EFFICIENT DOUBLE ROOT OPTIMAL PATH DETERMINATION

P. H. Richter
O & S-Consultancy
10369 Berlin, Hohenschönhauser Str. 1
Germany

OVERVIEW

Today's optimal path strategies are the more competitive the lower memory space and performance time they need. That is why the concept of hierarchical maps has been introduced to delegate the main part of long distance path determination to maps with levels as high as possible (larger regions ⇔ roads of higher levels). The size of the corresponding high level road graphs remains very tractable whereas that of low level road graphs remains not.

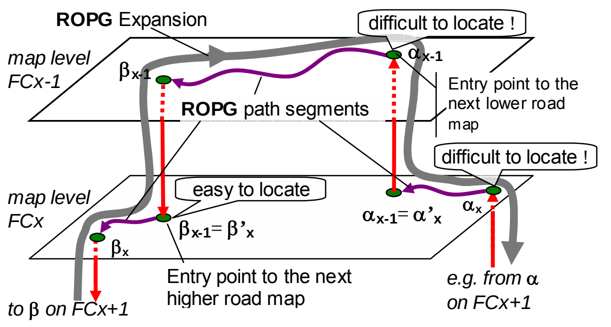


Fig. 1 Single root exploration building an ROPG

Fig. 1 depicts how the optimal path search proceeds if it adheres at an *One-root strategy* building a *Reverse Optimal Path Graph ROPG* at the target β running through the edges in their reverse direction**.** At each map level x the search is carried out for such a crossing β'_x that conveniently serves as transition crossing $\beta'_x = \beta_{x-1}$ to the next higher (lower when top-down) road map. The quality of such a one-root path search suffers regarding the top-down search (right site of Fig. 2, transition $\alpha_{x-1} \Rightarrow \alpha_x$). This results from the difficult to find the optimal connection point within the lower level map that fulfils the optimal path criterion with respect to an optimal path $\mathbf{P}_G(\alpha, \beta)$.
Therefore, this paper introduces a new algorithm here called ***A-2*** that caries out a double root strategy corresponding to Fig. 2. The provisional result is an

- partial *Optimal Path Graph* OPG $\mathfrak{T}(\alpha)$ and an
- partial *Reverse Optimal Path Graph* ROPG $\overleftarrow{\mathfrak{T}}(\beta)$ that include the wanted optimal path
- $\mathbf{P}_G(\alpha, \beta)= \mathbf{P}_G(\alpha, \lambda) \cup \mathbf{P}_G(\lambda, \beta)$ with $\mathbf{P}_G(\alpha, \lambda) \subseteq \mathfrak{T}(\alpha)$ and $\mathbf{P}_G(\lambda, \beta) \subseteq \overleftarrow{\mathfrak{T}}(\beta)$, Fig. 2.

The simultaneous development of $\mathfrak{T}(\alpha)$ and $\overleftarrow{\mathfrak{T}}(\beta)$ succeed (here simplified) if both partial sub-graphs meet in a common vertex $\lambda \in V_G$.
Notice, we have to abandon the concept *Optimal Path* *Tree* because turn restrictions may coerce optimal paths having cycles.

We emphasize that the *ROPG* $\overleftarrow{\mathfrak{T}}(\beta)$ (corresponding to the All-to-One problem) makes available rerouting information by a successor list whereas the *OPG* $\mathfrak{T}(\alpha)$ (corresponding to the One-to-All problem) only generates a predecessor list not suitable for re-routing information.

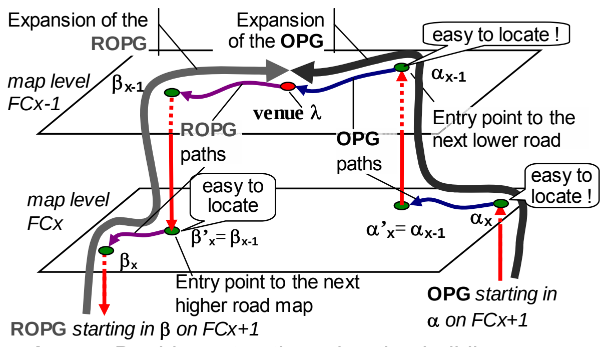


Fig. 2 Double root path exploration building an *OPG* and **an** ROPG

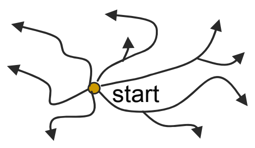
1. INTRODUCTION

We start from a

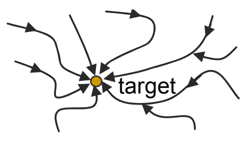
- road net graphs **G=** $[V_G, E_G]$
  with traffic points (vertices) V_G and the roads (edges) $E_G \subseteq V_G^2$ and a
- road cost function $\ell: E_G \rightarrow \boldsymbol{R}_+$
  denoting the effort $\ell(r)$ traversing road r (length, time).

OPG $\mathfrak{T}(\alpha)$ and *ROPG* $\overleftarrow{\mathfrak{T}}(\beta)$ can be seen as a bunch of overlapping optimal paths from all $x \in V_G$ to the target β or from start α to all $x \in V_G$:

$$\mathfrak{T}(\alpha) = \bigcup_{x \in V_G} \{\mathbf{P}_G(\alpha, x)\}.$$



$$\overleftarrow{\mathfrak{T}}(\beta) = \bigcup_{x \in V_G} \{\mathbf{P}_G(x, \beta)\},$$



In the following we assign some qualities to the graph's vertices and edges depending on the proceeding of the algorithm's path exploration. Differently from the pure coding that uses mostly bit signatures, we introduce here set notations to assign qualities to vertices and edges. If an element x has quality Q we write $x \in Q$, else $x \notin Q$. To assign a quality Q to an element x we use the short form $Q \cup= \{x\}$ instead $Q = Q \cup \{x\}$.

- $\pi: E_G \rightarrow \boldsymbol{R}_+$ is the ***edge potential*** (cumulated edge lengths). All those $e = (x, y)$ that have been connected either to the current partial $\mathfrak{T}(\alpha)$ or $\overline{\mathfrak{T}}(\beta)$ are assigned the cost $\pi(r)$ of the path in $\mathfrak{T}(\alpha)$ from start α or the cost of the path in $\overline{\mathfrak{T}}(\beta)$ to β. Initially, $\forall e \in E_G: [\pi(e) = \infty]$.
- $\Omega \subseteq E_G \cup V_G$ is the set of vertices and edges that are connected to the current *OPG* $\mathfrak{T}(\alpha)$. It is to grasp from the context, whether with $x \in \Omega$ x is an edge or a vertex.
- $\overline{\Omega} \subseteq E_G \cup V_G$ is the set of vertices and edges that are connected to the current *ROPG* $\overline{\mathfrak{T}}(\beta)$. It is to grasp from the context, whether with $x \in \overline{\Omega}$ is an edge or a vertex.
- $\Pi \subseteq E_G \cup V_G$ is the set of vertices and edges that have been labelled as "permanent". An edge $e \in \Pi$ cannot be improved further as to $\pi(e)$. If an edge $e \in E_G$ is "permanent", i.e. $e \in \Pi$, then its incident vertices are alo labelled "permanent".
  $\Pi \cup= \{e\}$ is set when edge $e = (x,y)$ is fetched from the queue Q as a minimum potential edge. It follows $\Pi \cup= \{e, x, y\}$.
- $\sigma: E_G \rightarrow E_G$ is an ***edge sequence function*** applied to edges as follows:
  - $e \in E(\overline{\mathfrak{T}}(\beta)) \Rightarrow \sigma(e) \in E(\overline{\mathfrak{T}}(\beta))$ is the successor edge of e for the current optimal path $e \rightarrow \sigma(e) \rightarrow \sigma(\sigma(e))$ …. (z, β) towards target β.
  - $e \in E(\mathfrak{T}(\alpha)) \Rightarrow \sigma(e)$ is the predecessor edge of e for the current optimal path $e \rightarrow \sigma(e) \rightarrow \sigma(\sigma(e))$ …. (α, y) backwards directed to the start α.

  It holds for some $e \in E_G$:

  $e \in \Omega \Rightarrow$ $e \in E(\mathfrak{T}(\alpha))$,
  $\sigma(e)$ is the predecessor edge of e.

  $e \in \overline{\Omega} \Rightarrow$ $e \in E(\overline{\mathfrak{T}}(\beta))$,
  $\sigma(e)$ is the successor edge of e.

  $e \notin \Omega \wedge e \notin \overline{\Omega} \Rightarrow$ $\sigma(e)$ is not defined;
  $\pi(e) = \infty$ (infinite). e hasn't been reached by by the current path development.

2. PROBLEM FORMULATION

We start from

- a digital ***road map*** given as connected, directed, and finite graph $\mathbf{G} = [V_G, E_G]$.
- a fixed ***start point*** $\alpha \in V_G$,
- a fixed ***target*** $\beta \in V_G$,
- the ***roads' lengths*** *(or time)*. $\ell :: E_G \rightarrow \boldsymbol{R}_+$
- ***turn restrictions*** $\overline{r}: V_G \rightarrow \mathfrak{P}(E_G^2)$ [1]
  to interpret as follows: "If a driver comes via road r1 to crossing x then driving away via road r2 is not allowed!" I.e. the tupelo (r1, r2) is to assign to crossing x as restriction, written as $(r1, r2) \in \overline{r}(x)$,

Let $\mathbf{H} \subseteq \mathbf{G}$ be an arbitrary sub-graph in $\mathbf{G}$, i.e. $V_H \subseteq V_G$ and $E_H \subseteq E_G \cap V_H^2$.

We denote with $C(\mathbf{H}) = \sum_{r \in E_H} \ell(r)$ = cost of sub-graph $\mathbf{H}$.

PROBLEM:

We look for a path $\mathbf{P}_G(\alpha, \beta)$ from α to β observing turn restrictions such that $C(\mathbf{P}_G(\alpha, \beta))$ is a cost minimal path.

2-ROOT OPTIMAL PATH PROBLEM WITH TURN RESTRICTIONS:

We look for sub-graph $\mathbf{P}_G(\alpha, \beta) \subseteq \mathbf{G}$ observing turn restrictions $\overline{r}$ with cost

$$C(\mathbf{P}_G(\alpha, \beta)) = \min_{\forall b \in V_G} .\{C(\mathbf{P}_G(\alpha, b) + C(\mathbf{P}_G(b, \beta))\}.$$

The formula was given in the form above to enable the view on the algorithm's parallel solution strategy such that $\mathbf{P}_G(\alpha, b) \subseteq \mathfrak{T}(\alpha)$ and $\mathbf{P}_G(b, \beta) \subseteq \overline{\mathfrak{T}}(\beta)$ might be built simultaneously developing two small path waves meeting about in the middle of the road sequence between α and β and having much less search effort than a large search wave spreading around α till β is reached, see Fig. 5. Algorithm ***A-2*** designed for this strategy coevally develops very efficient an *OPG* $\mathfrak{T}(\alpha)$ and an *ROPG* $\overline{\mathfrak{T}}(\beta)$ using only one priority queue till both par-tial sub-graphs $\mathfrak{T}(\alpha)$ and $\overline{\mathfrak{T}}(\beta)$ encounter each other in a point here denoted as $b \in V_G$ (simplified).

[1] $\mathfrak{P}(M)$ denotes the „power set" of set M

3. SOLUTION ALGORITHM *A-2*

3.1 Program Flowchart

1 Initialization

```
If( $\alpha == \beta$) return 0; // trivial case.
$\forall e \in E_G$: { $\sigma(e)=e$; $\pi(e)=\infty$};
$\Omega \cup= \{\alpha\}$; $\overline{\Omega} \cup= \{\beta\}$; $\Pi \cup= \{\alpha, \beta\}$;
**Q**= 0; $L_1$= -1; $L_2$= -1; $c_{min}= \infty$;
$\forall e= (\alpha, x) \in E_G$: {
    $\pi(e)=\ell(e)$; **Q** $\cup= \{e\}$; $\Omega \cup= \{e\}$;
};
$\forall e= (x, \beta) \in E_G$: {
    $\pi(e)=\ell(e)$; **Q** $\cup= \{e\}$; $\overline{\Omega} \cup= \{e\}$;
};
```

2 Priority queue empty?

```
If(**Q**==0) **goto 9**;
```

3 Process the minimum edge L.

```
L= (x, y) $\in$ **Q** mit $\pi(L)= \min_{L' \in Q} \{\pi(L')\}$ ;
**Q** \ = {L}; $\Pi \cup= \{L\}$;
If (L $\in \Omega$) **{**
   a= x; b= y; $L_{OPG}$= L; r = $\beta$; opg= 1;
**}** else {
   a= y; b= x; $L_{ROPG}$= L; r = $\alpha$; opg= 0
**};**
```

4 Does L connect OPG and ROPG?

```
If($c_{min} < \infty$)
   If($c_{min} < \pi(L_{OPG}) + \pi(L_{ROPG})$)
     goto 20;
If( (opg==1 $\wedge$ b $\notin \overline{\Omega}$) $\vee$
    (opg==0 $\wedge$ b $\notin \Omega$)) **goto 7**
```

5 Process the current connection via L.

```
ec= -1;
If ( b== r) { ec= L; cc= $\pi(L)$ }
else {
   If( opg==1) {
      $\forall$ e= (b, j) $\in E_G$ || {b}: {
         If (j== a) continue;
         If (e $\notin \overline{\Omega}$) continue;
         If ((L, e) $\in \overline{r}$ (b) continue;
         If(ec== -1) {ec= e; cc= $\pi(L)+ \pi(e)$ }
         else  If ($\pi(L)+\pi(e) < \pi(ec)$) {
                ec= e; cc= $\pi(L)+\pi(e)$ }
   } else {
      $\forall$ e= (j, b) $\in E_G^{-1}$ || {b}: {
         If (j== a) continue;
         If (e $\notin \Omega$) continue;
         If ((e, L) $\in \overline{r}$ (b) continue;
         If(ec== -1) {ec= e; cc= $\pi(L)+ \pi(e)$ }
         else  If ($\pi(L)+\pi(e) < \pi(ec)$) {
                ec= e; cc= $\pi(L)+\pi(e)$ }
   }
}
```

6 Is the new connection currently the best?

```
If(ec $\geq$ 0) {
     If(cc < $c_{min}$) {
          $L_1$= L; $L_2$= ec; $c_{min}$= cc;
     }
}
```

7 Mark the incident vertices of L.

```
$\Pi \cup= \{a, b\}$;
If (opg) $\Omega \cup= \{a, b\}$; else $\overline{\Omega} = \{a, b\}$;
```

8 Process the incident edges of L.

```
If( opg==1) {
  $\forall$ e= (b, x ) $\in E_G$ || {b}: {
        If (x== a) continue;
        If ((x $\in \Omega$) $\wedge$ (x $\in \Pi$) ) continue;
        If( e $\in \overline{\Omega}$ ) continue;
        If((L, e) $\in \overline{r}$ (b) continue;
        $\Omega \cup= \{x, b\}$;
        y= $\pi(L) + \ell(e)$;
        If( y < $\pi(e)$ {
             $\sigma(e)$= L: $\pi(e)$= y; $\Omega \cup= \{e\}$;
             **Q** $\cup= \{e\}$;
        }
  }
} else {
  $\forall$ e= (x, b) $\in E_G^{-1}$ || {b}: {
        If (x== a) continue;
        If ((x $\in \overline{\Omega}$ ) $\wedge$ (x $\in \Pi$) ) continue;
        If( e $\in \Omega$) continue;
        If((e, L) $\in \overline{r}$ (b) continue;
        $\overline{\Omega} \cup= \{x, b\}$;
        y= $\pi(L) + \ell(e)$;
        If( y < $\pi(e)$ {
             $\sigma(e)$= L: $\pi(e)$= y; $\overline{\Omega} \cup= \{e\}$;
             **Q** $\cup= \{e\}$;
        }
  }
}
**Goto 2**;
```

9 Final treatment

```
If($c_{min}== \infty$) ABEND("No solution");
$\Rightarrow$ The edge sequences
     $L_1$, $\sigma(L_1)$, $\sigma(\sigma(L_1))$, $\sigma(\sigma(\sigma(L_1)))$…
     and
     $L_2$, $\sigma(L_2)$, $\sigma(\sigma(L_2))$, $\sigma(\sigma(\sigma(L_2)))$…
     form the optimal path $\mathbf{P}_G(\alpha, \beta)$ with cost
     $\pi(L_1)+\pi(L_2)$. $\mathbf{P}_G(\alpha, \beta)$ ensures turn
     restrictions.
```

Fig. 3 Algorithm *A-2* simultaneously building an *OPG* $\mathfrak{T}(\alpha)$ and *ROPG* $\overline{\mathfrak{T}}(\beta)$ containing an optimal path $P_G(\alpha, \beta)$

Deailled Description of Fig. 3 above

1	Initialization
	▪ Return if target= start point. ▪ Assign all edges themselves as successor (ROPG) (predecessor (OPG) respectively de-pending on the development). ▪ Assign all edges an unrealistic high potential. ▪ Establish the priority queue Q, e.g. as binary heap. ▪ Set the border edges between OPG and ROPG L_1 and L_2 as 'unknown' (-1). ▪ Set the cost c_{min} of the future optimal path $\mathbf{P}_G(\alpha, \beta)$ to an unrealistic high value. ▪ For all edges leaving start α: - Set their potential to their length. - Put them into Q. - Assign them as OPG members. ▪ For all edges arriving target β: - Set their potential to their length. - Put them into Q. - Assign them as ROPG members.
2	Priority queue empty?
	▪ If no edge is within Q proceed with the final treatment
3	Process the minimum edge L.
	▪ Fetch the minimum queue edge L= (x,y). ▪ Mark L as 'permanent' (not improvable). ▪ If L belongs to the current OPG / ROPG set a, b, L_{OPG}, L_{ROPG}, r, and opg correspon-dingly.
4	Does L connect the current OPG and ROPG?
	▪ Prove whether a further enlargement of *OPG* and *ROPG* is necessary. So far at least one solution has been found ($c_{min} < \infty$) a connection with better cost than c_{min} cannot be found with the further algorithmic proceeding if the sum $\pi(L_{OPG}) + \pi(L_{ROPG})$) of the current connection via L_{OPG} and L_{ROPG} is greater than c_{min}. Thus, **goto 20**; ▪ A connection between *OPG* and *ROPG* occurs if vertex b belongs to the path graph different from that L belongs to. Otherwise, a normal adding of the edge incident to L happens lea-ding to the enlargement of the current optimal path graph L belongs to. In this case **goto 7**:
5	Process the current connection via L.
	▪ If L reaches the target (OPG) or start (ROPG) the connection edge ec is L itself . ▪ Otherwise, the edges e leaving b (OPG) or arriving b (ROPG) are only genuine connection edges (ec) if they are - not a reverse parallel edge - not belonging to the other path graph - ensuring turn restrictions whereby the minimum potential edge e is stored into ec with respect to current total path cost cc= $\pi(L)+ \pi(e)$.
6	Is the new connection better than the old ones?
	If the connection via L and ec is better than previous connections stored so far ($cc < c_{min}$) then store L_1= L; L_2= ec as the current best connection.
7	Assign the new property to the vertices of L
	▪ Mark a and b as 'permanent' and assign them the membership of L.
8	Process the incident vertices of L
	▪ If $L \in \Omega$: For all outgoing edges e= (b,x) of L leaving b - not going back to a, - not in Ω and not 'permanent', - not assigned to the other optimal path graph, - ensuring turn restrictions: assign x and b to Ω and prove whether the potential of e can be reduced by a correction such that $\pi(L)+ \ell(e) < \pi(e)$. If so set e its new predecessor, potential, and membership and put e with its new potential $\pi(e)= \pi(L)+ \ell(e)$ into the queue Q. **Goto 2** ▪ If $L \in \overline{\Omega}$: For all incoming edges e= (x, b) of L arriving b - not going back to vertex a, - not in $\overline{\Omega}$ and not 'permanent' - not assigned to the other optimal path graph, - ensuring turn restrictions assign x and b to $\overline{\Omega}$ and prove whether the potential of e can be reduced by a correction such that $\pi(L)+ \ell(e) < \pi(e)$. If so set e its new predecessor, potential, and membership and put e with its new potential $\pi(e)= \pi(L)+ \ell(e)$ into the queue Q. **Goto 2**
9	Final treatment
	▪ If c_{min} has its initial value, there is no path from α to β (graph not connected or insane turn restrictions). ▪ Dependent on the membership of L_1, σ is a successor list ($L_1 \in \overline{\Omega}$) or predecessor list ($L_1 \in \Omega$). a) $L_1 \rightarrow \sigma(L_1) \rightarrow \sigma(\sigma(L_1)) \rightarrow \sigma(\sigma(\sigma(L_1))) \rightarrow \rightarrow$ leads in edge direction towards target β and $L_2 \rightarrow \sigma(L_2) \rightarrow \sigma(\sigma(L_2)) \rightarrow \sigma(\sigma(\sigma(L_2))) \rightarrow \rightarrow$ leads backwards in reverse edge direction to the start α, b) $L_2 \rightarrow \sigma(L_2) \rightarrow \sigma(\sigma(L_2)) \rightarrow \sigma(\sigma(\sigma(L_2))) \rightarrow \rightarrow$ leads in edge direction towards target β and $L_1 \rightarrow \sigma(L_1) \rightarrow \sigma(\sigma(L_1)) \rightarrow \sigma(\sigma(\sigma(L_1))) \rightarrow \rightarrow$ leads backwards in reverse edge direction to the start α.

3.2 Exactness and Complexity of *A-2*

A-2 is an exact algorithm delivering a solution with polynomial time effort $\boldsymbol{O}(m^2)$.

Proof of exactness

Algorithm ***A-2*** is an edge *Label Setting* algorithm (LS). We start from the fact that LS – algorithms fetch only edges (here not vertices) from the queue Q that ensure a clear-cut sequence of predecessors (OPG) or successors (ROPG) constituting a not improvable optimal path! Every time a minimum potential edge L is π fetched from Q and has one of its two vertices (here called b) with $b \in \Omega \cap \overline{\Omega}$ there arises a preliminary (not necessarily optimal) path P= $\mathbf{P}_G(\alpha, b) \cup \mathbf{P}_G(b, \beta)$. It can only be said that at least one of the two partial paths is optimal. However, P is a solution candidate as long as its cost $C(P)= C(\mathbf{P}_G(\alpha, b)) + C(\mathbf{P}_G(b, \beta)) \le \pi(L_1) + \pi(L_2)$.

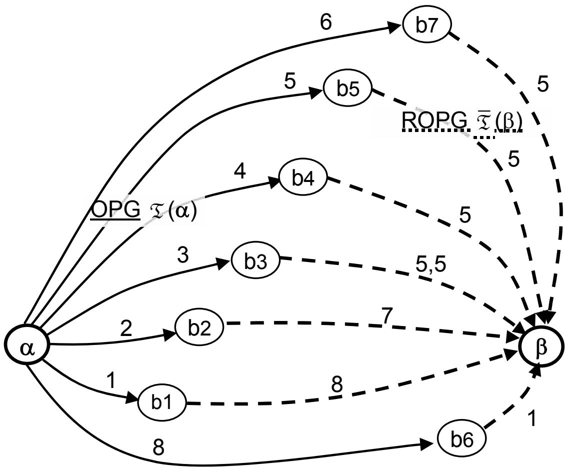
We assume, that path P under consideration is not the total optimal path $\mathbf{P}_G(\alpha, \beta)$. It follows that there is another L* fetched later from the queue Q with an incident edge ec* yielding

a) $\pi(L) \le \pi(L^*)$, $\Rightarrow$ L* after L from Q
b) $\pi(L^*) + \pi(ec^*) < C(P)= \pi(L) + \pi(ec)$,
$\Rightarrow$ better link via L* assumed
c) $\pi(L^*) \le \pi(ec^*)$, $\Rightarrow$ otherwise ec* as L* selected
d) $\pi(L^*) + \pi(ec^*) \le \pi(L_{OPG}) + \pi(L_{ROPG})$,
$\Rightarrow$ otherwise no improvement chance
e) $L^*= L_{OPG}$ or $L^*= L_{ROPG}$.
$\Rightarrow$ L* is current L_{OPG} or L_{ROPG}.

Since each edge (connected graphs!) can be reached by the potential underbidding control, those edges L are processed that have the current least potential. If such an L encounters an edge ec already belonging to the other optimal path graph, L and ec is a solution candidate, It might be replaced to a better one being found later. However, if the last stored L_{OPG} and L_{ROPG} have cost $\pi(L_{OPG}) + \pi(L_{ROPG}) > c_{min}$ the chance is over that a better candidate can be found because this c_{min} will never be decreased further. On the other hand, each tupelo (L*, ec*) that might exist further as solution candidate sufficing

$$c_{min} < \pi(L^*)+ \pi(ec^*) < \pi(L_{OPG})+ \pi(L_{ROPG})$$

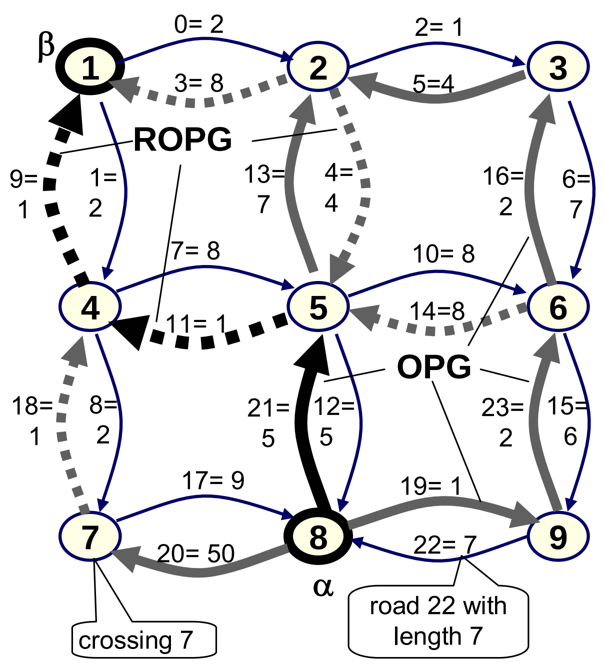
is assured to be found due to the exact underbidding control of the LS strategy we have implemented for edge an queuing startgy corresponding to block 3 to block 8, Fig. 3, ■



1.	L= (α, b1), ec= (b1, β), C(P)= 9. <u>c_{min}= 9</u>, L_1= (α, b1), L_2= (b1, β).	L_{OPG}= (α, b1), L_{ROPG}= $\perp$
2.	L= (b6, β), ec= (α, b6), C(P)= 9. L_1, L_2 not changed.	L_{OPG}= (α, b1), L_{ROPG}= (b6, β)
3.	L= (α, b2), ec= (b2, β), C(P)= 9. L_1, L_2 not changed.	L_{OPG}= (α, b2), L_{ROPG}= (b6, β)
4.	L= (α, b3), ec= (b3, β), C(P)= 8,5; <u>c_{min}= 8,5</u>, L_1= (α, b3), L_2= (b3, β).	L_{OPG}= (α, b3), L_{ROPG}= (b6, β)
5.	L= (α, b4), ec= (b4, β), C(P)= 9; L_1, L_2 not changed.	L_{OPG}= (α, b4), L_{ROPG}= (b6, β)
6.	L= (α, b5), ec= (b5, β), C(P)= 10; L_1, L_2 not changed.	L_{OPG}= (α, b5), L_{ROPG}= (b6, β)
7.	L= (b4, β), ec= (α, b4), C(P)= 9; L_1, L_2 not changed.	L_{OPG}= (α, b5), L_{ROPG}= (b4, β)
	Now (Fig. 3, block 4) the end condition is given: $\pi(L_{OPG})+\pi(L_{ROPG})$= 5 + 5= 10 > c_{min} $\Rightarrow$ End.	

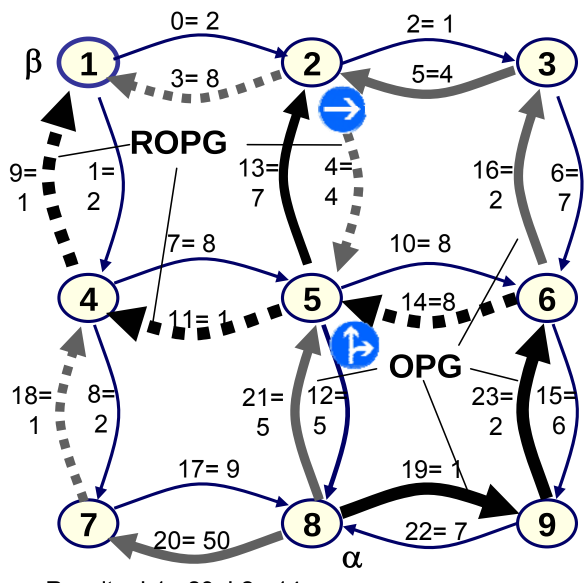
Fig. 4 Elucidating the exactness proof
Corresponding to Fig. 3, the algorithmic extension steps 1. .. 7 above explain the proceeding of ***A-2***. Simplifying matter, we consider a graph **G** whose edges are immediately assigned by block 1 of Fig. 3,

No traffic restrictions:
The shortest path from 8 to 1 has <u>length 7</u>.



Result: L1= 21, L2= 11, E($\mathbf{P}_G$ (a, b))= {21, 11, 9} with <u>cost 7</u>.

With traffic restrictions:
The shortest path from 8 to 1 found by ***A-2*** has <u>length 25</u>



Result: L1= 23, L2= 14, E($\mathbf{P}_G$(a, b))= {19, 23,14, 11, 9} with cost 13

Fig. 5 Example explaining *A-2*

<u>Proof Complexity $\boldsymbol{O}(m^2)$</u>

First, we assume the <u>average case</u> where the roads are uniformly distributed over the digital map **G**. Thus, one spreading path wave launched by a single rooted LS-algorithm in edge-queuing strategy takes a time effort $\boldsymbol{O}(m^2)= \boldsymbol{O}(r^2 \pi)$. Double root LS-algorithm ***A-2*** takes the effort $\boldsymbol{O}((r/2)^2 \pi+ (r/2)^2 \pi)= \boldsymbol{O}(½ r^2 \pi)$.
Fig. 6 explains this fact.

However, the worst case happens, if ,e.g., the start edges are very high evaluated and the remaining ones not. For this case ROPG $\overline{\mathfrak{T}}(\beta)$ is spreading nearly over the whole map resulting to $\boldsymbol{O}(m^2)$. ■

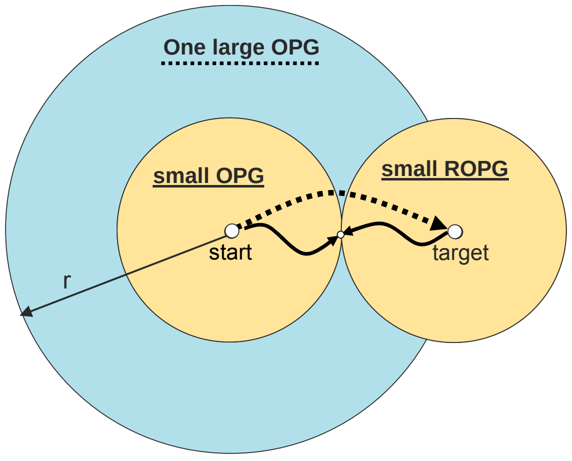


Fig. 6 Explaining the time effort of *A-2*

We assign all edges to generations g1, g2, …. as follows.
It follows that each generation has at least one edge whose potential cannot be improved and won't come again into Q. That means, that the theoretic maximum number of edges each generation steadily decreases at least by 1. Since m+(m-1)+(m-2)+ .. + 3+2+1= $\boldsymbol{O}(m^2)$, m=|E|.
Thus the worst case complexity is $\boldsymbol{O}(m^2)$. ■

4. PERFORMANCE ANALYSIS

4.1 Test Graphs

We used a random grid graph **G**= [V_G, E_G] , n= |V_G|, m= |E_G|, n ≤ 200.000 (vertices) and m = 796210 (edges), with road length function $\ell: E_G \rightarrow \boldsymbol{R}_+$ and $10 \leq \ell(r) \leq 40$ for all $r \in E_G$, Fig. 8. The graphs are directed. Each inner vertex $v \in V_G$ has four incoming edges and four outgoing edges. The graphs are compact and shaped as squares with rows and columns corresponding to about $\sqrt{n}$. The correctness of the results has been proved for smaller graphs as they and their optimal paths with turn restrictions were displayed on the screen.

Road Map G V(G)= 200000, E(G)= 798210 (500 columns) Edge Cost in [10; 14]			
start	**target**	**remote-ness**	**length**
100248	100252	4	45
100225	100275	50	566
100200	100300	100	1150
100175	100325	150	1724
100150	100350	200	2299
100125	100375	250	2887
100100	100400	300	3447
100075	100425	350	3998
100050	100450	400	4564
100025	100475	450	5136
100001	100500	499	5691

Fig. 7 Start conditions testing *A-2*

4.2 Comparative Algorithms Tackling Turn Restrictions

We consider only those algorithms that explore the paths using an edge queuing strategy (suffix _E). Not vertices but edges are put into the queue Q. This strategy enables algorithms to implement turn restrictions as described in . ***LS_E_OPG*** and ***LC_E_OPG*** return an edge predecessor list not (as ***A-2***) a successor list. It emerges an Optimal Path Graph *OPG* $\mathfrak{T}(\alpha)$ that can be denoted only then as *Optimal Path Tree* if turn restrictions are not observed.

– Algorithm ***LS_E_OPG***

  is an edge-queuing ***L***abel-***S***etting algorithm, i.e. the path exploration finishes if the target β has been reached. Queue Q is a priority queue organized as a binary heap to efficiently find the minimum potential edge. The path exploration begins at the start point α and ends if a minimum potential edge e= (z, β) has been fetched from the queue Q (β reached). An *OPG* emerges $\mathfrak{T}(\alpha)$. The run time decreases (increases) the nearer (more remote (edge number)) start and target are situated. It follows, that there is an essential performance difference dependent on the remoteness of α and β. Fig. 8 shows the great difference between *LS_E_OPG*-max. (great many edges between α and β) and *LS_E_OPG*-min (α and β within the near vicinity) .

  Worst case time effort: ***O***(m log m)

– Algorithm ***LC_E_OPG***

  is an edge-queuing ***L***abel-Correcting algorithm working with a queue Q being organized as a FIFO queue (first in – first out). The path exploration begins at the start point α and ends if no edges' potential can be improved. It emerges an *OPG* $\mathfrak{T}(\alpha)$. that can finish if the target has been reached, *LC_E_OPG* needs more time effort even if α and β are situated quite near together. Nevertheless, LC-E outperforms *LS_E_OPG*.

  Worst case time effort: ***O***(m^2)

– Algorithm ***A-1***

  Like *LC_E_OPG*, ***A-1*** [9] is an edge-queuing ***L***abel-Correcting algorithm working with a queue Q being organized as a FIFO queue. The path exploration begins at the target $\beta$and proceeds backwards to the start using the edges' reverse direction. The performance is comparable with that of *LC_E_OPG* but it emerges an *ROPG* $\overline{\mathfrak{T}}(\beta)$. Thus, the very attractive quality "ubiquitous path information" provided by σ is available: Each edge $e \in E_G$ has been assigned the successor edge towards the target (via an optimal path using the following successor edges), i.e. a driver can deliberately make detours without a new calculation of the optimal path (*ROPG*). He remains always informed through the edges best successor assigned to by σ.

  Worst case time effort: ***O***(m^2)

4.3 Evaluation

Fig. 7 shows the input data. The test concept should reveal the influence of the remoteness start $\Leftrightarrow$ target on the run time. The number of streets between start and target (both initially set into the centre of **G**) increases by 50 (see "remoteness"). Referring to Fig. 8, the behaviour of the remoteness on ***A-2*** is quite similar to that of single root LS-algorithm ***LS_E_OPG*** (shape of the curve), but the time difference is considerable different The claim ***A-2*** needs the half effort of ***LS_E_OPG*** is fulfilled for the medium remoteness range (the initialization effort must be about the same for both algorithm). The Label Correcting (LC) single root algorithm ***LC_E_OPG*** as well as ***LC_E_ROPG*** are really unimpressed be the varying remoteness. No wonder, both algorithms have to scan all edges (independent of the start-target-remoteness) as long as all edges have a not-improvable potential The curves are even slightly decreasing with increasing remoteness. This seems being based on the less effort for them building up the full path exploration wave from the graph's brink (start and target very fare remote): the exploration goes not into all directions but preferably into the half of all directions. The double-root advantage of ***A-2*** is the more convincing the closer start and target are situated together.

Algorithm ***A-2*** fulfils the following requirements:

1. ***A-2*** as an hybrid algorithm (OPG & ROPG) is the fastest algorithm in the range of medium remoteness. This is the range where most navigation applications operate!

   LC-algorithm ***A-1*** [9] outperforms ***A-2*** for the fare remoteness range. For very strong real time demands it is well advised to combine the digital maps used with a criterion that applications use to decide whether ***A-1*** or ***A-2*** is to execute.
2. Insertion and deletion of traffic regulations cannot be simpler. Traffic regulations have to observe dynamic storage allocation for real-time changes. We refer to [9] where the matter is sufficiently described.

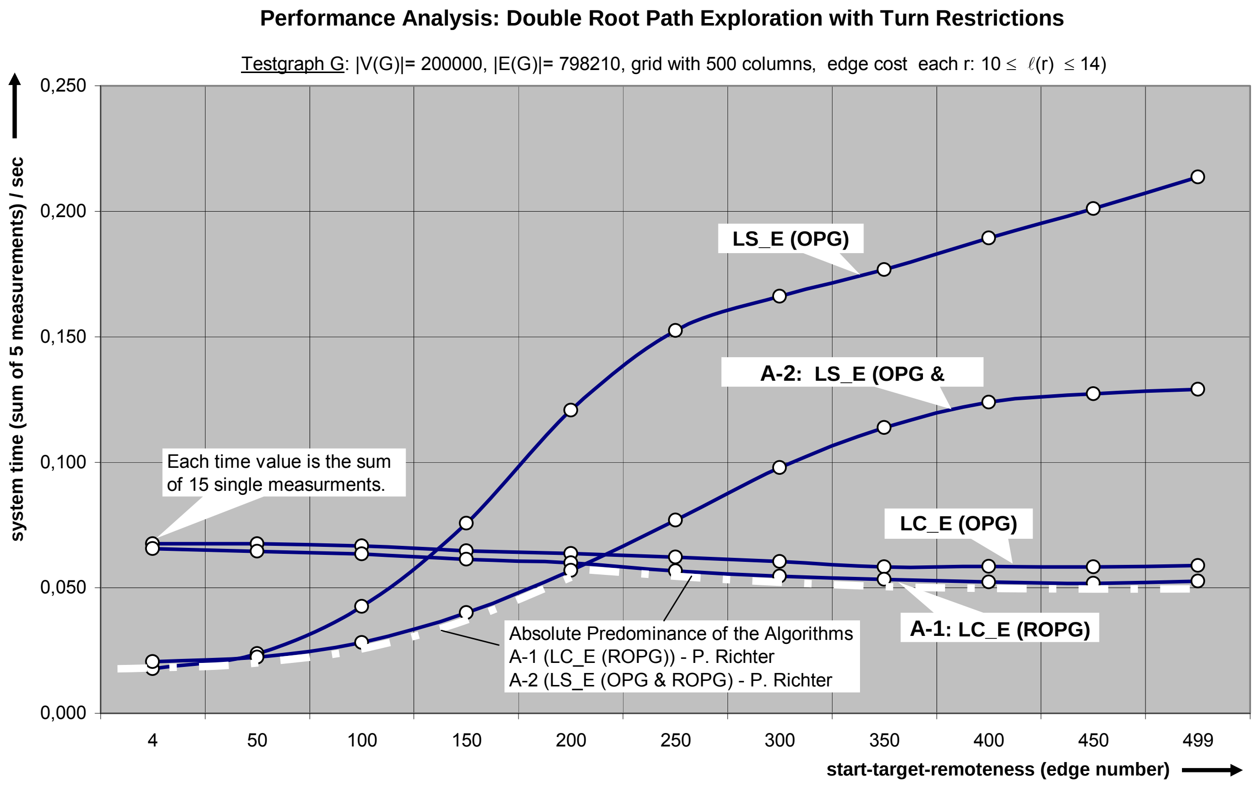


Fig. 8 Run time effort of algorithm *A-2* compared to competitors

5. SUMMARY

A-2 outclasses comparative algorithms under consideration observing turn restrictions.

The proposed algorithm ***A-2*** outperforms LC_E_OPG, LC_E_ROPG, LS_E_OPG and ***A-1*** [9] with respect to the run time effort in the range of medium start-target remoteness.
The combination of ***A-2*** with ***A-1*** is highly attractive for strong real time applications.

The simplicity to implement turn restrictions (adopted from [9]) is an very attractive feature offered for navigation applications that require a steady dynamic turn restriction change.

6. REFERENCES

[1] Bokhari, S.H., "A shortest tree algorithm for optimal assignments across space and time in a distributed processor system". IEEE Transactions on Software Engineering, 1981. SE-7: pp. 583-589.

[2] Deo N., Pang C.: Shortest Path Algorithms: Taxonomy and Annotations, NETWORKS, Vol. 14 , (1984), 275-323

[3] Dial R.: Algorithm 360 Shortest Paths Forest with topological Ordering, Communications of the ACM, 12, (1969) 632-633

[4] Dijkstra E.W.: A Note on two Problems in Connection with Graphs, Numerische Mathematik 1, (1959) 269-271

[5] Gilsinn I., Witzgall, C.: A Performance Comparison of Labeling Algorithms for Calculating Shortest Paths Trees, NBS Technical Note 772, US. Department of Commerce, (1973)

[6] Glover F., Glover R., Klingman D.: Computational Study of an Improved Shortest Path Algorithm, NETWORKS 14, (1984) 25-36

[7] Pape, V.: “Algorithm 562 Shortest Path Length”, ACM Transactions on Mathematical Software, Vol. 6, No. 3, Sep 1980, 450- 453

[8] Richter P.: “Efficient Shortest Path Algorithms for Road Traffic Optimization” ,
27th International *Symposium on Advanced Transportation Applications (ISATA): Dedicated Conference on Advanced Transport Telematics,* ISBN 0947719652, *Aachen*, 31.10.-4.11.(1994), 79-86

[9] Richter P.: “Optimal Path Determination Observing Turn Restriction”, CEAS 2007, Berlin, Ident Nr. 488; submitted for the journal “AeroSpace Science & Technology”

[10] Shier, Witzgall: Properties of Labeling Methods for Determining Shortest Path Trees, J. Res. Natl. Bur. Stand., (1986) 323-333