Department of Economics and Management
University of Brescia
Italy

WORKING PAPER

Heuristic Path Generation for the Proactive Route Guidance Approach

Enrico Angelelli
Valentina Morandi
Grazia Speranza

WPDEM 3/2016

Via S. Faustino 74/b, 25122 Brescia – Italy
Tel 00390302988742 Fax 00390302988703
email: paqricerca.dem@unibs.it

Heuristic path generation for the proactive route guidance approach

Enrico Angelelli (1) *Valentina Morandi* (2) *Grazia Speranza* (3)

(1) *Department of Economics and management, University of Brescia, Italy, enrico.angelelli@unibs.it*

(2) *Department of Economics and management, University of Brescia, Italy, valentina.morandi1@unibs.it*

(3) *Department of Economics and management, University of Brescia, Italy, grazia.speranza@unibs.it*

Abstract

The benefits in reducing traffic congestion of system optimum with respect to user equilibrium traffic assignments are well known. Recently a linear programming based approach was proposed that aims at achieving a compromise between the system perspective, namely eliminating congestion, and the user perspective, that is minimizing individual travel times. The approach, called proactive route guidance, assigns to users paths that increase the travel times by at most a given percentage, called maximum allowed travel inconvenience. The approach requires the enumeration of all feasible paths that may be memory and time consuming, especially when large networks and/or high values of the maximum allowed travel inconvenience are considered. In this paper a heuristic is presented to generate a subset of all feasible paths that is based on the iterative search of improving paths. Computational experiments show that the number of paths generated by the heuristic is smaller with respect to the complete set by one or two orders of magnitude on small instances and by higher orders of magnitude when the size of the instances increases. On instances with 150 nodes, where the complete enumeration takes an acceptable computational time, the results show that the quality of the heuristic solutions is very close to that of the optimal ones.

Keywords: Traffic congestion; Proactive route guidance; Heuristic path generation.

1 Introduction

Traffic is one of the most pressing problems in modern cities. The benefits in reducing traffic congestion of system optimum with respect to user equilibrium traffic assignments are well known

(see, for example, Mahmassani and Peeta [1993] and Roughgarden and Tardos [2002]). While system optimum approaches are used to coordinate traffic and possibly eliminate congestion when automated guided vehicles are involved (see Kaspi and Tanchoco [1990] and Bartlett et al. [2014]), the application of a system optimum approach to road networks has been considered to be not realistic due to fact that users take their own decisions. Nowadays connected vehicles and technological advances, including driverless vehicles, make the possibility of coordinating traffic more realistic. However, a still relevant drawback of a system optimum approach is that it does not consider fairness towards users.

The idea of integrating a system optimum approach with user fairness considerations has been studied in different papers. User fairness was first considered in Jahn et al. [2005], where the arc travel times were fixed to the travel times experienced under user equilibrium assignment and only the paths with duration smaller than the least duration obtained under user equilibrium were considered as feasible paths. A non linear optimization model was presented, a column generation solution method proposed and computational results were carried out on real road networks. User fairness was also considered in Lujak et al. [2015], where a mathematical programming model was proposed based on Nash welfare optimization. A linear programming based approach to control the trade-off between system and user perspectives, called proactive route guidance, was recently proposed in Angelelli et al. [2016]. The approach assigns paths to users with the goal of minimizing congestion while guaranteeing a maximum level of inconvenience to users. This is achieved by limiting the set of paths considered for an origin-destination pair to those that have a relative difference in length with respect to the shortest path below a given threshold. The approach hierarchically minimizes the maximum arc utilization, which measures the congestion and captures the system perspective, and the average travel inconvenience, that accounts for the user perspective. The fact that the approach is based on linear programming models makes it computationally valuable. However, the enumeration of all feasible paths is requested for every origin-destination pair and this step may be memory and time consuming, especially when big networks and/or high values of maximum allowed travel inconvenience are considered.

In this paper a heuristic is presented to generate the feasible paths for the proactive route guidance approach. Methods for the generation of paths in a road network have been widely studied in literature. According to Prato [2009] most of the deterministic path generation methods are based

on repeated shortest path searches over the network. These methods are popular because of the efficiency of shortest path algorithms. The well known k-shortest path algorithm proposed by Yen [1971] is an example of deterministic generation methods based on the shortest path calculation. Several variants of the k-shortest path algorithm have been developed. An interesting variant involves using more than one objective function (see Ramming [2001] and Van der Zijpp and Catalano [2005]). In Ramming [2001] a literature review on the main path generation methods is provided. Deterministic path generation is carried out also using heuristics as the link penalty and the link elimination methods. The idea underlying link penalty methods is to iteratively penalize the links belonging to the shortest paths as proposed in de la Barra et al. [1993]. In link penalty methods the weight of the links belonging to the shortest path is increased by a percentage and then a new shortest path search on the modified network is performed. Other variants are presented in Park and Rilett [1997], Bekhor and Prato [2006] and Bekhor et al. [2006]. The link elimination method consists in iteratively removing one or more links from the network and in performing a new shortest path search on the modified network. This latter method was first presented in Azevedo et al. [1993] and variants can be found in Prato and Bekhor [2006], Prato and Bekhor [2007] and Frejinger and Bierlaire [2007].

The heuristic presented in this paper can be seen as a link elimination method applied to the congested arcs of the network. The heuristic set of paths is generated through an iterative process. The algorithm starts with a set of paths that contains the shortest path for each origin-destination (OD) pair only. At each iteration, new paths that improve the current solution are identified and added to the restricted set until a stop condition is satisfied. Computational experiments show that the number of paths generated by the heuristic is smaller with respect to the complete set by one or two orders of magnitude on small instances and by higher orders of magnitude when the size of the instances increases. On instances with 150 nodes the complete enumeration takes an acceptable computational time and the solution obtained with the heuristic set of paths can be compared with the optimum obtained with the complete set. The results show that the quality of the heuristic solutions is very close to that of the optimal ones. In particular, if the maximum allowed travel inconvenience is 5%, the congestion (the maximum arc utilization) obtained with the heuristic set of paths coincides with the minimum congestion obtained with the complete enumeration. If the maximum allowed travel inconvenience is 10%, the heuristic congestion coincides with the optimum

on 38 over 40 instances. For higher values of the maximum allowed travel inconvenience the heuristic congestion increases on average by at most 0.79% (when maximum allowed travel inconvenience is 30%). The average travel inconvenience with the heuristic set of paths increases on average at most by 0.50 when maximum allowed travel inconvenience is 20%.

The paper is organized as follows. In Section 2 the proactive route guidance approach is recalled. In Section 3 the heuristic for the path generation is described. In Section 4 the results of a thorough computational analysis are presented and the benefits of the heuristic versus the complete enumeration are shown. Finally, some conclusions are drawn in Section 5.

2 The proactive route guidance approach

In order to make the paper self-contained, we recall in this section the proactive route guidance approach of Angelelli et al. [2016]. The problem is defined on a graph $G = (V, A)$ representing a road network, where vertices V represent road intersections and arcs A represent allowed directed links between intersections. For each arc $(i, j) \in A$, the practical capacity u_{ij} and its length l_{ij} are provided. The former represents the maximum rate of vehicles that can enter and traverse the arc in free-flow conditions, whereas the latter is proportional to the arc traversal time in free-flow conditions. A set C of origin-destination (OD) pairs is also given where each OD pair $c \in C$ is defined by three parameters: $O_c \in V$, $D_c \in V$ and $d_c > 0$, the origin, the destination and the rate of vehicles entering the network in O_c with destination D_c, respectively. The demand d_c of any OD pair $c \in C$ has to be routed through one or more paths in G from origin O_c to destination D_c. A path is measured on the basis of the so called *path travel inconvenience*, computed as the relative increment $\tau_{c,p}$ of the length of a path p with respect to the shortest path from O_c to D_c. A parameter τ, called maximum allowed travel inconvenience, defines the set of feasible paths P_c for each OD pair $c \in C$ as the set of paths p such that $\tau_{c,p} \leq \tau$. Traffic assignment is described by variables $y_{c,p}$, $c \in C$, $p \in P_c$, where variable $y_{c,p}$ represents the amount of demand routed on path $p \in P_c$. Parameters $a^{ij}_{c,p}$ connect feasible paths to arcs. More precisely, $a^{ij}_{c,p} = 1$ if arc (i, j) is traversed by path $p \in P_c$, 0 otherwise. The total rate of vehicles traversing an arc (i, j) is, thus, $x_{ij} = \sum_{c \in C} \sum_{p \in P_c} a^{ij}_{c,p} y_{c,p}$, and the *arc congestion level* is defined as the ratio x_{ij}/u_{ij}. If the arc congestion level exceeds 1, then the arc is said to be *congested*, while it is said *uncongested* otherwise. Analogously, we define the

path congestion level as the maximum congestion level of its arcs, and the *OD pair congestion level* as the maximum congestion level of its feasible paths. Finally, the *network congestion level* is the maximum arc congestion level (equivalently, the maximum path or OD pair congestion level). The network is said to be *congested* if its congestion level exceeds 1, and *uncongested* otherwise.

The proactive route guidance approach assigns the demand d_c of each OD pair $c \in C$ to a set of paths in P_c with the objective to minimize the so called weighted average travel inconvenience, that is the sum of the weighted travel inconvenience over all paths where the weight of a path is the ratio between the demand assigned to the path and the total demand. Assignment of demand to paths has to leave the network uncongested, if possible, or at the minimum congestion level otherwise.

The proactive route guidance approach consists in two hierarchical linear programming models: the *congestion model* and the *inconvenience model*. The congestion model finds the minimum network congestion level ρ^*. If $\rho^* \leq 1$, the inconvenience model is solved to minimize the weighted average travel inconvenience while keeping the network uncongested. Otherwise, the inconvenience model is solved imposing ρ^* as an upper bound on the network congestion level. In Table 1 we summarize the used notation.

The inconvenience model

$$\bar{\tau}^* \equiv \min \quad \frac{1}{\sum_{c \in C} d_c} \sum_{c \in C} \sum_{p \in P_c} \tau_{c,p} y_{c,p}$$

$$\rho \leq \max(1, \rho^*) \tag{1}$$

$$\frac{x_{ij}}{u_{ij}} \leq \rho \qquad \forall (i,j) \in A \tag{2}$$

$$x_{ij} = \sum_{c \in C} \sum_{p \in P_c} a_{c,p}^{ij} y_{c,p} \qquad \forall (i,j) \in A \tag{3}$$

$$d_c = \sum_{p \in P_c} y_{c,p} \qquad \forall c \in C \tag{4}$$

$$y_{c,p} \geq 0 \qquad \forall c \in C \quad \forall p \in P_c. \tag{5}$$

Constraints (1) and (2) ensure that the network remains uncongested if $\rho^* \leq 1$, or that its congestion

level is minimized when $\rho^* > 1$. Constraints (3) set the flow rate on arc (i,j) equal to sum of flows on paths containing the arc (i,j). Constraints (4) ensure that the demand of an OD pair $c \in C$ is completely routed on its feasible paths. Constraints (5) bound variables $y_{c,p}$ to be non-negative.

The minimum network congestion level ρ^* used in constraint (1) is the optimal value of the following congestion model.

The congestion model

$$\begin{aligned} \rho^* \equiv min \quad & \rho \\ \text{s.t.} \quad & (2) - (5) \end{aligned}$$

The congestion model shares with the inconvenience models the same decision variables and operative constraints. However, the congestion model focuses on the network congestion level only, while the inconvenience model without constraints (1) - (2) would take into account only user travel inconvenience.

In Angelelli et al. [2016] all the feasible paths from origin to destination for each OD pair are generated. The set of feasible paths obviously depends on the value of the maximum allowed travel inconvenience τ and the number of paths increases when τ increases. In the rest of the paper, we refer to this method as the *complete enumeration* (CE). It has been shown that, with the CE, not only the number of generated paths grows with τ, but that the total number of feasible paths, given τ, is, in the worst case, exponential in the instance size. Since each path is associated with a variable in the models, the number of variables in the models grows, in the worst case, exponentially.

For example, for a benchmark instance with 150 vertices and $\tau = 15\%$, 70125 paths had to be generated. For an instance with 300 vertices, the number of paths was 9649118. In Angelelli et al. [2016] it was observed that in the optimal solution of the proactive route guidance approach only few paths were used for each OD pair. This a characteristic of the solution that allows us to hope that a well designed heuristic can achieve high quality solutions.

Proactive route guidance approach notation

G	road network
V	set of vertices
A	set of arcs
u_{ij}	practical capacity of arc $(i,j) \in A$
l_{ij}	length of arc $(i,j) \in A$
C	set of OD pairs
d_c	flow rate for OD pair $c \in C$
O_c	origin vertex of OD pair $c \in C$
D_c	destination vertex of OD pair $c \in C$
SP_c	length of the shortest path for OD pair $c \in C$
$l_{c,p}$	length of path p from O_c to D_c
$\tau_{c,p}$	inconvenience of path p from O_c to D_c : $\tau_{c,p} = \frac{l_{c,p} - SP_c}{SP_c}$
τ	maximum allowed travel inconvenience
P_c	set of feasible paths for OD pair $c \in C$: $P_c = \{$ path p from O_c to $D_c \mid \tau_{c,p} \leq \tau\}$
$a_{c,p}^{ij}$	is 1 if path $p \in P_c$ contains arc (i,j), 0 otherwise
$y_{c,p}$	flow rate of OD pair $c \in C$ routed on path $p \in P_c$
x_{ij}	total flow rate entering arc $(i,j) \in A$: $x_{ij} = \sum_{c \in C} \sum_{p \in P_c} a_{c,p}^{ij} y_{c,p}$

Table 1. Notation

3 The heuristic path generation algorithm

The *Heuristic Path Generation* (HPG) algorithm aims at generating, with respect to the CE, a substantially smaller set of paths that contains most of the paths used in the optimal solution of the inconvenience model and, thus, such that the solution of the inconvenience model, computed using the heuristic set of paths, is close to the optimum.

The heuristic set of paths is generated through an iterative process. The algorithm starts with a set of paths, that we call restricted set, that contains the shortest path for each OD pair only. At each iteration, new paths are identified and added to the restricted set until a stop condition is satisfied. More precisely, at each iteration an estimate of ρ^* is computed by solving the congestion model using the paths currently available in the restricted set. Then, an improving set of paths, that can guarantee a reduction of the current congestion level of the network, is searched for. These paths are selected, for each OD pair, among the shortest ones that comply with the maximum allowed travel inconvenience τ. If an improving set of paths is found, then it is added to the restricted set, and the process continues. Otherwise, the algorithm stops and the current restricted set is the final heuristic set of paths.

The heuristic assignment of demand to the paths is found by solving the inconvenience model using the final restricted set of paths generated by the HPG algorithm and using as congestion value the final estimate of ρ^*.

The HPG algorithm is sketched in Algorithm 1. Variable P^{Res} represents the restricted set of paths and variable P^{Imp} represents the improving set. They are initialized with the empty set and the set of shortest paths for each OD pair, respectively. At each iteration the improving set is added to the restricted set, and the congestion model is solved on the new restricted set. The resulting optimal value ρ' and optimal solution y' are used in the next step to search for a new set of improving paths by means of the *SearchForImprovingPaths* routine that will be described in detail in Section 3.1. If the search is successful ($P^{Imp} \neq \emptyset$), the restricted set is updated and the congestion model is solved again. When routine *SearchForImprovingPaths* fails ($P^{Imp} = \emptyset$), the path generation ends. Then, the inconvenience model is solved using the restricted set of paths P^{Res} and as ρ^* the current estimate ρ'. The heuristic value of the objective function of the inconvenience model is denoted by $\bar{\tau}$.

Algorithm 1: Heuristic Path Generation (HPG) algorithm

input : Network G and set of OD pairs C
output: Approximate value of the minimum weighted average travel inconvenience $\bar{\tau}$
global : G, C
– $P^{Res} := \emptyset$;
– $P^{Imp} :=$ set of shortest paths from origin O_c to destination D_c for each $c \in C$;

while $P^{Imp} \neq \emptyset$ **do**
  – $P^{Res} := P^{Res} \bigcup P^{Imp}$;
  – solve congestion model on the path set P^{Res}:
    - $\rho' :=$ optimal value;
    - $y' :=$ optimal solution;
  /* searches for a set of paths able to improve the solution */
  – $P^{Imp} :=$ *SearchForImprovingPaths*(P^{Res}, y', ρ');
– solve the inconvenience model on the path set P^{Res} and $\rho^* = \rho'$:
  - $\bar{\tau} :=$ optimal value;
return $\bar{\tau}$

3.1 Searching for an improving set of paths

At each iteration of the HPG algorithm the congestion model provides a heuristic solution. The objective of the routine *SearchForImprovingPaths* is to find an improving set of paths, that is a set of paths such that, when added to the restricted set, the congestion model produces a new (sub-)optimal solution with a smaller network congestion. In order to provide details on how the routine *SearchForImprovingPaths* works we need to introduce a few definitions and state some properties of a feasible solution of the congestion model.

Let variables $\widehat{y}$ indicate a feasible solution of the congestion model, which we call a feasible flow assignment, and let $\widehat{\rho}$ be the corresponding network congestion level. For all $c \in C$, let $\widehat{P}_c \subseteq P_c$ be the set of paths with a positive flow ($\widehat{y}_{c,p} > 0$) and $\widehat{P} = \bigcup_{c \in C} \widehat{P}_c$. Finally, an arc is said to be critical if its congestion level equals $\widehat{\rho}$. Analogously, an OD pair c and a path $p \in P_c$ are said to be critical if their congestion levels equal $\widehat{\rho}$, respectively.

Proposition 1. *Let $\widehat{y}$ be a feasible flow assignment with network congestion level $\widehat{\rho}$ and let c be a critical OD pair. If a path $p \in P_c$ exists with congestion level strictly less than $\widehat{\rho}$, then a feasible flow assignment exists such that:*

1. *No arc increases its congestion level to $\widehat{\rho}$ or above.*

2. *Congestion level of path p is strictly less than $\widehat{\rho}$.*

3. *Congestion level of c is strictly less than $\widehat{\rho}$.*

4. *No other OD pair increases its congestion level to $\widehat{\rho}$.*

5. *Network congestion level does not increase.*

Proof. Let us move an arbitrarily small flow $\epsilon > 0$ from every critical path in $\widehat{P}_c$ to path p.

1. The only arcs that may increase their congestion level are those in p as they receive a positive flow ϵ from other paths. Provided ϵ is small enough to keep the path congestion level of p strictly lower than $\widehat{\rho}$, no arc increases its congestion level to $\widehat{\rho}$ or above.

2. It follows directly from Proposition 1.1.

3. All paths in $\widehat{P}_c$ with congestion level $\widehat{\rho}$ yield a positive flow ϵ to p. Consequently, their congestion level must decrease by a positive quantity. Proposition 1.1 completes the argument.

4. The only OD pairs that can increase their congestion level are those with paths sharing arcs with p, but from Proposition 1.2 we know that all these arcs maintain a congestion strictly less than $\widehat{\rho}$.

5. It follows directly from Proposition 1.1.

□

Corollary 1. *If the new flow assignment $\widetilde{y}$ produced under Proposition 1 assumptions has network congestion level $\widetilde{\rho} = \widehat{\rho}$, then:*

1. *Critical OD pairs in $\widetilde{y}$ are a proper subset of critical OD pairs in $\widehat{y}$. In particular, OD pair c is not critical in $\widetilde{y}$.*

2. *Critical arcs in $\widetilde{y}$ are a proper subset of critical arcs in $\widehat{y}$. In particular, critical arcs in critical paths $p \in \widehat{P}_c$ are not critical in $\widetilde{y}$.*

Proof. It follows directly from Proposition 1. □

Proposition 2. *Let $\widehat{y}$ be a feasible flow assignment with network congestion level $\widehat{\rho}$ and let h be a critical arc in a critical path $p \in \widehat{P}_c$ for some critical OD pair $c \in C$. Let us also assume that a path $p' \in P_c$ exists such that p' may contain some critical arcs from p, but not h and no critical arc from any other path.*

Then, a feasible flow assignment exists such that:

1. *No arc increases its congestion level to $\widehat{\rho}$ or above.*
2. *Congestion level of h is strictly less than $\widehat{\rho}$.*
3. *Congestion level of p' is at most $\widehat{\rho}$.*
4. *No other OD pair increases its congestion level to $\widehat{\rho}$.*
5. *Network congestion level is at most $\widehat{\rho}$.*

Proof. Let us move an arbitrarily positive flow $\epsilon > 0$ from p to p' small enough to keep the path congestion level of p' stricly lower than $\widehat{\rho}$.

1. Let us consider four different types of arcs:
   - Arcs shared by paths p and p' maintain their congestion level as the flow moved from p to p' converges again on these arcs. In particular, shared critical arcs maintain their congestion level at $\widehat{\rho}$, while other shared arcs maintain their congestion strictly below $\widehat{\rho}$.
   - Arcs in p but not in p' decrease their congestion level as the total flow through them is decreased by ϵ. This is true in particular for arc h.
   - Arcs in p' but not in p increase their congestion level; however, provided that ϵ is small enough, their congestion levels remain strictly smaller than $\widehat{\rho}$.
   - Arcs neither in p nor in p' remain unchanged.
2. It follows directly from argument for Proposition 2.1.
3. It follows directly from argument for Proposition 2.1.
4. The only OD pairs that can increase their congestion level are those with paths sharing arcs with p, but from Proposition 2.2 we know that all these arcs maintain a congestion strictly less than $\widehat{\rho}$.

5. It follows directly from argument for Proposition 2.1.

□

Corollary 2. *If the new flow assignment $\widetilde{y}$ produced under Proposition 2 assumptions has network congestion level $\widetilde{\rho} = \widehat{\rho}$, then critical arcs in $\widetilde{y}$ are a proper subset of critical arcs in $\widehat{y}$. In particular, arc h is not critical in $\widetilde{y}$.*

Proof. It follows directly from Proposition 2. □

Remark 1. *If assumptions of Proposition 1 are satisfied, assumptions of Propositions 2 are satisfied for all critical arcs in $\widehat{P}_c$. In this sense Proposition 1 is a particular case of Proposition 2.*

Proposition 3. *Let a feasible flow assignment $\widehat{y}$ on which Proposition 2 can be applied be given and let $\widehat{\rho}$ be its network congestion level. Let $\widetilde{y}$ the feasible flow assignment resulting from the Proposition 2 application and let $\widetilde{\rho}$ be its network congestion level. Then there are three possible cases:*

- *$\widetilde{\rho}$=$\widehat{\rho}$ and Proposition 2 can be applied on the new flow assignment $\widetilde{y}$;*
- *$\widetilde{\rho}$=$\widehat{\rho}$ and Proposition 2 cannot be applied because there exists no path satisfying the assumptions of the Proposition 2;*
- *$\widetilde{\rho}$<$\widehat{\rho}$, i.e. there are no more critical OD pair.*

Proof. The new flow assignment $\widetilde{y}$ can have network congestion level equal or smaller than $\widehat{\rho}$. If $\widetilde{\rho}$<$\widehat{\rho}$ than there are no more critical OD pairs since there is no arc with arc congestion level greater or equal to $\widehat{\rho}$. According to Corollary 2, if network congestion level $\widetilde{\rho}$, corresponding to the new feasible flow assignment $\widetilde{y}$ found by Proposition 2, equals $\widehat{\rho}$, the number of arcs with congestion level $\widehat{\rho}$ is reduced with respect to $\widehat{y}$. Thus, either Proposition 2 can be applied on $\widetilde{y}$ and, hence, the new flow assignment satisfy one of the three possible cases or the Proposition 2 cannot be applied because there exists no path satisfying the assumptions. □

Remark 2. *Let $\widetilde{y}$ be the new flow assignment obtained after applying the transformation described in Proposition 1 or Proposition 2. If $\widetilde{\rho}$=$\widehat{\rho}$ and a critical OD pair $\widetilde{c}$ for $\widetilde{y}$ has a path $\widetilde{p} \in \widetilde{P}_{\widetilde{c}}$ with no critical arcs, then Proposition 1 can be recursively applied on feasible flow $\widetilde{y}$, OD pair $\widetilde{c}$ and path $\widetilde{p}$.*

The basic step of the *SearchForImprovingPaths* routine is to search for a path $p' \in P$ that satisfies the assumptions of Proposition 2 for some critical arc h of a path $p \in \widehat{P}_c$ for some critical OD pair c. According to Corollary 2, such path is able to relieve arc h and, according to Remark 2 and Corollary 1, may also relieve a critical path, and, recursively, even more critical arcs, and critical paths. The recursion stops when the relieved arc is not able to relieve a whole path. We use the word *relieve* with respect to a critical element (arc, path, OD pair) in the sense that a new traffic assignment can be found such that the congestion level of the relieved critical element is strictly less than $\widehat{\rho}$ while the network congestion level does not get worse. If, according to Proposition 3, we manage to iterate the basic step until all critical OD pairs are relieved, we finally improve the network congestion. Notice that according to Remark 1 we do not need to check explicitly for assumptions of Proposition 1.

Routine *SearchForImprovingPaths* is sketched in Algorithm 2. The initialization phase exploits the current optimal solution of the congestion model on $\widehat{P}$, $\widehat{y}$ and $\widehat{\rho}$. The set $ImprovingPaths$ is first initialized to the empty set. Set A^{crit} is initialized with all critical arcs, set C^{crit} is initialized with all critical OD pairs, and sets $A^{crit}_{c,p}$ for all $c \in C^{crit}$ and $p \in \widehat{P}_c$ are initialized with critical arcs in path $p \in \widehat{P}_c$.

During the main loop execution, for each critical OD pair $c \in C^{crit}$, for each critical path $p \in \widehat{P}_c$ and for each critical arc $h \in A^{crit}_{c,p}$, routine *SearchForNewPath* is used to search for a path p' from O_c to D_c such that the assumptions of Proposition 2 are satisfied. If the search is successful, the new path is stored in $ImprovingPaths$ and *RelieveCriticalElements* routine is run to find the relieved critical elements in a potential flow assignment on $\widehat{P} \bigcup ImprovingPaths$ and, consequently, update the critical sets $C^{crit}, A^{crit}, A^{crit}_{c,p}$. Details on this task will be given in Section 3.3. When C^{crit} has been emptied, all critical OD pairs are relieved and the algorithm terminates with all the paths stored in $ImprovingPaths$. This set guarantees that there exists a feasible flow assignment on $\widehat{P} \bigcup ImprovingPath$ that allows a congestion network smaller than $\widehat{\rho}$. If, at some iteration, the search of a new path p' fails on all triples (c, p, h), then the algorithm is not able to relieve some of the critical OD pairs and returns an empty set.

Algorithm 2: *SearchForImprovingPaths*

input : $\widehat{P}, \widehat{y}, \widehat{\rho}$
output: A list of paths $ImprovingPaths$
global : $G, C, C^{crit}, A^{crit}, A^{crit}_{c,p}$
– $ImprovingPaths := \emptyset$;
– Using solution $\widehat{P}$, $\widehat{y}$, $\widehat{\rho}$, initialize:
  – A^{crit} := set of critical arcs;
  – C^{crit} := set of critical OD pairs;
  – $A^{crit}_{c,p}$:= set of critical arcs $\forall c \in C^{crit}, p \in \widehat{P}_c$;
while $C^{crit} \neq \emptyset$ **do**
  – $success := false$;
  for $c \in C^{crit}$ **do**
    for $p \in \widehat{P}_c$ **do**
      for $h \in A^{crit}_{c,p}$ **do**
        – $p' := SearchForNewPath(c, p, h)$;
        if p' *is not null* **then**
          – $success := true$;
          – add p' to $ImprovingPaths$;
          – Execute $RelieveCriticalElements(h)$ to update $C^{crit}, A^{crit}, A^{crit}_{c,p}$;
  if $success = false$ **then**
    – **return** $\emptyset$;
– **return** $ImprovingPaths$;

3.2 Searching for the new path

Given an OD pair $c \in C^{crit}$, a critical path $p \in \widehat{P}_c$ and a critical arc $h \in A^{crit}_{c,p}$, the *SearchForNewPath* routine, sketched in Algorithm 3, is devoted to finding a feasible path p' from O_c to D_c able to relieve arc h. In order to do that, a subgraph of G is created: all critical arcs A^{crit} are first removed from the graph, then all arcs in $A^{crit}_{c,p}$ but h are reinserted. In other words, the only critical arcs allowed in the new path are those that have already been relieved and the not-yet-relieved ones in p but h. The search for a path could address any path in P_c, here we search for the shortest one in view of the fact that this path will eventually concur to define a solution for the inconvenience model.

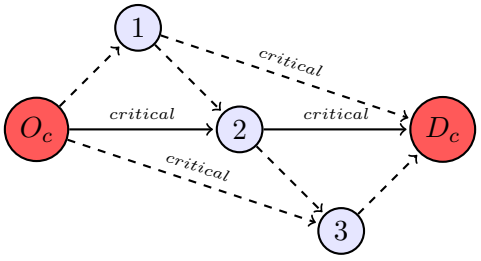
Algorithm 3: *SearchForNewPath*

input : c, p, h
output: p'
global : $G, A^{crit}, A^{crit}_{c,p}$
– $A' := (A \setminus A^{crit}) \bigcup (A^{crit}_{c,p} \setminus \{h\})$;
– $G' := (A', V)$;
– Search in G' the shortest feasible path p' from O_c to D_c;
if p' *exists* **then**
  – **return** p'
else
  – **return** *null*

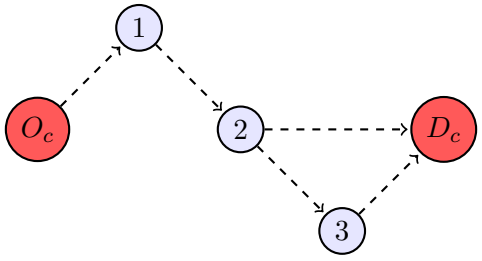
In Figure 1 we show an example of how *SearchForNewPath* routine operates in two different runs. In Figure 1(a) we have a critical OD pair c with a single path $p1 =< O_c - 2 - D_c >$ in $\widehat{P}$ emphasized with solid lines. Other arcs available in the graph are depicted with dashed lines. Critical arcs are appropriately labeled. For ease of exposition, suppose that all arcs have length 1 and that maximum allowed length is 3. Notice that path $< O_c - 1 - 2 - 3 - D_c >$, which could relieve both arcs $(O_c, 2)$ and $(2, D_c)$, is not feasible having length 4.

Suppose now that the first run is on path $p = p1$ and arc $h = (O_c, 2)$. Figure 1(b) shows the subgraph obtained after removing critical arcs not in p $\{(1 - D_c), (O_c, 3)\}$ and arc h; Figure 1(c) shows the new path $p2 =< O_c - 1 - 2 - D_c >$ found and able to relieve arc h.

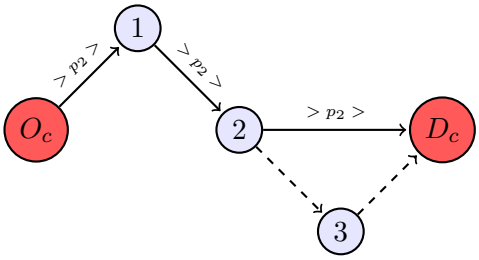
In Figure 1(d) we see the generated subgraph in a second run of *SearchForNewPath* routine where $p = p1$ and $h = (2, D_c)$: critical arcs not in p ($(1, D_c)$ and $(O_c, 3)$) and arc h are removed. Figure 1(e) shows the new path $p3 =< O_c - 2 - 3 - D_c >$ found and able to relieve arc h.



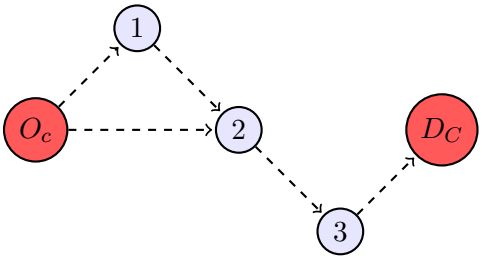
(a) Critical arcs based on $\widehat{P}, \widehat{y}, \widehat{\rho}$



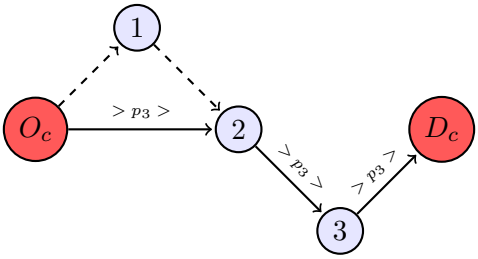
(b) Subgraph G' with $p_1 =< O_c - 2 - D_c >$, $h = (O_c, 2)$



(c) New path relieving arc $(O_c, 2)$



(d) Subgraph G' with $p =< O_c - 2 - D_c >$, $h = (2, D_c)$



(e) New path relieving arc $(2, D_c)$

Figure 1. An example of *SearchForNewPath* routine for a path containing two critical arcs

3.3 Relieving critical elements

When a new path p' is found by *SearchForNewPath* routine, at least one arc h can be relieved. As a consequence of this fact, the network congestion level cannot increase and the number of arcs with congestion level $\widehat{\rho}$ decreases as well (see Corollary 2). For this reason we remove the relieved arc from A^{crit} and all $A^{crit}_{c,p}$ where present. By doing this it could happen that a set $A^{crit}_{c,p}$ is emptied for some $c \in C^{crit}$ and $p \in \widehat{P}_c$. In this lucky case, according to Remark 2 and Proposition 2, we know that all critical arcs on p are relieved. Thus, all arcs still in $A^{crit}_{c,p}$ for $p \in \widehat{P}_c$ are automatically relieved and can be recursively treated by *RelieveCriticalElements* without adding any new path to $\widehat{P}_c$. Moreover, OD pair c is relieved and we remove it from C^{crit}.

This is what *RelieveCriticalElements* routine does. Details are provided in Algorithm 4. In particular, recursion is implemented in iterative form by adding relieved arcs to a queue.

Algorithm 4: *RelieveCriticalElements*

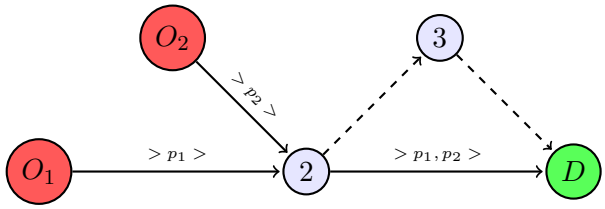
```
input  : h
global : P̂, C^crit, A^crit, A^crit_{c,p}
– R := {h};
while R ≠ ∅ do
    – Remove one arc a from R;
    – Remove a from set A^crit;
    for all c ∈ C^crit and p ∈ P̂_c do
        – Remove a from A^crit_{c,p}, if present;
    for all c ∈ C^crit do
        if p ∈ P̂_c exists such that A^crit_{c,p} = ∅ then
            – Remove c from C^crit;
            for all p ∈ P̂_c do
                – Add A^crit_{c,p} to R; // avoid duplicates
– return;
```

In Figure 2 an example is shown of how the *RelieveCriticalElements* routine works. In Figure 2(a) we see the optimal solution produced by the congestion model at the first iteration of the HPG algorithm. Set $\widehat{P}$ contains paths $p_1 =< O_1 - 2 - D >$ and $p_2 =< O_2 - 2 - D >$.

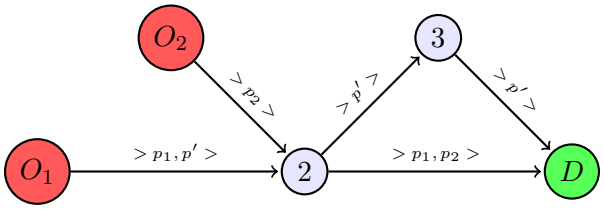
In a first scenario let us assume that only arc $(2, D)$ is critical. *SearchForImprovingPaths* routine would initialize $A^{crit} = \{(2, D)\}, A^{crit}_{1,p_1} = \{(2, D)\}, A^{crit}_{2,p_2} = \{(2, D)\}$ and $C^{crit} = \{1, 2\}$. When *SearchForImprovingPaths* routine executes *SearchForNewPath* on $c = 1, p = p_1$, and $h = (2, D)$ a path $p' =< O_1 - 2 - 3 - D >$ is found (see Figure 2(b)). Now arc $(2, D)$ can be relieved: routine *Relieve-*

CriticalElements removes arc $(2,D)$ from $A^{crit}, A^{crit}_{1,p_1}$ and A^{crit}_{2,p_2}; since A^{crit}_{1,p_1} and A^{crit}_{2,p_2} are empty, OD pairs 1 and 2 are removed from C^{crit}. Since C^{crit} is now empty, *SearchForImprovingPaths* return path p' which is sufficient to improve current solution.

In another scenario, let us assume that in Figure 2(a) arcs $(O_1,2)$ and $(2,D)$ are critical. *SearchForImprovingPaths* routine would initialize $A^{crit} = \{(O_1,2),(2,D)\}, A^{crit}_{1,p_1} = \{(O_1,2),(2,D)\}, A^{crit}_{2,p_2} = \{(2,D)\}$ and $C^{crit} = \{1,2\}$. When *SearchForImprovingPaths* routine executes *SearchForNewPath* on $c = 1, p = p_1$, and $h = (2,D)$ a path $p' =< O_1 - 2 - 3 - D >$ is found (see Figure 2(b)). Now arc $(2,D)$ can be relieved: routine *RelieveCriticalElements* removes arc $(2,D)$ from $A^{crit}, A^{crit}_{1,p_1}$ and A^{crit}_{2,p_2}; since A^{crit}_{2,p_2} only is empty, OD pair and 2 is removed from C^{crit}. Routine *SearchForImprovingPaths* would try another iteration to relieve OD pair 1.



(a) Solution of the congestion model on the first restricted set



(b) A new path p' for OD pair 1

Figure 2. An example of *RelieveCriticalElements* routine

3.4 HPG may miss to find some optimal paths

The HPG algorithm aims at generating a good solution for the inconvenience model by iteratively solving restricted versions of the congestion model. At each iteration a set of paths to be added to a restricted set are searched for in such a way that the congestion level can be reduced. When the *SearchForImprovingPaths* routine ends with a set of paths, we are sure that the current solution can be improved, but the converse is not true. Here, we provide two examples depicted in

Figures 3 and 4 to illustrate this point.

Figure 3 shows an instance in which the network congestion level obtained with the HPG algorithm is suboptimal. Let the network be as in Figure 3(a) and let the maximum allowed travel inconvenience be $\tau = 30\%$. All arcs have capacity u equal to 1, all the four OD pairs $C = \{1,2,3,4\}$ have demand equal to 1 and the arc duration is equal to 1 for each arc. In Figure 3(b) the optimal flows for the congestion and for the inconvenience models are shown: labels f_c on arcs represent the flow of OD pair c. It is easy to see that the demand is completely routed and the network congestion level is $\frac{5}{6}$. However, the HPG algorithm stops with a network congestion level of 1. That happens because the algorithm misses some paths used in the optimal solution of the congestion model. In the first iteration of *SearchForImprovingPaths* the shortest paths only are considered (Figure 3(c)). All OD pairs are critical and critical arcs are shown in Figure 3(d). The *SearchForNewPath* routine finds a new path only for OD pair 3 $SP3_2 \equiv < O_3 - D_4 - D_3 >$ (Figure 3(e)), the critical arc set is reduced by the *RelieveCriticalElements* and the critical OD pairs set becomes $C^{crit} = \{1,2,4\}$ (Figure 3(f)). Now *SearchForNewPath* is not able to relieve any other arc and fails. The same does *SearchForImprovingPaths* routine. Thus, the HPG algorithm stops at the value $\rho' = 1$. However, in this case the HPG algorithm finds the optimal solution of the inconvenience model because the shortest paths are sufficient to keep the network uncongested.

In Figure 4 an example where the HPG algorithm finds the minimum network congestion level and fails to find the optimal solution of the inconvenience model is provided. Let us consider a network with 2 OD pairs $C = \{1,2\}$, $\tau = 30\%$, and demand $d_1 = 8$ and $d_2 = 7$. Capacities and arc durations are as shown in Figure 4(a). In Figure 4(b) the optimal flows for the congestion and the inconvenience model are shown: labels f_c on arcs represent the flow of OD pair c on the arc. It is easy to see that the demand is completely routed, the network congestion level is equal to 1 and the weighted average travel inconvenience is around 2.5%. The paths considered in the first iteration of *SearchForImprovingPaths* are shown in Figure 4(c), critical OD pairs are $\{1,2\}$; there is only one critical arc with congestion level 1.5 shown in Figure 4(d). First iteration of routine *SearchForNewPath* finds a path $SP1_2$ represented in Figure 4(e) that relieves the only critical arc and OD pairs 1 and 2 altogether. Thus, *SearchForImprovingPaths* is successful and returns only one path $SP1_2$. The new obtained network congestion level is 1 with only one critical arc shown in Figure 4(f). At this point, during the second iteration of *SearchForImprovingPaths*,

routine *SearchForNewPath* fails and the same does *SearchForImprovingPaths*. The set of paths for the inconvenience model is suboptimal, although the congestion network level is optimal, and the resulting weighted average travel inconvenience is $8.\bar{3}\%$.

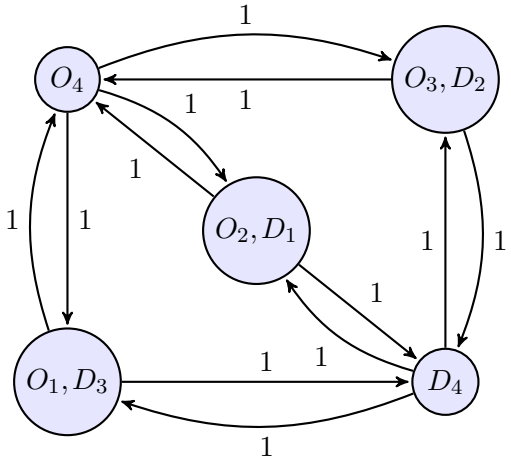
4 Computational results

The HPG algorithm and the CE were implemented in Java, and the optimization models were solved by CPLEX 12.6.0. The experiments were run on a Windows 64-bit computer with Intel Xeon processor E5-1650, 3.50 GHz, and 16 GB Ram. Experiments are organized in two parts. The first part is mainly devoted to comparing the set of paths generated with the CE and the set generated with the HPG algorithm on instances of increasing size. Experiments were carried out using 8 benchmark instances, with up to 330 nodes, and values of τ ranging from $\tau = 0\%$ to $\tau = 25\%$ with step 5%. The second part is devoted to comparing the optimal solutions of the congestion and inconvenience models (with paths generated through the CE) and the heuristic solutions of the models when only the paths generated with the HPG algorithm are used. Here experiments were carried out on 40 benchmark instances with 150 nodes and values for τ ranging from $\tau = 0\%$ to $\tau = 35\%$ with step 5%. The instances were generated taking into account different demand patterns and point attractiveness distributions as explained thoroughly in Angelelli et al. [2016] and are available at `http://or-brescia.unibs.it/instances`. The statistics collected for each instance are described in Section 4.1. Results for the former set of experiments are presented and discussed in Section 4.2. Section 4.3 is devoted to summarize results for the latter set of experiments.

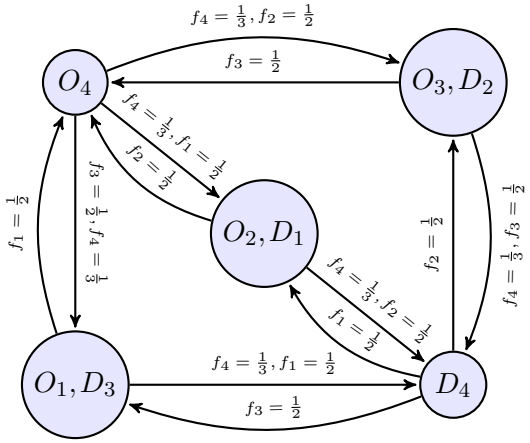
4.1 Statistics

A number of statistics on the CE and the HPG algorithm are collected or computed on all the tested instances.

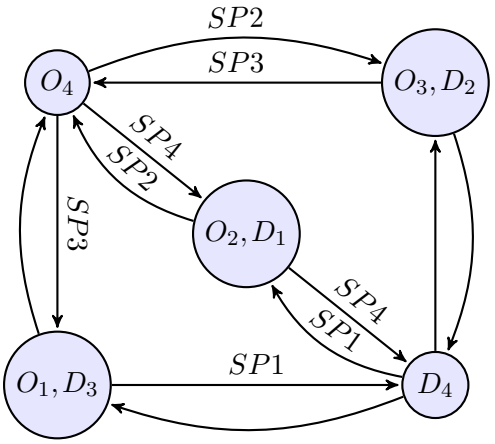
- Computational time. The computational time is computed considering the total computational time. For the CE this is the time to generate the paths and to solve the congestion and the inconvenience models. For the HPG algorithm it is the time required by the HPG



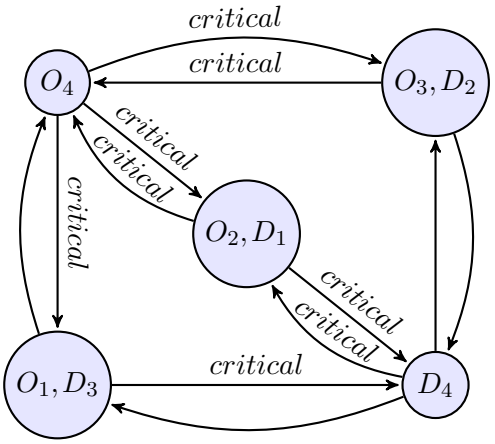
(a) Initial graph with arc durations



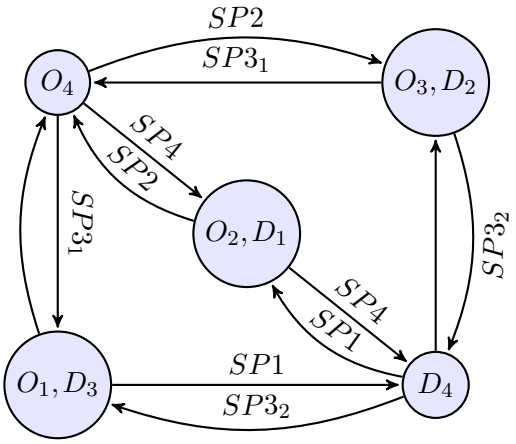
(b) Optimal flows for the congestion and the inconvenience models



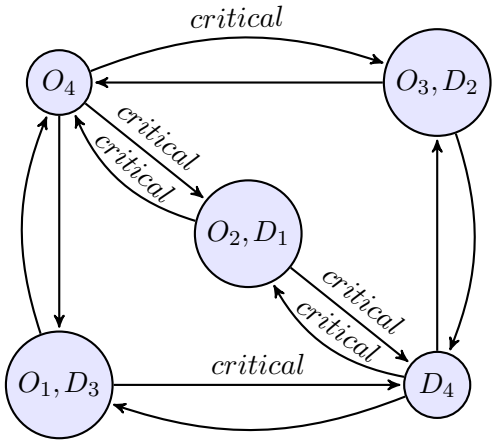
(c) Initial paths for the HPG algorithm: shortest paths



(d) Critical arcs with the shortest paths

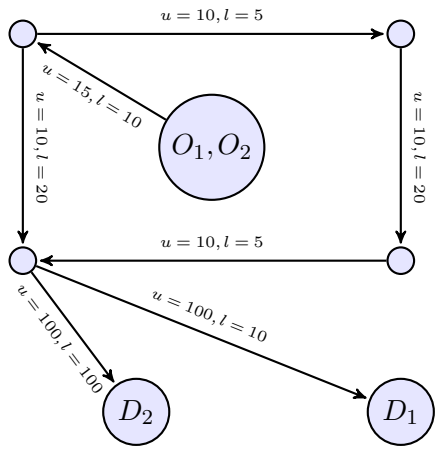


(e) Augmented path set after the first iteration of *SearchForNewPath*

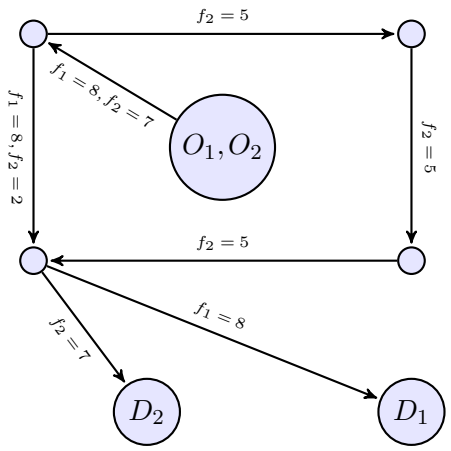


(f) Critical arcs after the first iteration of *SearchForNewPath*

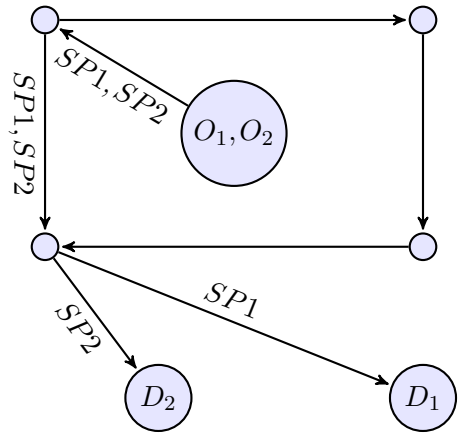
Figure 3. Failure of *SearchForImprovingPaths* routine at the first iteration



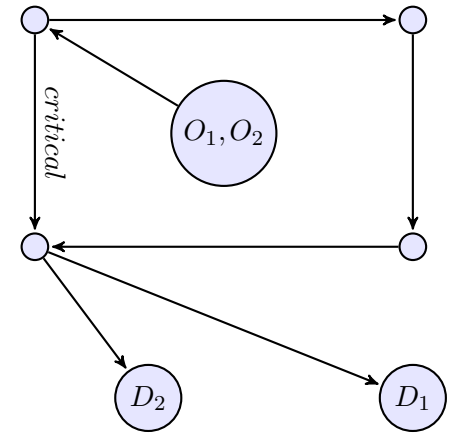
(a) Initial graph with arc capacities u and durations l



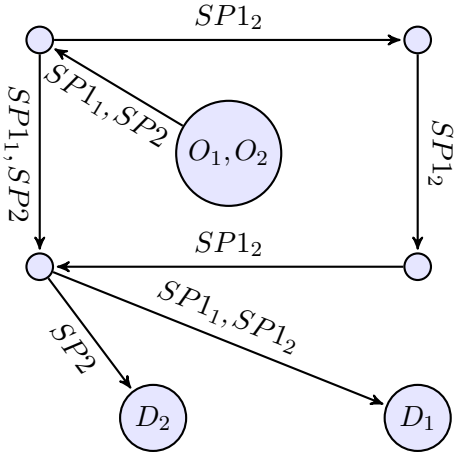
(b) Optimal flows for the congestion and the inconvenience models



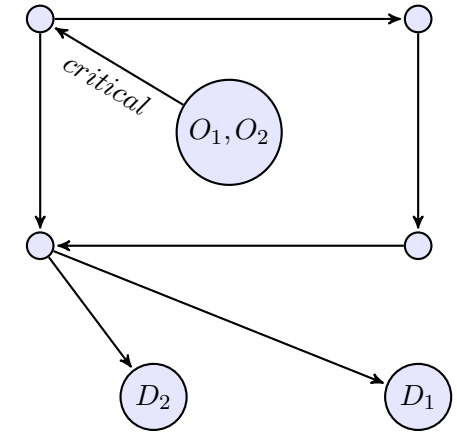
(c) Initial paths for the HPG algorithm: shortest paths



(d) Critical arcs with the shortest paths



(e) New path set after the first iteration of *SearchForImprovingPaths*



(f) Critical arc set during the second iteration of *SearchForImprovingPaths*

Figure 4. Failure of *SearchForImprovingPaths* routine at the second iteration

algorithm.

- Network Congestion.
  - network congestion level: the network congestion level corresponding to the generated solution.
  - the fraction of arcs falling in each of the following four classes for different τ values:
    - unused arcs ($x_{ij}/u_{ij} = 0$)
    - non-congested arcs ($0 < x_{ij}/u_{ij} \leq 1$)
    - lightly congested arcs ($1 < x_{ij}/u_{ij} \leq 1.5$)
    - heavily congested arcs ($1.5 < x_{ij}/u_{ij}$)

    For this statistic results are reported in a graphical form.
  - relative congestion gap $\epsilon_\tau = \frac{\rho_{HPG} - \rho_{CE}}{\rho_{CE}}$, where ρ_{CE} is the optimum of the congestion model (with paths generated through CE) while ρ_{HPG} is the value of the congestion model when paths are generated with the HPG algorithm.
- User Experience.
  - weighted average travel inconvenience corresponding to the generated solutions.
  - absolute inconvenience gap, $\kappa_\tau = \bar{\tau}_{HPG} - \bar{\tau}_{CE}$, where $\bar{\tau}_{CE}$ is the optimum of the inconvenience model (with paths generated through CE), while $\bar{\tau}_{HPG}$ is the value of the inconvenience model when paths are generated with the HPG algorithm. Usually gaps are computed in terms of relative error but the weighted average travel inconvenience is a percentage itself and, using a percentage of a percentage as quality measure can be misleading because if the weighted average travel inconvenience is very small, then the relative error will turn out to be very high, even if the difference between the two solutions is small.
- Memory usage. The number of generated paths.
- Iterations. The number of iterations performed by the HPG algorithm.

4.2 Comparison of the heuristic set of paths with the complete set

As already mentioned, experiments were carried out considering 8 networks, with a number of nodes that ranges from 120 to 330 and with τ values ranging from $\tau = 0\%$ to $\tau = 25\%$ with step 5%. In Tables 2 and 3 the statistics are presented. At each step the number of nodes in the network is increased by 30 nodes. Note that, for instances with a number of nodes greater than 150 and for some values of τ, the statistics for the CE are not shown. This is because the solver was not able to produce a solution because it ran out of memory. For these cases we report the number of paths generated by the CE with the computational time required to generate them (values are marked with *). We observe how slowly the number of paths generated by the HPG algorithm grows with respect to the CE as the number of nodes increases. With $\tau = 25\%$, for the 120 nodes instance this number is approximately 3% of the number generated by the CE while for the 180 nodes instance it is approximately 0.4%. The percentage with $\tau = 25\%$ and a higher number of nodes is not available because the solver is not able to handle the number of variables needed for the CE. However, considering $\tau = 20\%$, for the 210 nodes instance the percentage is approximately 0.3%. Considering 240, 270 and 300 nodes instances, the percentage experienced with $\tau = 15\%$ is 0.4%, 0.2% and 0.06%, respectively. For the 330 nodes instance, the value of τ for which we have the statistic is 10% and the percentage of paths generated with the HPG algorithm with respect to the CE is 0.1%. These statistics indicate that the HPG algorithm requires a memory that may be 3 orders of magnitude less than the CE. From the point of view of the computational time, it is possible to observe the same positive behaviour of the HPG algorithm. The percentage of the computational time required by the HPG algorithm with respect to the CE is around 50% considering $\tau = 25\%$ and 120 nodes, whereas considering 150 nodes this percentage decreases to 2.7%. For a higher number of nodes we do not have results for $\tau = 25\%$. However, with $\tau = 20\%$ on the 180 nodes instance the HPG algorithm takes only 2.8% of the computational time taken by the CE. The best result in terms of computational time happens with the 300 nodes instance where the largest value of τ for which we have the CE solution is 10%. In this case the HPG algorithm takes 0.28% of the time spent by the CE.

After the analysis of the memory and time saving of the HPG algorithm on these instances, we consider its effectiveness in generating high quality solutions. The relative congestion gap ϵ_τ is very low. The maximum experienced value of ϵ_τ is around 4.4% with 150 nodes and with $\tau = 25\%$.

From the point of view of the weighted average travel inconvenience, the κ_τ value has a tendency to increase with τ, but is always below 1.1%. This means that a user routed using the HPG algorithm can experience, on average, a travel inconvenience that is 1.1% higher than with the CE. Furthermore, this value appears with high values of τ. With $\tau = 25\%$ the weighted average travel inconvenience is on average 7%. It means that with the HPG algorithm users will experience on average 8%, in the worst case.

4.3 Effectiveness of the heuristic set of paths

In this section we summarize the results obtained on 40 benchmark instances with 150 nodes and values of τ ranging from $\tau = 0\%$ to $\tau = 35\%$ with step 5% In Table 4 the computational times required by the CE and by the HPG algorithm are shown. The HPG algorithm confirms to be widely less time consuming than the CE, especially when high values of τ are considered. In fact, with $\tau = 35\%$ the CE, on average, spends more than 1000 seconds per instance while the HPG algorithm spends 90 seconds per instance. In this case the use of the HPG algorithm with τ up to 10% is not convenient because it takes an amount of time similar to the CE. However, as shown in Table 3, the HPG algorithm becomes faster than the CE when the number of nodes increases, even for small values of τ.

In Tables 5 and 6 statistics on the congestion gap ϵ_τ are shown. In Table 5 the average and the maximum ϵ_τ over all instances and for different τ values are shown. Note that, for $\tau = 5\%$, ϵ_τ is 0 as all instances are optimally solved with respect to the congestion model. With $\tau = 10\%$ two instances present positive, although negligible, values of ϵ_τ. With higher values of τ, the average ϵ_τ increases. The worst case takes place when the τ value is 30% where the average ϵ_τ value is equal to 0.79%. The maximum value of ϵ_τ follows the behaviour of the average ϵ_τ with no congestion gap with $\tau = 5\%$ and a very small value, 0.02%, with $\tau = 10\%$. As in the average case, the worst case appears with $\tau = 15\%$ for which the maximum value is 3.26%.

Table 6 reports the number of instances reporting ϵ_τ in different classes of values. Almost all the instances are experiencing an ϵ_τ value under 3%. One exception is with $\tau = 15\%$ where 5% of the instances (2 over 40 instances) are experiencing a value of ϵ_τ between 3% and 3.26%, which is the maximum ϵ_τ over all the instances with $\tau = 15\%$. The other exception is when $\tau = 25\%$ is considered and 5% of the instances have a congestion gap between 3% and 3.05% (maximum

Number of nodes	Stats	τ 0%	5%	10%	15%	20%	25%
120	Paths CE	1170	2335	6766	17448	39528	79935
	Paths HPG	1170	1242	1549	2034	2367	2676
	Time CE (sec)	1	2	4	9	18	35
	Time HPG (sec)	0	1	4	7	10	18
	Congestion gap ϵ_τ (%)	0	0	0	0	0	0
	Inconvenience gap κ_τ (%)	0	0.0001	0.001	0.002	0.005	0.008
150	Paths CE	1170	4157	18849	70215	216528	575120
	Paths HPG	1170	1343	2460	3758	4626	5305
	Time CE (sec)	1	4	13	42	285	1609
	Time HPG (sec)	0	3	15	29	41	45
	Congestion gap ϵ_τ (%)	0	0	0	0	0	4.4
	Inconvenience gap κ_τ (%)	0	0.04	0.4	0.7	1.1	1
180	Paths CE	1170	6758	45828	230508	916527	3068987
	Paths HPG	1170	1286	2876	4123	10397	12508
	Time CE (sec)	1	7	39	376	4848	1089(*)
	Time HPG (sec)	0	2	20	31	137	169
	Congestion gap ϵ_τ (%)	0	0	0	0	1.8	–
	Inconvenience gap κ_τ (%)	0	0	0.4	0.6	0.2	–
210	Paths CE	1170	10496	98330	640963	3212485	–
	Paths HPG	1170	1532	2832	5821	9443	14555
	Time CE (sec)	2	11	95	2688	1472(*)	–
	Time HPG (sec)	0	4	19	61	111	261
	Congestion gap ϵ_τ (%)	0	0	0	0	–	–
	Inconvenience gap κ_τ (%)	0	0.1	0.4	0.5	–	–

Table 2. Results for different number of nodes in the network

Number of nodes	Stats	τ 0%	5%	10%	15%	20%	25%
240	Paths CE	1170	19084	257488	2201324	–	–
	Paths HPG	1170	2025	2653	8363	13929	16770
	Time CE (sec)	2	21	771	1408(*)	–	–
	Time HPG (sec)	0	15	16	109	275	374
	Congestion gap ϵ_τ (%)	0	0	0	–	–	–
	Inconvenience gap κ_τ (%)	0	0.2	0.4	–	–	–
270	Paths CE	1170	26820	449250	4588806	–	–
	Paths HPG	1170	1728	2093	8479	10119	9507
	Time CE (sec)	3	37	1892	3121(*)	–	–
	Time HPG (sec)	0	8	13	119	182	167
	Congestion gap ϵ_τ (%)	0	0	0	–	–	–
	Inconvenience gap κ_τ (%)	0	0.1	0.3	–	–	–
300	Paths CE	1170	37475	777428	9649118	–	–
	Paths HPG	1170	1554	2060	6215	9855	8967
	Time CE (sec)	3	60	5106	7476(*)	–	–
	Time HPG (sec)	0	9	15	92	194	160
	Congestion gap ϵ_τ (%)	0	0	0	–	–	–
	Inconvenience gap κ_τ (%)	0	0.1	0.3	–	–	–
330	Paths CE	1170	54599	1481404	–	–	–
	Paths HPG	1170	1386	2291	5260	13306	13136
	Time CE (sec)	4	93	1640(*)	–	–	–
	Time HPG (sec)	0	4	16	65	350	340
	Congestion gap ϵ_τ (%)	0	0	–	–	–	–
	Inconvenience gap κ_τ (%)	0	0.1	–	–	–	–

Table 3. Results for different number of nodes in the network

	τ							
Time (sec)	0%	5%	10%	15%	20%	25%	30%	35%
HPG	0	3	11	19	27	42	62	90
CE	1	3	11	31	85	207	456	1084

Table 4. Computational time (sec)

	τ							
ϵ_τ (%)	0%	5%	10%	15%	20%	25%	30%	35%
Average	0	0	0	0.51	0.41	0.75	0.79	0.17
Maximum	0	0	0.02	3.26	1.87	3.05	2.94	0.84

Table 5. Average and maximum relative congestion gap

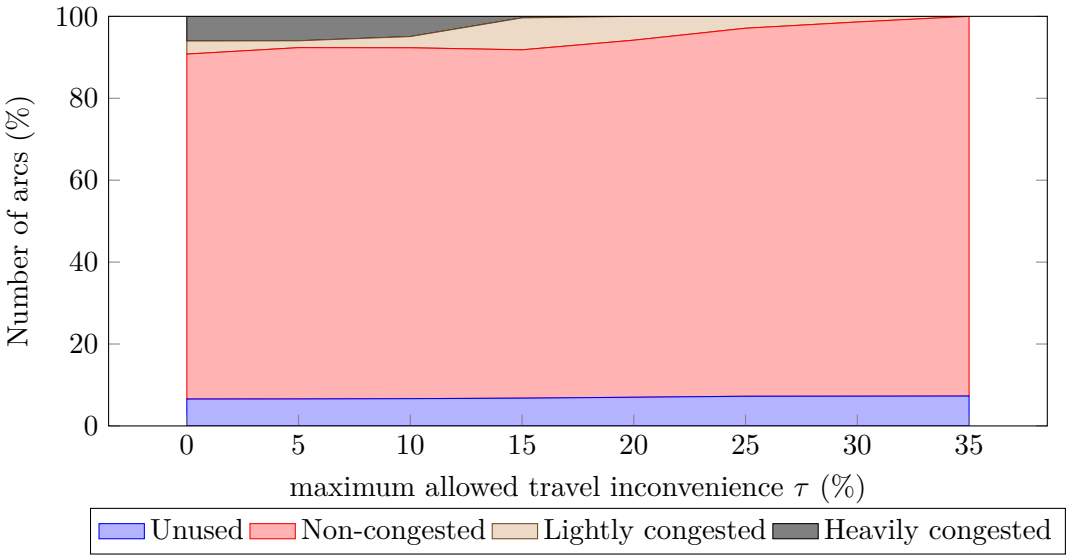
congestion gap over all instances with $\tau = 25\%$). With $\tau = 5\%$ the congestion gap is always 0 and with $\tau = 10\%$ the congestion gap is very low (under 0.5%). When $\tau = 10\%$, the HPG algorithm has strictly positive ϵ_τ only on 5% of the instances, that is on 2 over 40. The HPG algorithm produces $\epsilon_\tau = 0$ with $\tau = 15\%$ on 80% of the instances and the rest of the instances are experiencing ϵ_τ values distributed between 1% and 3%. For higher values of τ the percentage of instances experiencing a congestion gap equal to 0 decreases. For $\tau = 35\%$, 55% of the instances present no congestion gap and all the instances are affected by congestion gaps that are not greater than 1%.

	τ							
ϵ_τ classes (%)	0%	5%	10%	15%	20%	25%	30%	35%
ϵ_τ =0%	100	100	95	80	70	40	45	55
0%< ϵ_τ ≤0.5%	0	0	5	0	5	15	10	35
0.5%<ϵ_τ ≤1%	0	0	0	0	5	15	10	10
1%<ϵ_τ ≤2%	0	0	0	5	20	15	25	0
2%<ϵ_τ ≤3%	0	0	0	10	0	10	10	0
3%<ϵ_τ ≤4%	0	0	0	5	0	5	0	0

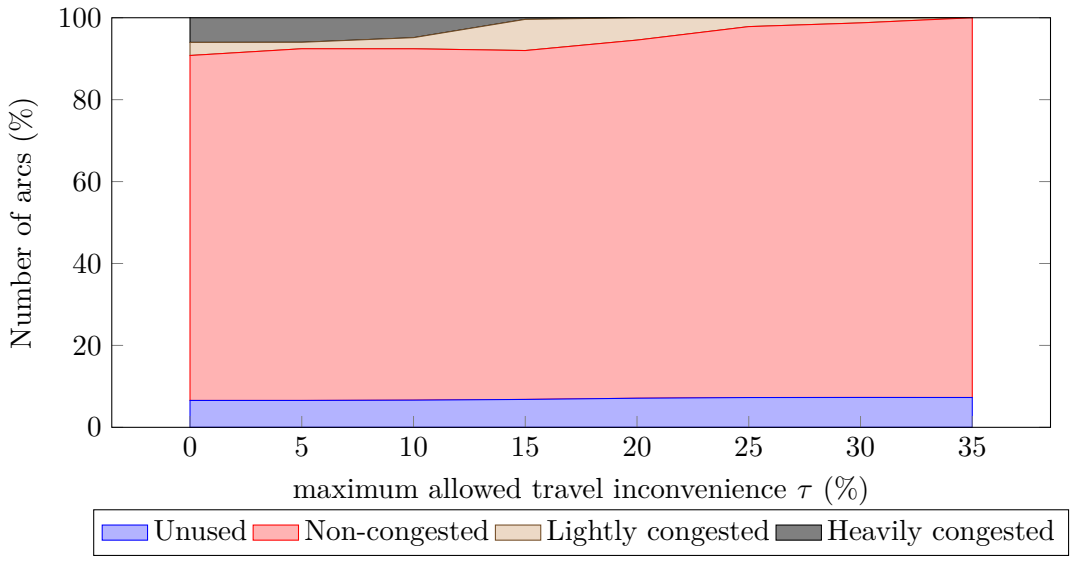
Table 6. Instance experiencing ϵ_τ classes

In Figures 5(a) and 5(b) the distribution of arc congestion level on different congestion classes is shown. Note that this value is averaged over all instances. The two graphics present almost the same behaviour. This means that, on average, the traffic distribution over all arcs is well approximated by the HPG algorithm.

Table 7 shows the behaviour of the absolute inconvenience gap κ_τ of the weighted average travel inconvenience. In the worst case, $\tau = 20\%$, the value of κ_τ is on average less than 0.5%. This means



(a) Arc congestion level distribution with the HPG algorithm



(b) Arc congestion level distribution with the CE

Figure 5. Arc congestion level distribution for the two methods

that users are diverted to a route that is, on average, 0.5% longer than the one assigned by the CE. In terms of maximum, κ_τ is, in the worst case, less than 2.5%. In Table 8 the distribution of κ_τ among different classes is shown. Note that, considering $\tau = 5\%$ and $\tau = 10\%$, all the instances present κ_τ values greater than 0 but always lower than 0.5%. Regarding $\tau = 10\%$, only 10% of the instances have $\kappa_\tau = 0$, but almost all of the remaining instances are experiencing κ_τ values under 1%. For higher values of τ there are no instances experiencing $\kappa_\tau = 0$, but very few instances are experiencing high values of κ_τ. In fact, in the worst case, $\tau = 25\%$ and $\tau = 30\%$, the percentage of instances experiencing κ_τ values greater than 1.5% is only 10%.

	τ							
κ_τ (%)	0%	5%	10%	15%	20%	25%	30%	35%
Average	0	0.04	0.14	0.26	0.50	0.49	0.44	0.44
Maximum	0	0.1	0.43	1.08	1.34	1.93	2.44	2.48

Table 7. Average and maximum absolute inconvenience gap on $\bar{\tau}$

	τ							
κ_τ classes (%)	0%	5%	10%	15%	20%	25%	30%	35%
$\kappa_\tau = 0\%$	100	0	0	10	0	0	0	0
$0\% < \kappa_\tau \leq 0.5\%$	0	100	100	55	55	55	70	70
$0.5\% < \kappa_\tau \leq 1\%$	0	0	0	30	10	25	20	15
$1\% < \kappa_\tau \leq 1.5\%$	0	0	0	5	35	10	0	10
$1.5\% < \kappa_\tau \leq 2\%$	0	0	0	0	0	10	5	0
$2\% < \kappa_\tau \leq 2.5\%$	0	0	0	0	0	0	5	5

Table 8. Instance experiencing κ_τ classes

In order to compare the memory usage of the HPG algorithm with respect to the CE, we report the number of paths generated by the two methods in Table 9. The number of paths generated with the CE rapidly grows with increasing values of τ. Notice that with $\tau = 35\%$ less than 0.3% of paths are generated by the HPG algorithm with respect to the CE.

In Table 9 also the number of iterations of algorithm *SearchForImprovingPaths* is shown. Consider that, at each iteration, a linear programming problem is solved on the restricted set and, on average with $\tau = 35\%$, more than 1200 iterations are performed.

	τ							
Approach	0%	5%	10%	15%	20%	25%	30%	35%
HPG	1170	1449	2212	3129	4090	5397	6881	8375
CE	1170	4055	18536	69449	214936	572058	1355029	2944161
% HPG on CE	100	36	12	5	2	1	0.5	0.3
Iterations	1	149.35	366	607.8	782.5	928	1071	1202.6

Table 9. Number of paths and iterations of the HPG algorithm

5 Conclusions

In this paper we have presented a heuristic for the generation of paths for the proactive route guidance approach, a linear programming based approach that aims at finding a system optimum traffic assignment that takes into account fairness for users by using only paths with limited inconvenience. The computational complexity of the approach is determined by the number of paths that, in the worst case, grows exponentially with the number of nodes of the network. The computational experiments show that the heuristic reduces by orders of magnitude the number of generated paths and the amount of memory usage. The results also show that the quality of the solutions of the proactive route guidance approach obtained using only the heuristic set of paths is very close to that of the optimal ones. Although the heuristic has been designed to generate a set of paths that allow us to find high quality solutions of the proactive route guidance approach, it may be useful for the generation of paths required by other approaches and models for traffic assignment.

References

A. Angelelli, I. Arsik, V. Morandi, M. Savelsbergh, and M.G. Speranza. Proactive route guidance to avoid congestion. *Transportation Research Part B: Methodological*, 94:1 – 21, 2016. doi: http://dx.doi.org/10.1016/j.trb.2016.08.015.

J. Azevedo, M.E. O Santos Costa, J.J.S. Madeira, and E. Martins. An algorithm for the ranking of shortest paths. *European Journal of Operational Research*, 69:97–106, 1993.

K. Bartlett, J. Lee, S. Ahmed, G. Nemhauser, J. Sokol, and B. Na. Congestion-aware dynamic routing in automated material handling systems. *Computers & Industrial Engineering*, 70:176–182, 2014.

S. Bekhor and C. Prato. Effects of choice set composition in route choice modeling. In *Proceedings of the 11th Conference of the International Association for Travel Behavior Research*, pages 16–20. Kyoto Japan, 2006.

S. Bekhor, T. Toledo, and J.N. Prashker. Implementation issues of route choice models in path-based algorithms. In *11th international conference on travel behaviour research, Kyoto, Japan*, 2006.

T. de la Barra, B. Perez, and J. Anez. Multidimensional path search and assignment. In *PTRC Summer Annual Meeting, 21st, 1993, University of Manchester, United Kingdom*, 1993.

E. Frejinger and M. Bierlaire. Capturing correlation with subnetworks in route choice models. *Transportation Research Part B: Methodological*, 41:363–378, 2007.

O. Jahn, R. H. Möhring, A. S. Schulz, and N. Stier-Moses. System-optimal routing of traffic flows with user constraints in networks with congestion. *Operations Research*, 53:600–616, 2005.

M. Kaspi and J. M. A. Tanchoco. Optimal flow path design of unidirectional agv systems. *The International Journal of Production Research*, 28:1023–1030, 1990.

M. Lujak, S. Giordani, and S. Ossowski. Route guidance: Bridging system and user optimization in traffic assignment. *Neurocomputing*, 151:449–460, 2015.

H. S. Mahmassani and S. Peeta. Network performance under system optimal and user equilibrium dynamic assignments: implications for advanced traveler information systems. *Transportation Research Record*, 1993.

D. Park and L. Rilett. Identifying multiple and reasonable paths in transportation networks: A heuristic approach. *Transportation Research Record: Journal of the Transportation Research Board*, pages 31–37, 1997.

C. Prato. Route choice modeling: past, present and future research directions. *Journal of Choice Modelling*, 2:65–100, 2009.

C. Prato and S. Bekhor. Applying branch-and-bound technique to route choice set generation. *Transportation Research Record: Journal of the Transportation Research Board*, pages 19–28, 2006.

C. Prato and S. Bekhor. Modeling route choice behavior: how relevant is the composition of choice set? *Transportation Research Record: Journal of the Transportation Research Board*, 2007.

M. S. Ramming. *Network knowledge and route choice*. PhD thesis, Massachusetts Institute of Technology, 2001.

T. Roughgarden and É. Tardos. How bad is selfish routing? *Journal of the ACM (JACM)*, 49, 2002.

N.J. Van der Zijpp and S.F. Catalano. Path enumeration by finding the constrained k-shortest paths. *Transportation Research Part B: Methodological*, 39:545–563, 2005.

J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.