.

What remains to be done, is establishing the edge costs of the edges in the star graph such that the costs of the routes between every pair of boundary nodes remain the same as in the clique graph. We do that by introducing rules between every pair of adjacent edges in the star graph. Consider a cell C_k with n boundary nodes, and let c_{ij} denote the minimum route cost in C_k between boundary nodes i and j with $i,j=1,\ldots,n$ and $i\neq j$. We create a star graph $S_k=\left(\{1,\ldots,n\}\cup\{u\},\{e\,|\,\delta_{i+1}(e)=j,\delta_{2-i}(e)=u, i=0,1, j=1,\ldots,n\}, w, r\right)$. The cost of each edge in the star graph is set to half the minimum route cost between all pairs of boundary edges rounded down, i.e. $w(e)=c=\min_{i,j\in S_k}\left\lfloor \frac{1}{2}c_{ij}\right\rfloor$ for all $e\in S_k$. We introduce rules $r(e_1,e_2)=c_{ij}-2c$, for every pair of boundary nodes i and j with $i=\delta_1(e_1)$, $j=\delta_2(e_2)$, $\delta_2(e_1)=\delta_1(e_2)=u$, and $i\neq j$. For each boundary node we introduce two more rules. Specifically, we introduce rules $r(e_1,e_2)=\infty$, for route edges e_1 and e_2 in S_k such that $i=\delta_1(e_1)=\delta_2(e_2)$, and such that $\delta_2(e_1)=\delta_1(e_2)=i$ for every boundary node $i\in S_k$. These last rules, which forbid certain edge combinations, are introduced for the same reason as the rules in the clique graph. That is, to prevent forbidden routes from becoming feasible, see Figure 5. Using $A=(N_A,E_A,w,r)=\bigcup_k S_k$, the searchgraph for start node s and destination d is given by $G_S=C_s\cup C_d\cup B\cup A\setminus(S_s\cup S_d)$. With these edge costs and rules, we have that between every pair of boundary nodes i, j the minimum route cost is equal to c_{ij} as it is in the clique graph. This is formally stated in the next lemma.

Lemma 5.1

Let u and v be two boundary nodes of cell C_k. There exists a path p from u to v in K_k with cost c if and only if there is a path p' from u to v in S_k with cost c.

From Lemma 5.1, it follows that Lemma 4.1 remains valid when star graphs are created instead of clique graphs. When we need to plan a route in a boundary graph that is constructed with these star graphs, we have to use an algorithm that is suitable for taking restrictions on pairs of adjacent edges into account. However, because turn restrictions are also present in a normal road network, this is not an additional requirement. Also for the clique graphs we needed to use an algorithm suitable for taking rules on pairs of adjacent edges into account. So, in both structures it is necessary to check for rules on pairs of edges. Therefore, the additional turn restrictions do not influence the speed of a single iteration of the algorithm. Note however that in the star graph, the number of rules is usually larger.

6. Route planning with star graphs

Dijkstra's algorithm [1] can be used to plan an optimum route in a graph without turn restrictions: it repeatedly selects the node with minimum cost from the start node to be evaluated. Because we have to take turn restrictions into account, we evaluate edges instead of nodes, and thus select the edge with minimum cost from the start node to be evaluated next. In order to reduce the number of evaluated edges, the $A*$-algorithm [4] uses an estimation of the cost from the current node to the destination, which is called the h-value. The minimum cost from the start node to the current node is called the g-value. The $A*$-algorithm selects the node with minimum expected cost from the start node to the destination, i.e. the node with minimum $g+h$-value. Because of the turn restrictions, we select the *edge* with minimum expected cost from the start node to the destination. The $A*$-algorithm can be used to plan optimum routes if the h-value under-estimates the real cost from the current node (or edge in our case) to the destination, and if the h-value is a so-called dual feasible estimator. The Euclidean distance from a node to the destination is an under-estimation of the remaining distance to the destination, and it is also a dual feasible estimator. For nodes with a geographical location associated with it, the Euclidean distance to the destination can be determined. Because we are concerned with real-world roadmaps, every node in the roadgraph has a geographical location. We can use the Euclidean distance as h-value if the edge costs are equal to the length of the edge. For edge costs equal to the driving time, the Euclidean distance divided by the overall maximum speed can be used as h-value to plan optimum routes.

As noted in the last section, we need to check the presence of turn restrictions on pairs of adjacent edges, for both the star and the clique graph. Therefore, both approaches need to evaluate edges instead of nodes. However, there is a difference with respect to planning with a star graph compared to a clique graph. This difference only occurs for a planning algorithm that selects the edges to be evaluated partly on the Euclidean distance from the evaluated edge to the destination. The $A*$-algorithm that uses a h-value based on the Euclidean distance from the end node of an edge is such an algorithm. The difference between both structures lies in the fact that the added dummy nodes in the star graphs do not exist in reality and therefore have no geographic location associated with them. Without a geographic location, the remaining Euclidean distance is not defined. We first define the h-value for an edge ending with a dummy node, and then we show that this can be used to plan optimum routes in a searchgraph that contains star graphs.

Definition 6.1

Let $G_S = (N, E, w, r)$ be a searchgraph that contains star graphs and a set of dummy nodes U. Let $\tilde{h}(u)$ denote the expected remaining cost from node $u \in N \setminus U$, to destination $d \in N \setminus U$.

Define: $$h(e) = \begin{cases} \tilde{h}(\delta_1(e)) - w(e), & \text{if } \delta_2(e) \in U \\ \tilde{h}(\delta_2(e)), & \text{if } \delta_2(e) \notin U \end{cases}.$$

The h-value of edge e is equal to the expected remaining cost from the end node of the edge to the destination. We show that the expected remaining cost defined above is an under-estimation of the actual remaining cost. Furthermore, a permanent label set by a labeling algorithm is never modified if all rule costs are non-negative and if the expected remaining cost $h(e)$ under-estimates the remaining cost and is a dual feasible estimator. If $\tilde{h}(u)$ under-estimates the remaining cost then $h(e)$ also under-estimates the remaining cost because the remaining cost from the end-node of e with $\delta_2(e) \in U$, is equal to the remaining cost from $\delta_1(e)$ minus the cost of edge e, $\tilde{h}(\delta_1(e)) - w(e)$. Remains to be proven that $h(e)$ is a dual feasible estimator and that a permanent label set by a labeling algorithm is never modified if all rule costs are non-negative and if the expected remaining cost $h(e)$ is a dual feasible estimator. Recall the definition of a dual feasible estimator, see also Pearl [9].

Definition 6.2

An estimator $\tilde{h}(u)$ is called a dual feasible estimator *if $\tilde{h}(\delta_2(e)) - \tilde{h}(\delta_1(e)) \le w(e)$ for every $e \in E$ of roadgraph $G = (N, E, w, r)$.*

This definition can be easily generalized for estimators for edges. Pearl [9] proves that for roadgraphs without turn restrictions, a permanent label set by a labeling algorithm is never modified if the expected remaining cost $\tilde{h}(u)$ is a dual feasible estimator. This can be easily generalized to roadgraphs with turn restrictions if all rule costs are non-negative. Finally, we can show that the estimator defined in Definition 6.1 is a dual feasible estimator.

Lemma 6.1

Let $G = (N, E, w, r)$ be a roadgraph. If $\tilde{h}(u)$ is a dual feasible estimator, then $h(e)$ is a dual feasible estimator.

Figure 8 shows algorithm T^* for planning optimum routes in searchgraphs with turn restrictions that contain stars.

```
Input: Cell-partition $\{C_1, \ldots, C_k\}$, boundary graph $B$, roadgraph $A = \bigcup_{i=1}^{k} S_i$, start node $s$ and destination $d$.

    1. Create the search graph.
         $G_S = (N, E, w, r) = C_s \cup C_d \cup B \cup A \setminus (S_s \cup S_d)$.
    2. Create sets of unexpanded edges.
         $S_E = \{e \in B \mid \delta_1(e) \in C_s\}$, $D_E = \{e \in B \mid \delta_2(e) \in C_d\}$, $H = E$.
    3. Initialize all costs.
         $c(e) = \infty, \forall e$, $c_n(u) = \infty, \forall u$, $c(e) = w(e) + h(e)$ for all edges adjacent to node $s$, and $c_n(s) = \tilde{h}(s)$.
    4. Select the edge $e$ from $H$ with minimum cost $c(e)$.
    5. If $c_n(\delta_2(e)) \ge c_n(d)$, or no edge is selected, then stop.
    6. For every (allowed) adjacent edge $e_1$ in $G_S$ of edge $e$, update the minimum cost $c(e_1)$, and
         $c_n(\delta_2(e_1))$, using the $h$-value of Definition 6.1.
    7. Update the sets of unexpanded edges.
         $S_E = S_E \setminus \{e\}$, $D_E = D_E \setminus \{e\}$, $H = H \setminus \{e\}$.
    8. If possible, reduce the set $H$ of unexpanded edges.
         If $(C_s \ne C_d) \wedge (S_E = \emptyset)$ then $H = H \setminus E_s$. If $(C_s \ne C_d) \wedge (D_E = \emptyset)$ then $H = H \setminus (E_B \cup E_A)$.
    9. Go to Step 4.

Output: minimum route cost $c_n(d)$.
```

Figure 8: Algorithm T^*.

Algorithm T^* uses the dual feasible estimator of Definition 6.1 to direct the planning process to the destination (Step 6). In Steps 5, 7 and 8, we use that the used estimator is dual feasible. It also uses the fact that for every route between two nodes in different cells, the start cell has to be left and the destination cell has to be entered (Step 8). As soon as all minimum cost routes to the end nodes of edges leaving the start cell have been found, no more internal edges of the start cell have to be evaluated. As soon as all minimum cost routes to the end nodes of the edges entering the destination cell have been found, no more route edges or boundary edges have to be evaluated. Furthermore, it is relatively easy to let the algorithm select only allowed adjacent boundary edges in Step 6. Therefore, the turn restrictions between edges from and to the dummy node in the star graph do not have to be created explicitly anymore. Assuming the turn restrictions can be checked efficiently for a pair of edges, the next lemma holds.

Lemma 6.2

Algorithm T^* is a polynomial time algorithm of order $O(R + m \log m)$ that gives as output the minimum route cost between two nodes in a searchgraph that may contain stars. The number of edges is given by $m = 2\max_i |E_i| + |E_B| + 2|N_B|$, and the number of rules by $R = |\{(e_1, e_2) \in E \times E \cup E_A \times E_A \mid r(e_1, e_2) > 0\}|$.

7. Comparing stars and cliques

In this section we compare the number of evaluated edges of algorithm T^* using as input the searchgraph where $A_i = S_i$ for all cells, with an input searchgraph where $A_i = K_i$ for all cells. Note that the number of evaluated edges by algorithm T^* represents the number of iterations of the algorithm and is thus an objective measure of the run time of the algorithm. For our comparison we use four partitioned real world road networks, the networks of Sophia, Eindhoven, G3 and G2, see Flinsenberg [3]. Sophia is a city in the south of France. Its cell-partition consists of 14 cells and the entire road network contains approximately 2,000 edges. Eindhoven is a city in the south of The Netherlands. Its cell-partition consists of 9 cells and the entire road network contains approximately 15,000 edges. As an illustration, a cell-partition of Eindhoven is given in Figure 9. G3 is a road network in the west of Germany containing approximately 24,000 edges. Its cell-partition consists of 18 cells. Finally, G2 is a road network also in the west of Germany that contains G3. It consists of approximately 88,000 edges, and its cell-partition consists of 28 cells.

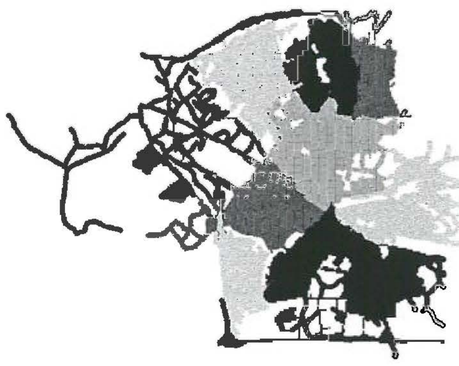

Figure 9: Cell-partition of Eindhoven.

For each road network, we have planned 400 routes, minimizing both the route length and the travel time, between randomly selected start- and end-nodes. These 400 routes are divided in four groups of 100 routes each. The first 100 start nodes and destinations (Route type Random in Table 1) have been chosen completely at random. The second group of 100 routes (Route type Commuter in Table 1) consists of start and end nodes, of which 55% is part of an urban area, and 45% has a Euclidean distance of at most 50 kilometers. This group is believed to represent a distribution of start and end nodes as they occur in real life for a commuter. The third group of 100 routes (Route type Rural in Table 1) has only start- and end nodes in rural areas, and the last group (Route type Urban in Table 1) has only start nodes and destinations in urban areas. Consider Figure 10. On the vertical axis, the gain in efficiency of the star graph approach relative to the efficiency of the clique graph approach is displayed. So a y-value of 25% means that with star graphs the algorithm evaluates only 75% of the edges that are evaluated with clique graphs. On the horizontal axis the 100 routes of a single type have been sorted on increasing y-value for each road network. Figure 10 is characteristic for all considered route types and planning criteria. All results are briefly summarized in Table 1. For each network, type of routes and planning criterion, the table presents the mean, minimum and maximum percentage of additional edges that have to be evaluated by the clique approach.

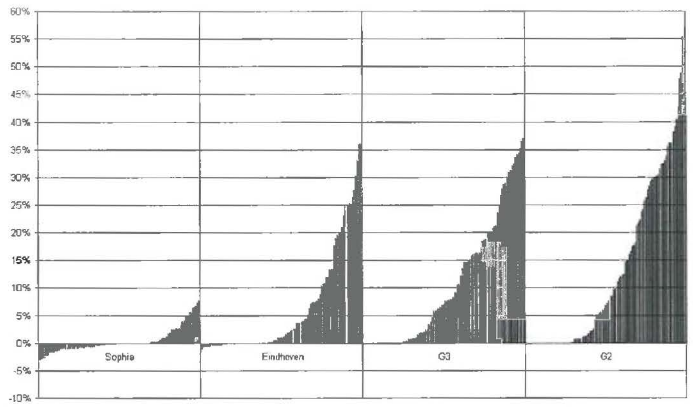

Figure 10: Percentage of additional edge evaluations of the clique approach, for fastest commuter routes.

Network	Route type	Criterion	Mean	Min.	Max.	Network	Route type	Criterion	Mean	Min.	Max.
Sophia	Random	Length	0.38	-7.50	14.70	G3	Random	Length	7.01	-0.92	53.23
		Time	0.64	-3.48	14.70			Time	7.46	-0.83	47.16
	Commuter	Length	0.21	-3.33	7.61		Commuter	Length	10.24	-0.76	35.53
		Time	0.24	-4.05	6.70			Time	10.74	-0.72	37.07
	Rural	Length	0.04	-6.67	17.00		Rural	Length	10.50	-0.48	48.80
		Time	0.29	-5.56	17.13			Time	10.91	-0.23	43.66
	Urban	Length	0.47	-3.43	7.34		Urban	Length	8.23	-0.74	40.78
		Time	0.60	-5.06	8.09			Time	8.88	-0.38	43.47
Eindhoven	Random	Length	3.80	-0.70	20.90	G2	Random	Length	12.23	-0.55	51.03
		Time	2.69	-9.05	17.35			Time	11.92	-0.35	50.23
	Commuter	Length	6.97	-1.13	36.77		Commuter	Length	13.86	-0.38	59.30
		Time	6.30	-1.60	36.10			Time	13.35	-0.27	59.89
	Rural	Length	9.60	-0.39	71.15		Rural	Length	15.84	-0.28	62.04
		Time	9.16	-5.72	71.15			Time	15.47	-0.63	60.33
	Urban	Length	4.54	-1.13	25.63		Urban	Length	11.20	-0.44	57.06
		Time	3.65	-2.55	26.84			Time	10.87	-0.43	55.11

Table 1: Percentage of additional edge evaluations of the clique approach.

From these results, it is clear that for road networks of different sizes, for different planning criteria, and different types of routes, the star approach outperforms the clique approach. The star approach requires much fewer edge-evaluations in general. For cell-partitions with cells with more boundary nodes, the star approach is most effective. This can be explained by the fact that the larger the number of boundary nodes is, the larger is the difference between the number of edges in both structures. Furthermore, the larger the network is, the more effective the star approach becomes. This is probably related to the fact that the efficiency gain is higher for long routes than for short routes, because for larger networks, the routes get longer and the number of long routes increases. When we took a closer look at the rural routes in the city of Eindhoven, it appeared that almost all start and end nodes were located on the border of the map, leading to relatively long routes for this road network. From the results of these routes, we can conclude that the efficiency gain is higher for long routes than for short routes. The larger efficiency gain for long routes can be explained by observing that for long routes, more cells have to be traversed to reach the destination cell. If no edges in the boundary graph are evaluated to find the minimum cost route, then the change from a star to a clique graph does not make any difference. This is most likely to occur for a start node and destination node that are contained in a single cell. This explains, for the networks of Eindhoven and Sophia, the larger number of routes for which there is no difference between both approaches with respect to the number of evaluated edges. Because the cell-partitions of these networks contain only a few cells, the chance that the start node and destination are contained in the same cell is relatively high.

In normal road networks, only for a few routes, the star approach leads to a small increase in the number of edge evaluations. This is caused by the difference in the evaluation of the edges of a clique and a star graph. The difference in the number of evaluated edges for a star and clique graph depends, among others, on which edges of the clique graph are evaluated and on the costs of the edges in the star graph. Compared to the gain in edge evaluations for other routes, the increase in the number of evaluated edges for these few routes is negligible. We can safely conclude that using star graphs is the better way to store the minimum path costs between every pair of boundary edges in a cell.

Conclusions

Schmid [10] and Winter [12] have shown that in real-world road networks that contain turn restrictions, optimum routes can be planned. We can speed up the route planning process by creating a cell-partition. When we create this cell-partition, we have to take the turn restrictions into account, if we want to guarantee that optimum routes can still be planned. We have shown that the minimum route costs can be stored in a so-called star graph, and that optimum routes can be planned using this structure, if they can be planned using the standard method of storing the minimum route costs. Furthermore, we developed an algorithm for planning optimum routes with a searchgraph that contains star graphs. By planning optimum routes for different road networks, different planning criteria, and different route types, we have shown that our new algorithm is generally considerably faster for real-world road networks, using our new boundary graph with star graphs than using the old boundary graph with clique graphs. In the future, we will investigate whether different cell-partitions can be created that allow an even better performance of our new planning algorithm, in combination with using a star graph to store the minimum route costs between boundary nodes of a single cell.

References

1. **E.W. Dijkstra**. A Note on Two Problems in Connexion with Graphs, Numerische Mathematik 1: 269-271, 1959.
2. **Gerhard Ertl**. Shortest path calculation in large road networks, OR Spectrum 20: 15-20, 1998.
3. **Ingrid Flinsenberg**. Graph Partitioning for Route Planning in Car Navigation Systems, to appear in: Proceedings of the 11th IAIN World Congress, Berlin, Germany, October 2003.
4. **G. Galperin**. On the Optimality of A*, Artificial Intelligence 8(1): 69-76, 1977.
5. **Monica R. Henzinger, Philip Klein, Satish Rao, Sairam Subramanian**. Faster shortest-path algorithms for planar graphs, Journal of Computer and System Sciences 55: 3-23, 1997.
6. **Sungwon Jung, Sakti Pramanik**. An Efficient Path Computation Model for Hierarchically Structured Topological Road Maps, IEEE Transactions on Knowledge and Data Engineering 14(5), September/October 2002.
7. **Kihong Kim, Seungwon Yoo, Sang K. Cha**. A partitioning scheme for hierarchical path finding robust to link cost update, in Proceedings of the 5th world congress on ITS, Seoul, Korea, October 1998.
8. **Richard J. Lipton, Robert Endre Tarjan**. A separator theorem for planar graphs, SIAM Journal of Applied Mathematics 26(2): 177-189, April 1979.
9. **Judea Pearl**. Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, Reading, Massachusetts, 1984.
10. **Wolfgang Schmid**. Berechnung Kürzester Wege in Straßenetzen mit Wegeverboten, Ph.D.thesis, Universität Stuttgart, Breitwiesenstr. 20/22, D-70565 Stuttgart, July 2000.
11. **Stefan Szeider**. Finding paths in graphs avoiding forbidden transitions, Discrete Applied Mathematics 126: 261-273, 2003.
12. **Stephan Winter**. Modeling Costs of Turns in Route Planning, GeoInformatica 6(4): 345-360, December 2002.