Christoph Hansknecht and
Sebastian Stiller

Heuristics for a collaborative routing problem

PREPRINT REIHE MATHEMATIK
Institut für Mathematik, Technische Universität Berlin
ISSN 2197-8085

Preprint Nr. 33-2015

http://www3.math.tu-berlin.de/preprints/files/Preprint-33-2015.pdf

Heuristics for a collaborative routing problem

Christoph Hansknecht

Sebastian Stiller

TU Berlin

December 11, 2015

Abstract

We study the problem of computing socially optimal routes with respect to a game-theoretic dynamic flow model. We consider different algorithms to heuristically solve the problem and compare their performance.

1 Introduction

Routing problems are among the classic problems studied in combinatorial optimization. In their simplest form a routing problem is characterized as follows: Given a directed graph $D = (V, A)$, a *source* s and a *sink* t in V, and a weight function $c : A \mapsto \mathbb{R}_{\geq 0}$, what is the shortest path from s and t with respect to c? The problem was considered and solved by Dijkstra in [Dij59]. Several improvements have been made regarding the computational efficiency in situations where for a given graph D and cost c requests between multiple source/sink pairs s_i, t_i are made successively, for a survey see [WW07]. However, this simple routing model is insufficient for many real-world applications due to the following shortcomings:

Firstly travel times throughout networks are not generally constant but rather time-dependent: Consider for example a public transport line which is serviced by vehicles in regular intervals. In this case the travel time depends on the time at which a traveler arrives at a certain stop of the line. Therefore in this case the routing problem consists of a directed graph $D = (V, A)$, time-dependent weight functions $c : A \times \mathbb{N} \to \mathbb{R}_{\geq 0}$ for the arcs $a \in A$, a pair $s, t \in V$ and a departure time $t_d \in \mathbb{N}$. The task is again to find a shortest s-t-path with respect to c. The problem becomes NP-hard in this case, however, there are some important special cases which remain tractable. Firstly, if the weights c obeys the FiFo (*first in first out*) principle a shortest path can be computed using a variant of the algorithm proposed by Dijkstra. Secondly, if the traveler is permitted to wait at intermediate nodes in the graph it is possible to transform the problem to an equivalent problem which weights c' obeying the FiFo principle which can then be solved efficiently.

The problem becomes more complex when we consider examples such as road networks. In the best case the time taken to traverse a road segment will only

depend on the length of the segment and the imposed speed limit. This holds true for off-peak times. However, during rush-hour time the travel times will increase due to congestion. Therefore the travel times are most certainly time-dependent. But even a time-dependent weights are not sufficiently accurate to model the travel times for road-networks. This is due to the fact that travelers traversing the same road segment at the same time influence each other. As a result the travel time for each individual traveler does not only depend on their path but also on the paths taken by every other traveler in the network.

2 Related literature

Ford and Fulkerson introduce dynamic flows in [FF58]. They study the problem of sending a maximum amount of flow from a given source to a given sink within a fixed time horizon T. The problem can be reduced to a static flow problem which can be solved efficiently. An alternative to sending the maximum amount of flow from source to sink within a given time horizon it is sometimes desired to maximized the amount of flow sent through the network at every single point in time. The resulting *earliest arrival flows* can be computed efficiently, for a survey see [Sku09].
The first theoretical investigation of road networks is due to Wardrop in [War52] who introduced the *Wardrop equilibirum*, an equilibrium concept similar to a that of a (mixed) Nash equilibrium. It was remarked that equilibria often suffer from inefficiency with respect to a common social welfare. The inefficiency can be quantified using the so-called *Price of Anarchy*, a term introduced in [KP99]. The Price of Anarchy for road networks utilizing the Wardrop model has been studied in [RT02].

3 Preliminaries

We assume throughout the rest of this paper that we are given a directed graph $D = (V, A)$. A (static) *path* $P = (a_1, \dots, a_l)$ in D is a tuple of arcs $a_i \in A$ where for $a_i = (u, v)$ it holds that $a_{i+1} = (v, w)$. We make the assumption that the path is simple in the sense that no node is visitited more than once. The source $s(P)$ of P is v such that $a_1 = (v, v')$, the target $t(P)$ is defined analogously. We denote by $A(P)$ the arcs contained in the path P and we let $\mathcal{P}$ be the sets of all paths in D. For given nodes $s, t \in V$ we define $\mathcal{P}_{s,t}$ to be the set of paths with source s and target t.
An instance of a routing problem is given by a set of *demands* $d_i = (s_i, t_i, \hat{t}_i)$ for $i = 1, \dots, k$ where $s_i, t_i \in V$ are the source resp. sink nodes and $\hat{t}_i \in \mathbb{N}$ is the *departure time*. A (static) solution x is a tuple of paths leading from sources to sinks:

$$x = (P \in \mathcal{P}_{s_i,t_i})_{i=1,\dots,k} \tag{1}$$

For a path $P = (a_1, \dots, a_l)$ we let t_P^+, t_P^- be the *entering* resp. *leaving* time of the arcs in P:

$$t_P^+, t_P^- : A(P) \to \mathbb{N} \tag{2}$$

4 The deterministic queuing model

Let us assume that we have a given set $S = (P_1, \ldots, P_k)$ of paths and some departure times $(\hat{t}_P)_{P \in S}$. A *dynamic model* M may be used to derive the travel times of the paths in S:

$$M : (S, \hat{t}) \to (t_P^+, t_P^-)_{P \in S} \tag{3}$$

A variety of dynamic models has been proposed to model traffic flows in road networks. The different models are often used to simulate traffic flows observed in certain areas. Dynamic models are judged based on their ability to accurately model traffic flows observed in the real world. In general there is a tradeoff between the accuracy of the model and its computational tractability. In the following we shall focus on the *deterministic queuing model* as propsed by Gawron [Gaw98]. In this model each arc a has three nonnegative parameters (l_a, o_a, u_a). The value l_a denotes the time to travel across the arc disregarding any congestion arising from any traffic situation. The value on the other hand o_a denotes the number of travelers which can leave the arc a in a fixed time interval. Its value might be dependent on the number of lanes of a given road segment. The last parameter, u_a quantifies the *storage capacity* of the arc a, i.e. the maximum number of travelers which can occupy arc a at any given time. The intuition behind the model is the following: A traveler $i = 1, \ldots, k$ uses some path $P \in \mathcal{P}_{s_i,t_i}$ to travel from his source to his destination. For each arc $a \in P$ the following steps take place:

1. The traveler decides if it is possible to enter arc a, i.e. if the limit imposed by u_a is reached or not.
2. The traveler enters a at time $t \in \mathbb{N}$ and proceeds through the arc arriving at its tip at time $t + l_a$.
3. The traveler enters the queue at the tip of arc a. There might be some travelers in the queue ahead of him. In order to proceed he has to wait for those travelers to leave. At each time step at most o_a travelers can leave the queue, however, a traveler can only leave an arc if it is possible to enter the succeeding arc on his path. After some additional waiting time traveler i can enter the next arc a' in P.

It is easily possible to apply the deterministic queuing model to the tuple $(S, \hat{t})$ by stepping through the time from past to present. The details are shown in Algorithm 1.

Remark 1.
- It is possible that travelers can't immediately start along their path due to the capacity constraints. In this case we allow the traveler to wait at the source node.
- It is possible for a deadlock to occur in the model. Consider the situation depicted in Figure 1: If we let $l_a, o_a, u_a = 1$ for all arcs $a \in A$ and all travelers start at $t = 0$ along paths leading from node i to node $i + 2 \mod 3$ then it is not possible for any traveler to continue to the target node. Deadlocks can be detected during the execution of Algorithm 1.

Algorithm 1 Applying the deterministic queuing model

Require: A set S of paths, corresponding departure times $(\hat{t}_i)_{i \in S}$
Ensure: A set of travel times $(t_P^+, t_P^-)_{P \in S}$

for $P \in S$ **do**
    Mark P to be considered at $\hat{t}_P$
end for
while There is some $P \in S$ to be considered **do**
    $P \leftarrow$ next traveler to be considered
    $t \leftarrow$ current time
    if P is on top of the queue of an arc a or P is at their source **then**
        if P is at their destination **then**
            Remove P from the queue a
            $t_P^-(a) \leftarrow t$
        else
            $a' \leftarrow$ next arc on the path P
            if P can enter a' **then**
                Remove P from the queue at a
                $t_P^-(a) \leftarrow t$
                Mark P to be reconsidered at $t + l'_a$
            else
                Mark P to be reconsidered at $t + 1$
            end if
            if The queue at a' is not empty but no longer contains P **then**
                $P' \leftarrow$ the traveler on top of the queue
                Mark P' to be considered at t
            end if
        end if
    else if P is about to enter the queue at arc a **then**
        Add P to at the tail of the queue
        $t' \leftarrow$ time at which P leaves the queue
        Mark P to be reconsidered at t'
        $t_P^+(a) \leftarrow t$
    end if
end while

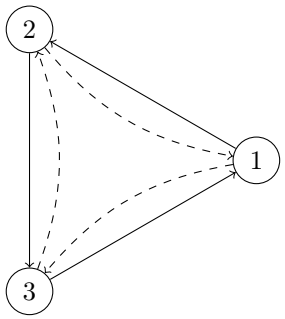

Figure 1: Demands along a cycle may lead to a deadlock. Arcs are drawn solid, demands dashed.

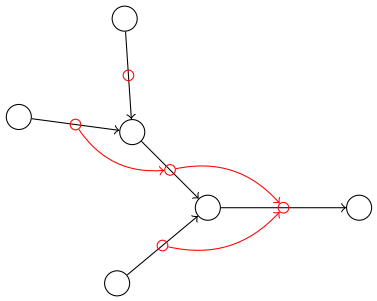

Figure 2: Pairs of blocking arcs (red) as part of the line graph of a given graph.

- The deterministic queuing model satisfies the FiFo principle, that is a traveler entering an arc earlier than an other traveler is guaranteed to leave the arc no later than that traveler.
- The deterministic queuing model can be seen from a game-theoretic standpoint: The travelers are players in a game, traveler i has as strategies the paths in P_{s_i,t_i}. The cost for player i according to the resulting travel times $(T(P_i))_{i=1,\dots,k}$ can be set to the travel time $T(P_i)(t(P_i)) - \hat{t}_i$. Due to the the fact that the FiFo principle is satisfied the existence of a Nash equilibrium is guaranteed. However, the inefficiency given by the Price of Anarchy (PoA) with respect to the total travel time may be unbounded in this case.
- In general there might be ties between travelers arriving at queues at the same time. These ties can be broken using the ordering of the travelers.

Algorithm 1 may be used to apply the deterministic queuing model to a given set of paths. However, the algorithm will not generally complete in polynomial time. This is due to the fact that for a traveler on top of a queue may have to be considered at every time step in succession until it is possible for them to enter the next arc.
In order to improve the running time to a polynomial in the input it is necessary to store information about *blocking arcs* of the graph D. We say that an arc a is *blocked* by a successor a' of a at time $t \in N$ if the traveler on top of the queue of a can't enter a' due to the capacity constraints imposed on a'. At every time step t pairs of blocking arcs form an induced subgraph $B(t)$ of the directed line graph of A (see Figure 4). We assume that $B(t)$ is acyclic, otherwise the graph would be deadlocked at t. Clearly travelers on top of queues of a blocked arc a don't have to be considered until a time $t' > t$ where a is a sink of $B(t')$.
It is possible to modify Algorithm 1 to maintain the blocking graph. As a result the following holds:

Theorem 1. *The deterministic queuing model can be applied to a set of k paths on a digraph with m arcs in $\mathcal{O}(mk \log k)$*

Proof. We modify Algorithm 1 by lazily maintaining blocking arcs: If we find that a traveler attempts to enter an arc which is at capacity we add the corresponding pair of blocking arcs, we remove blocking arcs once the capacity decreases. Thus in every step one of the following actions are performed:

1. A traveler enters the queue at the tip of some arc. In total this can happen at most mk times.

2. A traveler leaves the queue to either enter the succeeding arc or to complete their journey. This might trigger the removal of blocking arcs throughout the network. However, each time a blocking arc is removed there is at least one traveler who can continue his journey. Thus, in total at most mk blocking arcs are removed.

3. A traveler attempts to enter an arc but fails. At this point a blocking arc is added. Since the number of added blocking arcs is equal to the number of removed blocking arcs there will be at most mk events of this type.

The operations performed in the loop are of constant complexity so the total cost in terms of running time is given by the costs caused by finding the next event to be considered. Since at any time during the algorithm there are at most k events to be considered a priority queue can be used to reduce the required running time to $\mathcal{O}(\log(k))$. In total the running time is in $\mathcal{O}(mk \log k)$ as claimed.

□

5 Algorithms

Let us assume that a central authority is informed about the demands $(d_i)_{i=1,\dots,k}$. Such an authority might aim at providing a set of paths $(P_i)_{i=1,\dots,k}$ such that the total travel time (given by (4)) is minimized.

$$T_{\text{tot}} := \sum_{i=1}^{k} T(P_i)(t(P_i)) - \hat{t}_i \tag{4}$$

Unfortunately the resulting optimization problem is NP-hard in the case of the deterministic queuing model. In particular it is already hard to find add an optimal path for a single traveler to an existing set. This is due to the fact that decisions made at an early point in time may affect different travelers at a much later point in time in a complex fashion. In the following we will consider various heuristics yielding sets of paths and compare their performance.
If we are given a set S of paths together with their travel times $(T_P)_{P \in S}$ it is possible to obtain an estimate regarding the time which an additional traveler would need to traverse a single arc a: For each path $P \in S$ there is a (possibly empty) interval I_P in time at which the traveler corresponding to P is waiting in the queue Q_a associated with a. If a new traveler arrives at a point t in time there will be a set

$$S_a(t) := \{P \in S \mid t \in I_P\} \tag{5}$$

Thus, the traveler would have to wait at least $\lfloor |S_a(t)|/o_a \rfloor$ units of time in order to be able to reach the target node of a. We can therefore bound the time-dependent travel time from below via

$$c_{S,a}(t) := l_a + \lfloor |S_a(t + l_a)|/o_a \rfloor \tag{6}$$

We can make use of this estimation to derive a simple heuristic (see Algorithm 1). Informally we repeat the following: After having assigned routes to a subset S of the travelers we use the estimation given by (6) to find a route for the next traveler which we then add to the set S.

Algorithm 2 SUCCESSIVE SHORTEST PATH

Require: Demands $(d_i)_{i=1,\dots,k}$
Require: A permutation $\pi \in S(\{1,\dots,k\})$
Ensure: Paths $(P_i)_{i=1,\dots,k}$
1: $S \leftarrow \emptyset$
2: **for** $j = 1,\dots,k$ **do**
3: $\quad i \leftarrow \pi(j)$
4: $\quad P_i \leftarrow \text{SHORTESTPATH}(s_i, t_i, \hat{t}_i, (c_{S,a})_{a\in A})$
5: $\quad S \leftarrow S + P_i$
6: **end for**

Since the paths $(P_i)_{i=1,\dots,k}$ depend on the order π in which the demands are considered the heuristic can be improved by considering multiple randomly chosen permutations $\pi_1,\dots,\pi_l$ and chose the set of paths which provide the lowest total travel time. As an alternative we can attempt to improve the paths simultaneously yielding Algorithm 3.

Algorithm 3 DYNAMIC FRANK-WOLFE

Require: Demands $(d_i)_{i=1,\dots,k}$
Require: A number of iterations l
Ensure: Paths $(P_i)_{i=1,\dots,k}$
1: $S \leftarrow \emptyset$
2: **for** $j = 1,\dots,l$ **do**
3: $\quad$ **for** $i = 1,\dots,k$ **do**
4: $\quad\quad P_i \leftarrow \text{SHORTESTPATH}(s_i, t_i, \hat{t}_i, (c_{S,a})_{a\in A})$
5: $\quad$ **end for**
6: $\quad S \leftarrow \{P_i \mid i = 1,\dots,k\}$
7: $\quad$ Determine a new lower bound $(c_{S,a})_{a\in A}$ using Algorithm 1
8: **end for**

6 Instances

For our experiments we used a digraph of the Berlin region, which was derived from data gathered by the OpenStreetMap contributors, see `www.openstreetmap.org`. The digraph has $|V| = 36,308$ nodes and $|A| = 88,199$ arcs. The parameters required for the model can be obtained from the *tags* which are associated with the *ways* stored in the raw OSM data. The physical length of an arc a divided by the imposed speed limit yields a reasonable value for the length l_a. The outflow o_a of a can be assumed to be given by the number of lanes of the way corresponding to a. The storage capacity is generally determined from the previous two attributes:

$$u_a = (\text{physical length of } a \times \#\text{lanes of } a)/\text{length occupied by a car} \quad (7)$$

where the length occupied by a car is assumed to be roughly 7.5 meters.
It is on the other hand quite difficult to gather data related to the real-world demands d_i. Since the origins and destinations of travelers are hard to determine explicitly several algorithms haven been proposed to derive O-D matrices from observed traffic flows (for a summary see [Wil78]). However, to derive O-D matrices it is assumed that the system is in a steady state and effectively time-independent. Since we assume a dynamic model these methods are not applicable. We resorted to randomly generate demands. We considered two different scenarios:

1. On the one hand Berlin is split into various districts which have geometric boundaries corresponding to subsets of the nodes V of the Berlin digraph. For most (92 of 98) of these districts there is some statistical data available, including the gross population and the imposed business taxes. Based on this data it is possible to derive a simple model: We assume that demands are due to people traveling to and from work within the Berlin area. The origins of the demands are randomly distributed among the *district* node sets with a weight proportional to the district population. Within these sets we assume a uniform distribution of nodes. For the destinations we merely change the weight of the districts to be proportional to the imposed business taxes.

2. As an alternative we consider outgoing traffic, i.e. travelers leaving a central district traveling towards the periphery of the city.

We further assume that the travel times are normally distributed. In this case the standard deviation of the distribution is highly significant: For a large deviation the travelers are less likely to meet at the same arc at the same time. Therefore there will be fewer interdependence between the travelers. To show this difference we choose a *narrow* normal distribution with a standard deviation of $\sigma_{\text{narrow}} = 5$ minutes and a *wide* one with $\sigma_{\text{wide}} = 1$ hour.

7 Results

In the following we compare the resulting total travel times as determined by various algorithms. We assume that without a central authority the travelers choose their routes independently from each other. Also, the travelers don't have any information regarding the other travelers' origins, destinations or travel times. However, due to the availability of GPS-assisted route planning software we assume that the the travelers do follow shortest paths. Without any information regarding the actual traffic situation it is reasonable to assume that these shortest paths are chosen with respect to the lengths $(l_a)_{a\in A}$. The actual outcome will of course have and increased total travel time with respect to the deterministic queuing model.
It is apparent from Table 1 that the estimation by the travelers is generally much lower than the actually experienced travel time which is the case even for small instances. This is due to the fact that the deterministic queuing model only lets travelers switch to new arcs at fixed time intervals. However, the previously introduced heuristics offer a notable improvement upon the independently chosen routes, a fact which is apparent from Table 2: The total travel time T_{tot}

Instance	**Estimated $T_{\mathbf{tot}}$**	**Actual $T_{\mathbf{tot}}$**
inner-wide-50	57,886	93,149
inner-narrow-50	56,687	92,561
outgoing-narrow-50	63,848	108,346
outgoing-wide-50	61,212	103,560
inner-wide-500	531,376	879,703
inner-narrow-500	548,544	891,001
outgoing-narrow-500	604,928	1,362,134
outgoing-wide-500	602,821	2,173,771

Table 1: Estimated vs actual values of T_{tot}

Instance	Naive value	Successive shortest path	Dynamic Frank-Wolfe
inner-wide-50	93,149	90,179	90,559
inner-narrow-50	92,561	88,941	89,271
outgoing-narrow-50	108,346	106,236	106,496
outgoing-wide-50	103,560	102,110	102,480
inner-wide-500	879,703	872,473	841,093
inner-narrow-500	891,001	861,471	865,191
outgoing-narrow-500	1,362,134	1,214,734	1,226,114
outgoing-wide-500	2,173,771	1,307,081	1,156,511

Table 2: T_{tot} computed by various algorithms

is reduced significantly across the instances, the effect increases as the number of demands grows.

7.1 Algorithmic parameters

In the following we show several attempts at improving the introduced algorithms with respect to their running time. The largest portion of computation time within Algorithm 2 is spent applying the deterministic queuing model using Algorithm 1. In order to avoid a portion of the computation it is possible to recompute the dynamic paths only every $l > 1$ iterations. It is on the other hand possible to vary the number of permutations taken into consideration.
Figure 3 shows the computation time and travel time for various combinations of recomputation frequencies and numbers of permutations. It is obvious that increasing the number of permutations does not lead to an improvement with respect to T_{tot} and merely increases the computation time.

References

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

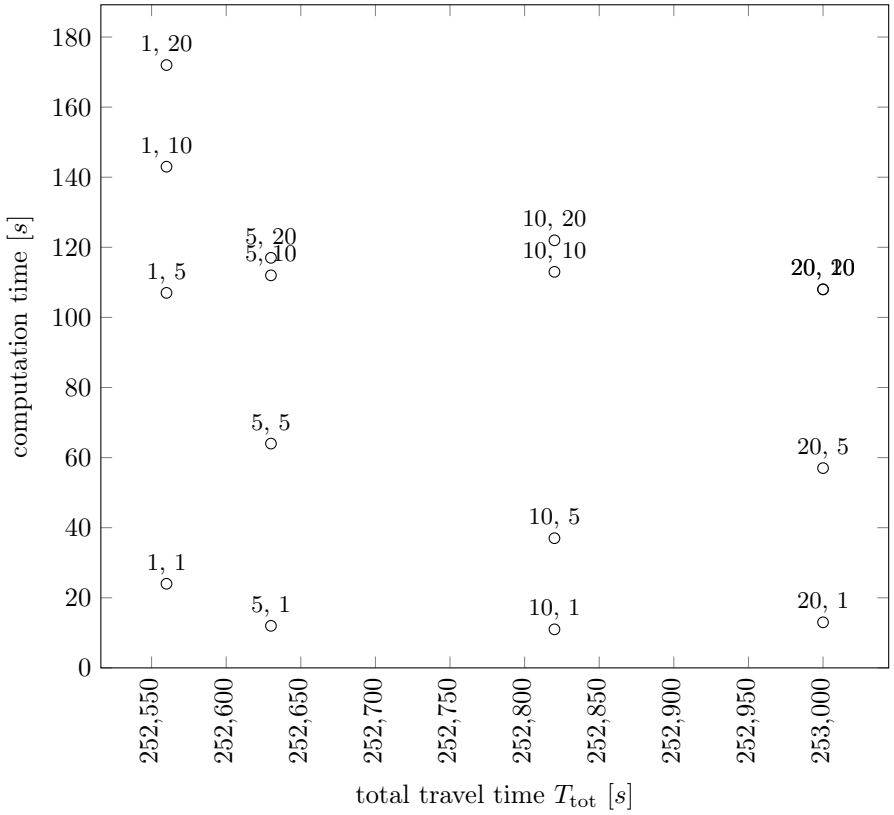

Figure 3: The total travel time vs computation time for the instance *bottleneck-100*, the labels denote the recomputation frequency and the number of permutations

[FF58] L. R. Ford and D. R. Fulkerson. Constructing Maximal Dynamic Flows from Static Flows. *Operations Research*, 6(3):419–433, 1958.

[Gaw98] Christian Gawron. An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model. *International Journal of Modern Physics C*, 9(3):393–407, 1998.

[KP99] Elias Koutsoupias and Christos Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th Annual Conference on Theoretical Aspects of Computer Science*, STACS'99, pages 404–413, Berlin, Heidelberg, 1999. Springer-Verlag.

[RT02] Tim Roughgarden and Éva Tardos. How bad is selfish routing? *J. ACM*, 49(2):236–259, March 2002.

[Sku09] Martin Skutella. An introduction to network flows over time. In William Cook, László Lovász, and Jens Vygen, editors, *Research Trends in Combinatorial Optimization*, pages 451–482. Springer Berlin Heidelberg, 2009.

[War52] J. Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers, Part II*, 1(36):352–362, 1952.

[Wil78] L G Willumsen. Estimation of an o-d matrix from traffic counts – a review, 1978.

[WW07] Dorothea Wagner and Thomas Willhalm. Speed-up techniques for shortest-path computations. In *Proceedings Of The 24th International Symposium on Theoretical Aspects of Computer Science (STACS'07)*, pages 23–36. Springer, 2007.