Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

SIMPLE AND FAST TRIP GENERATION FOR LARGE SCALE TRAFFIC SIMULATION

Takashi Imamichi
Rudy Raymond

IBM Research – Tokyo
NBF Toyosu Canal Front
6-52, Toyosu 5-chome
Koto-ku, Tokyo 135-8511, JAPAN

ABSTRACT

A large-scale traffic simulator with microscopic model requires trip generation for millions of vehicles. To achieve a realistic result, the trip generation should provide a variety of trips between pairs of locations from Origin-Destination (OD) table reflecting the choices of drivers. Shortest paths take long time to generate and often differ from the choices of drivers. We propose a simple and fast tree-based algorithm in this paper. Our algorithm mixes shortest path trees starting from some location nodes in each subarea of the OD table as preprocessing and then generates trips by probabilistically traversing the mixed shortest path trees. Experiments reveal that the tree-based algorithm runs much faster than the naive one. We also confirm that, under certain conditions on the granularity of the OD table, the results of simulation using trips generated by our algorithm do not differ much from traffic conditions observed in the real world.

1 INTRODUCTION

There are many situations when transportation authorities must rely on traffic simulation to evaluate the effectiveness of their action to keep the transportation system functioning properly, for example, when deciding which sections of a road during a natural disaster should be close to avoid congestion around key facilities (hospitals, police stations, or fire stations), where to construct new roads for reducing congestion and CO_2 emission, anticipating the future traffic flows, and so on.

Depending on the granularity, there are roughly two models of traffic simulation: macroscopic and microscopic (Ni 2010; Jeannotte et al. 2004). A macroscopic model can track essential features of traffic flows, such as, average speed and density of a road. On the other hand, a microscopic model allows more detailed analysis and is often considered a better choice for action evaluation (Jeannotte et al. 2004). For example, Nishi et al. (2009) reported that a small change in the configuration of merging lanes could significantly reduce congestion, but impacts from such small changes cannot be evaluated easily from a macroscopic one. Also, we can easily study the impact of traffic controls, ambulances, buses routes and other special vehicles in more detail by a microscopic one. For this reason, there has been a significant amount of efforts in building traffic simulators with microscopic models (Gomes et al. 2004; Behrisch et al. 2011) with agent-based simulation as a key technique.

Agent-based simulation of traffic flows tracks the location of each agent, representing a driver of a vehicle. Each agent is assigned a trip, that consists of an origin point, a destination point, a route connecting the points according to several models of route choices. The agent then travels from a given starting time along the selected route while interacting with other agents according to several types of predefined actions, such as, changing driving speed, selecting lanes, avoiding traffic jams, and so on.

A major challenge in microscopic models is reducing the computational cost of generating trips for millions of such agents. The trips are usually based on the solutions of the shortest path problem with utility functions such as travel distances, driving time, and so on, where Dijkstra algorithm (Dijkstra 1959)

or A* algorithm (Hart et al. 1968) can be utilized. Due to the limitation of preferring shorter trips, the number of varieties in trips is small. Moreover, when used in the simulation, unrealistic traffic congestion, where too many agents try to pass the same roads shared by shortest trips, are often observed. This also hints that real-world drivers might not necessarily prefer shortest trips.

Another challenge is how to generate trips from coarse OD tables obtained from limited amount of data sources. The OD tables often only contain number of trips from a (large) subarea to another, instead of point-to-point traffic flows for the trip generation. In this case, it is not clear how to find short but realistic trips between points in those large areas. The widely accepted intuition is that the trips should be close to the shortest ones but contain varieties reflecting drivers' choices in the real world.

We consider two requirements for microscopic traffic simulation models to address the challenges above: (1) to generate a large amount of trips very fast; (2) to generate a variety of realistic trips that are highly correlated to observed ones.

We propose a tree-based algorithm to generate trips satisfying the requirements. The main advantages of the tree-based algorithm are its simplicity and efficiency. The algorithm is derived by modifying the Dijkstra's algorithms to utilize the shortest path trees efficiently. A shortest path tree derived by the Dijkstra's algorithm is usually used only once to find an *s-t* shortest path and then discarded. On the other hand, our tree-based algorithm reuses it to obtain a variety of paths around the shortest ones, and therefore is faster. We give comparison of the running time between our proposed algorithm with the naive shortest-path one experimentally.

For comparing the quality of trips generated, we present empirical evaluation in various settings. First, we compare the roads that are obtained from our proposed method against those from the naive one using a real-world OD table in Nairobi city, and an artificial OD table in Rio de Janeiro city. We show that despite its simplicity, the proposed method can generate trips sufficiently close to those from the naive one. Second, we run agent-based traffic simulators in Nairobi city using trips generated from the proposed method and the naive one, and find that the simulation results are not that different from actual reported traffic congestion. These imply that our proposed trip generation can be used for a faster and accurate microscopic simulation model from coarse inputs such as OD tables.

Third, however, we uncover the limitation of trip generation when dealing with trajectory data as inputs using real GPS sequences of taxis in Beijing city provided by T-drive in Yuan et al. (2010) and Yuan et al. (2011). We show evidences that real-world taxi trips might not necessarily follow shortest paths (both our and naive methods produce similar trips with low correlation to observed ones). We also find that when trips start and end at the same subarea, our proposed method can be inferior to the naive one.

As a simulator, we use IBM Mega Traffic Simulator (Osogami et al. 2012), or Megaffic, that can simulate millions of vehicles in an entire city to evaluate lengths of traffic congestion, travel time of vehicles, CO_2 emission, etc. Megaffic builds its simulation model from map data and various input sources, such as, census data and probe-car data. The census data gives OD tables whose entries represent the numbers of trips from a subarea of the map to another during each hour of a day. The probe-car data record the trip histories of multiple drivers of vehicles, where a trip history is a sequence of longitude, latitude, and time observed with the Global-Positioning-Satellite (GPS) devices. Morimura and Kato (2012) developed an algorithm to generate OD tables from such probe-car data.

However, in practice it is usually difficult to obtain sufficient GPS data in many cities. In such cases, Megaffic must rely on a simpler model to generate trips. It divides the city map into rectangular regions corresponding to regions in OD tables. It then assigns the number of vehicles to run between a pair of regions based on the given tables. As an example of such a case, we use data by Gonzales et al. (2009) for Nairobi city. As Kenya's economy develops, so does the traffic demand in Nairobi city. However, the growth of the traffic flows outpaces that of the infrastructure. Consequently, Nairobi city suffers from severe traffic congestion. Since construction of new roads takes time, possible actions to mitigate traffic congestion at the moment are limited to road closures or lane additions. A microscopic simulation model with input from coarse OD tables is the tool to simulate effects of such policies. Our contribution is in

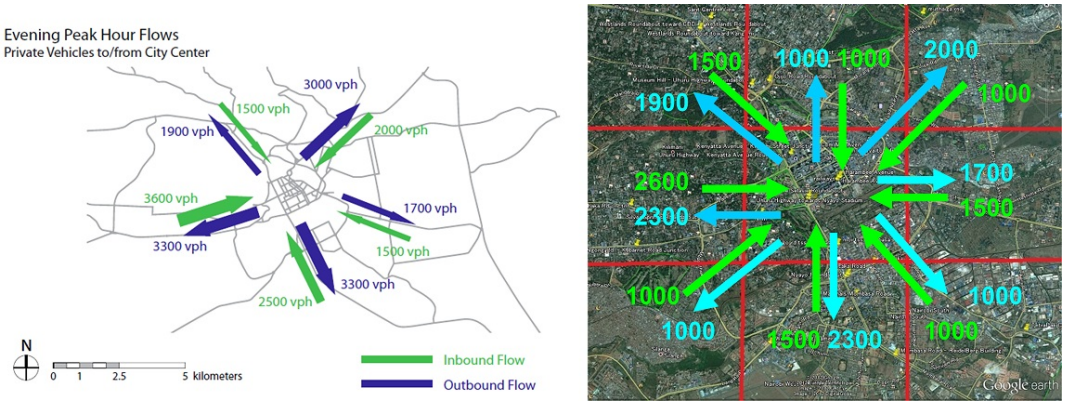


Figure 1: An example of OD table of Nairobi city. The left figure is cited from Figure 6 by Gonzales et al. (2009) and we design an OD table in the right figure. Notice that the map of the Nairobi city is split into 9 rectangular subareas.

providing fast and simple trip generation that can be used as inputs to agent-based simulation to achieve a sufficient level of precision.

1.1 Related Work

The approach to accelerate shortest path queries in generating trips by an efficient preprocessing has been extensively studied such as the contraction hierarchies (Geisberger et al. 2008), the hub labeling (Abraham et al. 2011), the transit node (Bast et al. 2007), and the ALT algorithm (Goldberg and Harrelson 2005). These approaches reduce each shortest path query drastically. But, the algorithms are quite complicated and can be overkill for generating trips from coarse OD tables because shortest path query is for a trip between points and is of different granularity level with OD tables used in the simulation.

Wang and Niedringhaus (1993) presented a distributed and revised Floyd algorithm for routing in a distributed, discrete-event traffic simulation. Since this approach first computes all pairs shortest paths and outputs trips of origin and destination node pairs, it is useful to try various OD tables. But, it may not be applicable to to large networks because the running time is $O(n^3)$ time in the worst case, where n is the number of nodes in a network. Thulasidasan and Eidenbenz (2009) proposed an approach that combined the discrete-event queue model with scalable parallelization. They adopted the Dijkstra's algorithm and A* algorithm for routing trips. They reported that routing trips by A* algorithm takes more than 60% of computation time in a single CPU run.

The rest of the manuscript is organized as follows. We introduce a baseline algorithm based on the Dijkstra's algorithm in Section 2 and then propose a new tree-based algorithm in Section 3. We report the experimental results in Section 4 and make a concluding remark in Section 5.

2 BASELINE ALGORITHM

We introduce notation and a baseline algorithm for the trip generation in this section. A directed graph $G = (N, A)$ is given as a road network, where nodes N and arcs A correspond to roads and cross points, respectively. Let $n = |N|$, $m = |A|$, and $w : A \to \mathbb{R}_+$ be a non-negative weight of arcs representing travel time of roads. The graph is split into k subareas $\{C_i \mid C_i \in N, 1 \leq i \leq k\}$ where the subareas may overlap. Let C_s be a source area, and C_d be a destination area. All the *log*s use the base 2 as default.

The task is to generate a set P of b paths that depart nodes in C_s and arrive at nodes in C_d. The number b corresponds to that in the OD table corresponding the traffic flow between subareas C_s and C_d. The baseline algorithm assumes that drivers will choose the shortest path to drive from an origin node in C_s to a destination node in C_d. This stems from the fact that although the actual paths might not necessarily be

the shortest ones, intuitively they should be around them. In the traffic simulation, each generated path is assigned to each vehicle who will follow the assigned path.

A baseline algorithm to generate shortest paths is composed using the Dijkstra's algorithm directly. It randomly chooses the source and destination nodes and connect them by the Dijkstra's algorithm to create one trip following the shortest paths between them. To generate b trips the baseline algorithm repeats the same operation b times. Note that Thulasidasan and Eidenbenz (2009) adopted a similar algorithm for routing trips. We describe the baseline algorithm formally in Algorithm 1.

Algorithm 1 : Baseline algorithm to generate a set of b trips following the shortest paths connecting from C_s to C_d.

1: $P \leftarrow \emptyset$.
2: **repeat**
3: Choose nodes $u \in C_s$ and $v \in C_t$ randomly.
4: Apply the Dijkstra's algorithm to find the shortest path p between u and v.
5: **if** u and v are connected **then**
6: $P \leftarrow P \cup \{p\}$.
7: **end if**
8: **until** $|P| = b$
9: **return** P.

The time complexity of the baseline algorithm depends on the connectivity of C_s and C_d. It invokes the Dijkstra's algorithm b times at least where the running time of each call of the Dijkstra algorithm is $O(n \log n + m)$ time due to Fredman and Tarjan (1987). As a target road network grows, the simulation for the network is likely to require more vehicles running. Consequently, the running time of the trip generation expands in the order of $\Omega(n \log n + m)$. Suppose $b = O(m)$, then the baseline algorithm takes $O(mn \log n + m^2)$, for example.

The baseline algorithm has several drawbacks. First, since it computes the shortest paths between each pair of origin and destination nodes, the resulting trip set tends to concentrate in several specific arcs (or, roads), for example, if there is a highway between the source and destination subareas. This situation might not arise in the actual traffic condition, but simulation using shortest-path trips can easily results in traffic congestion on such roads. Second, shortest paths between two particular nodes might be overkill because the input OD tables only contain rough estimation of flows between two sets of nodes. Our proposed tree-based method is designed to reduce time complexity of generating paths that are close to the shortest ones and to mitigate the above drawbacks.

In order to reduce the running time of the shortest path queries by preprocessing, we also try ALT algorithm (Goldberg and Harrelson 2005) alternative to the Dijkstra's algorithm (line 4 in Algorithm 1) in our experiments. In the implementation of ALT algorithm we select 16 landmarks using the farthest landmark selection method and use 4 landmarks for the computation of the lower bound of distance. Note that the baseline algorithm outputs same trips with both Dijkstra's algorithm and ALT algorithm.

3 TREE-BASED ALGORITHM

We introduce a simple and fast algorithm to generate trips based on the shortest path trees derived from the Dijkstra's algorithm. The Dijkstra's algorithm first computes the shortest path tree from a source node to other nodes including the destination node. It then traverses the shortest path tree from the destination node to the source node in reverse to construct the shortest path between them. To compute the shortest path between another pair of source and destination nodes, the Dijkstra's algorithm creates a new shortest path tree from scratch.

The key point of our new algorithm is to keep some shortest path trees in advance and find near shortest paths between different pairs of source and destination nodes by traversing the shortest path trees we already

have. It means that we can omit the calculation of the shortest path trees for many pairs of source and destination nodes. Unlike the traverse of a single shortest path tree in Dijkstra's algorithm, there could be more than one candidate nodes when traversing the shortest path trees in our algorithm. Thus, we need to choose a previous node in a probabilistic manner. This corresponds to switching from a shortest path tree to another. We design the tree-based algorithm because we assume that many shortest paths starting from the same subarea use similar arcs many times. Also, intuitively, the switching randomization works to generate various near shortest paths that represents drivers' choices of changing paths to reach a destination (due to new information of traffics and so on).

We first give the outline of the proposed trip generation. To generate a set of paths connecting subareas C_s and C_d we first combine α shortest path trees that departs from C_s as preprocessing, where the roots of the shortest path trees are chosen randomly in C_s and α is a user-defined parameter. Suppose that we apply the Dijkstra's algorithm to a source node $u \in C_s$ and obtain the shortest path tree T_u, let $\mu_{T_u}(v)$ be the parent nodes of a node v in T_u or $\mu_{T_u}(v) = null$ if there is no parent node of v in T_u. We omit T_u of μ_{T_u} in the rest of this paper if there is no confusion. We count the numbers $\lambda_s(u,v)$ of occurrences of arc $(u,v) \in A$ in a set of shortest path trees generated in the preprocessing.

In the main phase of the tree-based algorithm, we generate each trip by traversing the mixture of the shortest path trees in reverse. We have another user-defined parameter β that controls the number of hops of trips in the source subarea. Starting from a randomly chosen node v in the destination subarea C_d, we iteratively choose a previous node $u \in B(v)$ randomly according to the weights $\phi(\lambda_s(u,v))$, $u \in B(v)$, where let $B(v) = \{u \in N \mid (u,v) \in A\}$ be a set of the previous nodes of v, and $\phi : \mathbb{R}_+ \to \mathbb{R}$ be a normalization function for arc occurrences λ_s. We adopt $\phi(x) = \ln(1+x)$ in our experiments because the function is monotonically increasing, and it understates frequent arcs. Note that this normalization function ignores all arcs that do not appear in the shortest path trees because $\phi(0) = 0$. The algorithm repeats this operation until it walks $rand(\lceil \beta \sqrt{|C_s|} \rceil)$ times in C_s, where $rand(x)$ generate a random natural number in a range $[1,x]$. We describe the tree-based algorithm formally in Algorithm 2. Note that the tree-based algorithm behaves exactly the same as the baseline algorithm if $\alpha = 1$, that is, when we use only one shortest path tree to generate trips.

4 EXPERIMENTS

We compare the baseline and tree-based algorithms through computational experiments in this section. We conduct three types of comparison; (1) comparison of the tree-based and baseline algorithms for Nairobi and Rio de Janeiro OD data, (2) comparison of the simulation results by the algorithms and the observed traffic congestion in Nairobi by Gonzales et al. (2009), and (3) comparison of the trips by the algorithms and probe-car data called "T-drive" by Yuan et al. (2010) and Yuan et al. (2011). We implement the baseline and the tree-based algorithm in Python and conduct experiments on a PC with Intel Xeon E5540 CPU (2.53 GHz) and a Python runtime PyPy 2.0 beta 1. Note that we implement the algorithms as single-threaded programs.

We obtain road network data of Nairobi in Kenya, Rio de Janeiro in Brazil, and Beijing in China from OpenStreetMap (http://www.openstreetmap.org). The graph of Nairobi city has 3434 nodes and 5308 arcs. That of Rio de Janeiro city has 20058 nodes and 30015 arcs, and that of Beijing has 9338 nodes and 18047 arcs.

Comparison of the trips generated by the baseline and tree-based algorithms. We first compare the trips generated by the baseline and proposed algorithms for Nairobi and Rio de Janeiro. We design an OD table for Nairobi city based on Gonzales et al. (2009) as depicted in Figure 1. Note that the data is for evening peak hour. Since we do not have such information related to OD tables for Rio de Janeiro city, we design an OD table artificially to realize a condition where people commute to the center of the city from suburbs. Figure 2 shows the OD table we use. Note that the number of trips for Nairobi city is 24300 and that of Rio de Janeiro city is 30000.

Algorithm 2 : Tree-based trip generation algorithm to create a set of b trips connecting from C_s to C_t with parameters (α, β).

{Preprocessing phase}
$\lambda_s(u,v) \leftarrow 0$, for all $(u,v) \in A$.
for $i = 1$ **to** α **do**
  Choose a node $u \in C_s$ randomly.
  Apply the Dijkstra's algorithm to the source node u to obtain the parent nodes μ.
  $\lambda_s(\mu(v),v) \leftarrow \lambda_s(\mu(v),v) + 1$, for all $v \in N$.
end for

{Main phase}
$P \leftarrow \emptyset$.
for $i = 0$ **to** b **do**
  Choose a node $v \in C_t$ with $\mu(v) \neq null$, randomly.
  Let $p = (v)$ be a node sequence containing only v.
  repeat
    {Traverse outside C_s}
    Choose $u \in B(v)$ randomly according to the weights $\phi(\lambda_s(u,v)), u \in B(v)$.
    Append u to p.
    $v \leftarrow u$.
  until $v \in C_s$
  for $j = 1$ **to** $rand(\lceil \beta \sqrt{|C_s|} \rceil)$ **do**
    {Traverse inside C_s}
    Choose $u \in B(v)$ randomly according to the weights $\phi(\lambda_s(u,v)), u \in B(v)$.
    Append u to p.
    $v \leftarrow u$.
  end for
  Reverse the order of p.
  Prune cycles in p.
  $P \leftarrow P \cup \{p\}$.
end for
return P.

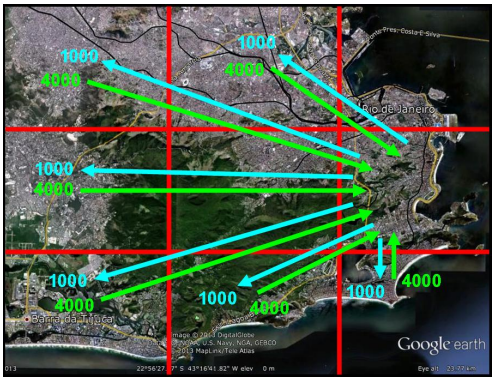


Figure 2: An OD table for Rio de Janeiro city we use for the 1-hour traffic simulation. The numbers denote the numbers of vehicles departing during the 1-hour simulation.

Figure 3: Heat maps of road occurrences in the trips for Nairobi city. The left figure is a result by the baseline algorithm and the right one is a result by the tree-based algorithm.

We try different parameters of the tree-based algorithm; the number α of shortest path trees in each subarea and the distance β in the source subarea. We use ranges $\alpha \in \{10,20,40,60,80,100\}$ and $\beta \in \{1,2,4,8\}$ in the experiments.

After generating trips, we apply IBM Mega Traffic Simulator as the microscopic traffic simulator to the trips. We conduct 1-hour traffic simulation for both Nairobi city and Rio de Janeiro city. We set the trip departure times as follow; 20% of the trips depart at the first second and the departure times of the other trips are chosen within a range of one hour randomly.

Table 1 compares the baseline and tree-based algorithms for Nairobi and Rio de Janeiro data. The columns "α" and "β" are parameters for the tree-based algorithm. The column "T_{pre}" denotes the running time of the preprocessing, "T_{trip}" denotes the running time of the main phase, and "P_{150}" denotes the proportion of trips whose travel time is less than or equal to 150% of that of the shortest path, i.e., $P_{150} = |\{p \in P \mid (d(p) - d^*(p))/d^*(p) \leq 1.5\}|/|P|$, where $d(p)$ denotes the travel time of a path p and $d^*(p)$ denotes the travel time of the shortest path connecting the source and the destination node of p. Let $D_{\text{avg}} = |P|^{-1} \sum_{p \in P} (d(p) - d^*(p))/d^*(p)$ represent the average difference of the travel time. Suppose that P_{baseline} and P_{tree} are the resulting paths by the baseline and tree-based algorithms, respectively, Cor_{trip} is the correlation coefficient of $count(a, P_{\text{baseline}})$ and $count(a, P_{\text{tree}})$ for $a \in A$ where $count(a,P) = |\{p \in P \mid a \in p\}|$. Let $passed(a,P)$ represent the number of vehicles passing a road a in the last 10 minutes of the 1-hour simulation by Megaffic with a trip set P, and Cor_{sim} denote the correlation coefficient of $passed(a, P_{\text{baseline}})$ and $passed(a, P_{\text{tree}})$ for $a \in A$. The last row "baseline" shows the computation time of the baseline algorithm, where "(Dijkstra)" and "(ALT)" show the results with the Dijkstra's algorithm and ALT algorithm, respectively.

We observe that the running time of the tree-based algorithm is shorter than that of the baseline algorithm with any parameter configuration. The trips get closer to the shortest path trips with smaller β. However, too small β worsens the correlation Cor_{trip} of the road occurrences and the correlation Cor_{sim} of the number of vehicles passing roads in the simulation between the baseline and tree-based algorithms. Note that we draw underlines to the best Cor_{sim} values.

We compare the distribution of roads occurred in the resulting trips of the baseline and tree-based algorithms using heat maps for Nairobi city and Rio de Janeiro city in Figure 3 and Figure 4, respectively. We pick up parameters with the best correlation coefficient of the simulation results, that is, $\alpha = 20$, $\beta = 4$ for Nairobi city and $\alpha = 40$, $\beta = 8$ for Rio de Janeiro city. In the heat maps, road with no color means that they do not appear in the trips, green roads rarely appear in the trips, and red roads frequently appear in the trips. More precisely, green color means the number of occurrences is less than 500 and red color means

Table 1: Comparison of the results by the baseline algorithm and those by the tree-based algorithm for Nairobi and Rio de Janeiro data.

tree		Nairobi						Rio de Janeiro					
α	β	T_{pre}	T_{trip}	P_{150}	D_{avg}	Cor_{trip}	Cor_{sim}	T_{pre}	T_{trip}	P_{150}	D_{avg}	Cor_{trip}	Cor_{sim}
10	1	0.87	1.23	0.95	0.09	0.91	0.45	5.02	6.09	0.94	0.11	0.94	0.81
10	2	0.86	1.48	0.93	0.14	0.94	0.58	4.99	6.58	0.94	0.13	0.95	0.85
10	4	0.86	1.92	0.85	0.25	0.95	0.63	4.99	7.54	0.92	0.17	0.95	0.88
10	8	0.85	2.63	0.74	0.39	0.93	0.54	4.99	9.14	0.86	0.24	0.95	0.87
20	1	1.45	1.33	0.95	0.09	0.93	0.46	9.68	5.90	0.93	0.14	0.89	0.77
20	2	1.45	1.50	0.93	0.13	0.96	0.60	9.61	6.73	0.92	0.17	0.90	0.82
20	4	1.45	2.02	0.86	0.23	0.96	0.68	9.64	7.27	0.88	0.22	0.91	0.84
20	8	1.44	2.62	0.77	0.35	0.95	0.54	9.67	9.57	0.81	0.31	0.91	0.85
40	1	2.62	1.35	0.94	0.11	0.92	0.49	18.86	6.83	0.92	0.16	0.92	0.78
40	2	2.62	1.66	0.92	0.15	0.95	0.56	18.87	7.24	0.90	0.19	0.92	0.81
40	4	2.62	1.90	0.84	0.25	0.96	0.65	18.86	8.31	0.86	0.28	0.93	0.87
40	8	2.62	2.85	0.73	0.39	0.95	0.63	18.90	10.07	0.79	0.34	0.94	0.89
60	1	3.79	1.33	0.94	0.11	0.93	0.50	27.92	6.53	0.91	0.16	0.90	0.78
60	2	3.85	1.64	0.91	0.16	0.95	0.54	27.82	7.07	0.90	0.19	0.91	0.82
60	4	3.78	1.94	0.83	0.28	0.95	0.62	27.90	8.11	0.86	0.25	0.91	0.84
60	8	3.78	2.83	0.72	0.40	0.95	0.56	27.88	10.31	0.79	0.36	0.91	0.85
80	1	4.88	1.38	0.93	0.13	0.93	0.51	36.79	6.61	0.91	0.18	0.89	0.78
80	2	4.86	1.57	0.90	0.18	0.95	0.58	36.78	7.22	0.89	0.20	0.90	0.81
80	4	4.84	1.98	0.82	0.28	0.96	0.65	36.64	8.63	0.85	0.26	0.91	0.85
80	8	4.85	2.98	0.71	0.43	0.95	0.66	36.70	9.87	0.79	0.36	0.91	0.84
100	1	6.28	1.39	0.92	0.14	0.93	0.51	45.78	6.63	0.91	0.17	0.90	0.76
100	2	6.01	1.62	0.89	0.19	0.95	0.53	45.60	7.20	0.89	0.20	0.91	0.81
100	4	5.92	2.15	0.82	0.29	0.96	0.60	45.61	8.18	0.85	0.26	0.92	0.86
100	8	5.95	2.76	0.71	0.44	0.95	0.61	45.64	10.30	0.79	0.35	0.92	0.86
baseline (Dijkstra)			65.30						690.56				
(ALT)		0.48	39.23					1.85	515.99				

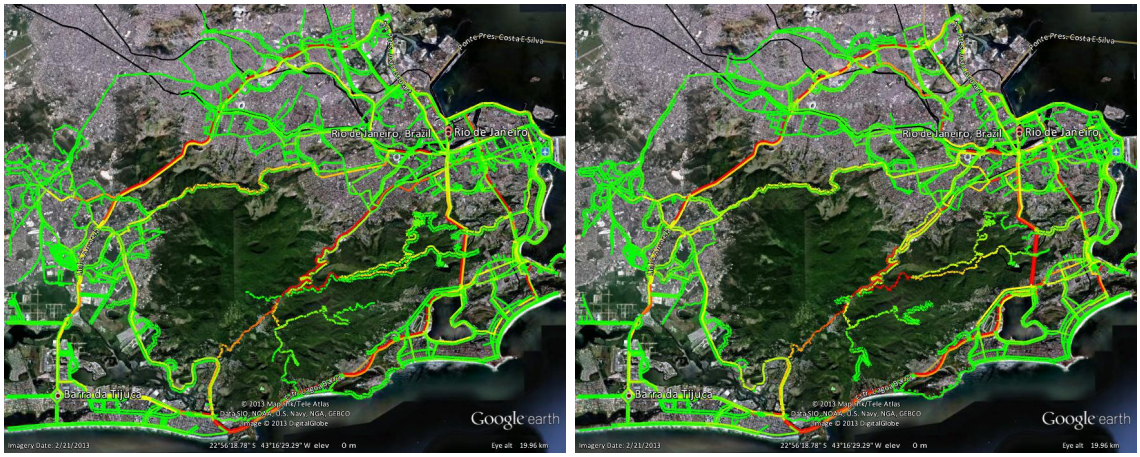


Figure 4: Heat maps of road occurrences in the trips for Rio de Janeiro city. The left figure is a result by the baseline algorithm and the right one is a result by the tree-based algorithm.

Table 2: Correlation coefficient of road occurrences in the probe-car data and the trips by the algorithms.

γ_{hop}	$Cor_{\text{gps,baseline}}$	$Cor_{\text{gps,tree}}$
5	0.601	0.601
10	0.642	0.635
20	0.686	0.665
30	0.673	0.669
40	0.688	0.658

that is more than 3000. We observe that the trips by the tree-based algorithm almost cover the roads by the baseline method and the distribution looks similar. However, the tree-based algorithm cannot choose some roads that appear in the trip by the baseline algorithm.

Comparison of the simulation results and observed traffic congestion in Nairobi. We then compare our simulation results with the actual traffic congestion reported by Gonzales et al. (2009) in Figure 5. The above figure is cited from Figure 14 by Gonzales et al. (2009) that denote the congested roads in morning and evening. The lower left figure and lower right figure in Figure 5 represent the results by the baseline algorithm and the tree-based algorithm with $\alpha = 20$, $\beta = 4$, respectively. We draw colors on roads whose average speed is less than 20 km/h; red color means the average speed is almost zero and the green means the average speed is around 20 km/h. The blue ellipses surround the area where the simulation results match the observed traffic congestion. The accuracy is not different between the baseline algorithm and the tree-based algorithm.

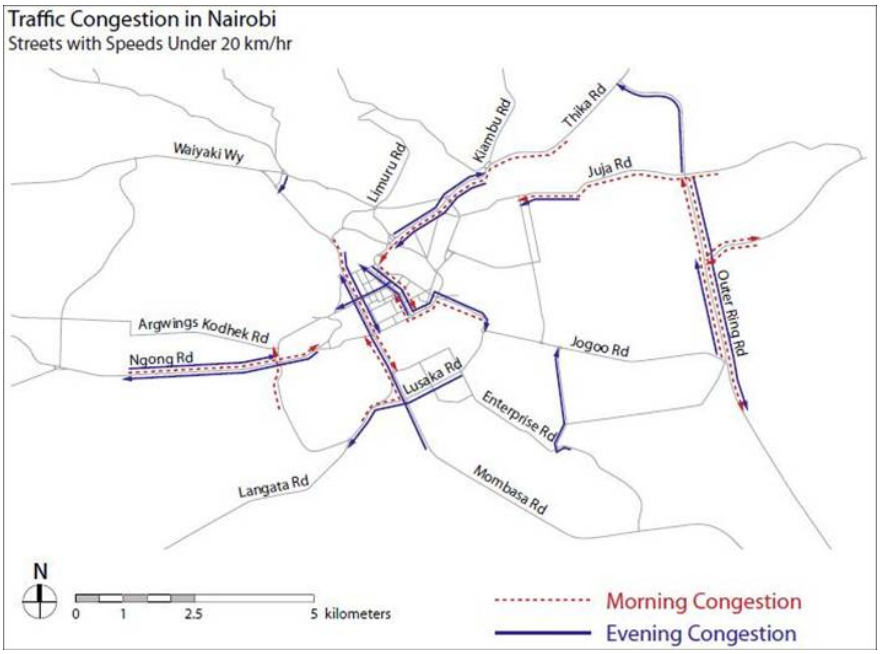
Comparison of the trips by the algorithms and probe-car data. We finally compare the trips generated by the algorithms to probe-car data. We use "T-drive" data by Yuan et al. (2010) and Yuan et al. (2011). T-drive data consists of taxi probe car data in Beijing where the trajectories are represented by sequences of longitudes and latitudes. We apply a map matching algorithm by Raymond et al. (2012) to convert the sequences of coordinates into a sequences of nodes. We pick up a area in Beijing that covers many probe-car data and split it into 5×5 equal-size rectangular subareas. As there are many short trips in T-drive data, we sieve such trips with a threshold γ_{hop} of the minimum number of hops of trips, i.e., we discard trips whose numbers of hops are less than γ_{hop}. We try γ_{hop} in a range $\{5, 10, 20, 30, 40\}$. We set the parameters of the tree-based algorithm by $\alpha = 20$ and $\beta = 8$ in this experiments.

We generate the OD table from the probe-car data by extracting the origin and destination nodes from the trajectories and count the numbers between the subareas. We observe that most of the trajectories in T-drive data run in the same subareas, for example, 2616 trajectories stay in the same subarea among 2955 trajectories with $\gamma_{\text{hop}} = 10$. We compare the road occurrences in the probe-car data and those in the trips by the algorithms in Table 2. Suppose that P_{gps} is the paths in the probe-car data, we let $Cor_{\text{gps,baseline}}$ denote the correlation coefficient of $count(a, P_{\text{gps}})$ and $count(a, P_{\text{baseline}})$ for $a \in A$ and $Cor_{\text{gps,tree}}$ denote the correlation coefficient of $count(a, P_{\text{gps}})$ and $count(a, P_{\text{tree}})$ for $a \in A$.

We observe that both algorithms achieved similar correlation coefficients with the probe-car data. Figure 6 depicts the heat maps of road occurrences in the probe-car data, the trips by the baseline algorithm, and those by the tree-based algorithm with $\gamma_{\text{hop}} = 10$. Note that red means the number of occurrences is more than 50 and green means that is less than 10. We observe that the tree-based algorithm generates more trips than the probe-car data. As the OD table made from the probe-car data is converged locally, the tree-based algorithm cannot utilize the advantages of the shortest path trees across different subareas.

5 CONCLUSIONS

We have presented a simple and efficient tree-based trip generation algorithm in this paper. Instead of computing a shortest-path tree for every pair of origin and destination nodes, the tree-based trip generation uses only a small number of shortest-path trees to generate trips between subareas defined in the OD tables. Because traffic simulation users usually generate a lot of trips from different OD tables to check traffic conditions under several scenarios, the proposed trip generation clearly can help reducing the running time



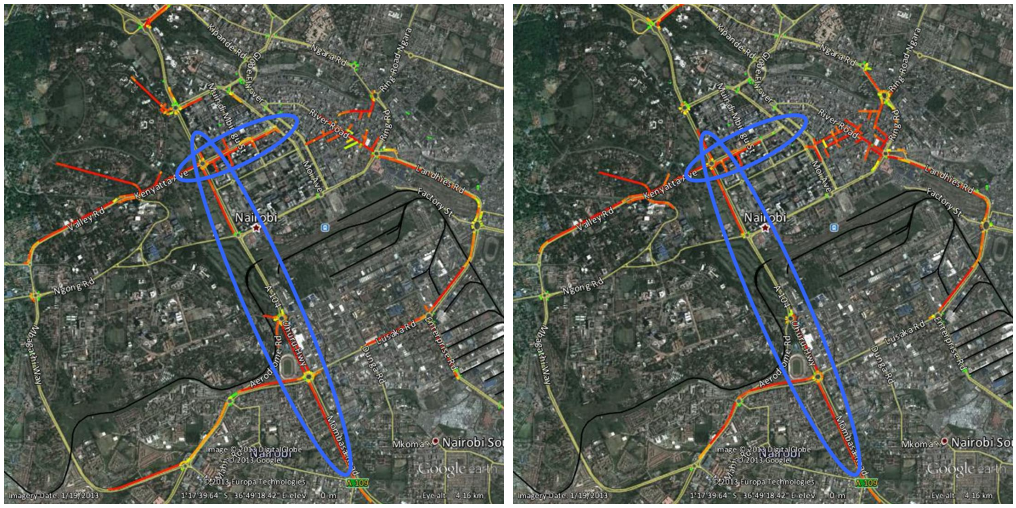


Figure 5: A figure above is cited from Figure 14 by Gonzales et al. (2009). Heat maps below depict average speed of the last 10 minutes in the 1-hour simulation for Nairobi city. The left figure is a result by the baseline algorithm and the right one is a result by the tree-based algorithm. The blue ellipses point the area where the simulation results match the observed traffic congestion.

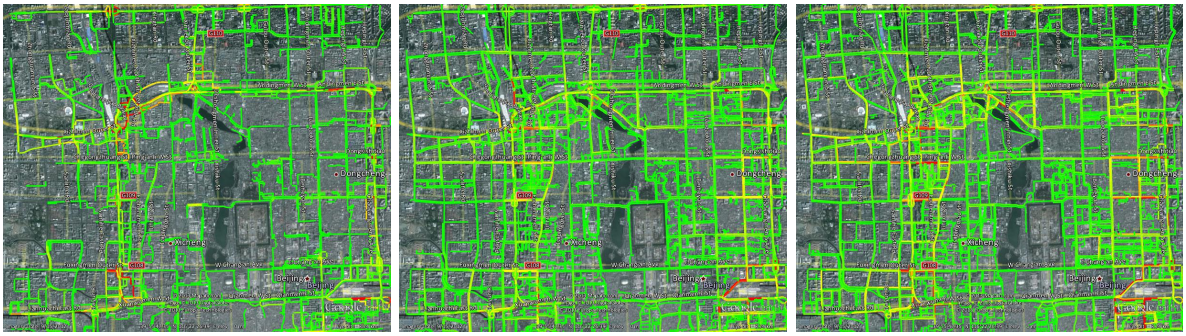


Figure 6: Heat maps of road occurrences among map matched probe-car data (left), trips by the baseline algorithm (middle), and trips by the tree-based algorithm (right).

of the simulation. By computational experiments, we observed that the tree-based algorithm could generate quite similar trips to those by the (shortest-path-based) baseline algorithm, and also could produce similar results of traffic simulation. Comparing the simulation results with the actual traffic congestion in Nairobi city, we showed that our method could reproduce some observed traffic congestion even from a coarse OD table that divides the city into 3×3 subareas. We believe that the proposed algorithm is suitable for large-scale trip generation from such coarse OD tables. However, we also found that it had a limitation when dealing with trajectory datasets from GPS devices of taxis. Namely, the generated trips can be quite different from the observed ones. There are several explanation, such as, experienced taxi drivers following *fastest* paths instead of shortest ones to avoid traffic jams (Yuan et al. 2010). We leave it as a future work to deal with the limitation.

REFERENCES

Abraham, I., D. Delling, A. Goldberg, and R. Werneck. 2011. "A Hub-Based Labeling Algorithm for Shortest Paths in Road Networks". In *Experimental Algorithms*, edited by P. Pardalos and S. Rebennack, Volume 6630 of *Lecture Notes in Computer Science*, 230–241. Springer Berlin Heidelberg.

Bast, H., S. Funke, P. Sanders, and D. Schultes. 2007. "Fast Routing in Road Networks with Transit Nodes". *Science* 316 (5824): 566.

Behrisch, M., L. Bieker, J. Erdmann, and D. Krajzewicz. 2011. "SUMO – Simulation of Urban Mobility: An Overview". In *Proceedings of The Third International Conference on Advances in System Simulation*, 23–28.

Dijkstra, E. W. 1959. "A Note on Two Problems in Connexion with Graphs". *Numerische Mathematik* 1 (1): 269–271.

Fredman, M. L., and R. E. Tarjan. 1987. "Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms". *J. ACM* 34 (3): 596–615.

Geisberger, R., P. Sanders, D. Schultes, and D. Delling. 2008. "Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks". In *Experimental Algorithms*, edited by C. McGeoch, Volume 5038 of *Lecture Notes in Computer Science*, Chapter 24, 319–333. Berlin, Heidelberg: Springer Berlin Heidelberg.

Goldberg, A. V., and C. Harrelson. 2005. "Computing the Shortest Path: A* Search Meets Graph Theory". In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, 156–165. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Gomes, G., A. May, and R. Horowitz. 2004. "Congested Freeway Microsimulation Model Using VISSIM". *Journal of the Transportation Research Board* 1876:71–81.

Gonzales, E. J., C. Chavis, Y. Li, and C. F. Daganzo. 2009. "Multimodal Transport Modelling for Nairobi, Kenya: Insights and Recommendations with an Evidence-Based Model". Technical Report UCB-ITS-VWP-2009-5, UC Berkeley Center for Future Urban Transport, A Volvo Center of Excellence.

Hart, P. E., N. J. Nilsson, and B. Raphael. 1968. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics* 4 (2): 100–107.

Jeannotte, K., A. Chandra, V. Alexiadis, and A. Skabardonis. 2004. "Traffic Analysis Toolbox – Volume II: Decision Support Methodology for Selecting Traffic Analysis Tools". Technical Report FHWA-HRT-04-039, Federal Highways Administration.

Morimura, T., and S. Kato. 2012. "Statistical Origin-destination Generation with Multiple Sources". In *Proceedings of the 21st International Conference on Pattern Recognition*, 3443–3446. Piscataway, New Jersey: IEEE.

Ni, D. 2010. "A Spectrum of Traffic Flow Modeling at Multiple Scales". In *Proceedings of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, 554–566. Piscataway, New Jersey: IEEE.

Nishi, R., H. Miki, A. Tomoeda, and K. Nishinari. 2009. "Achievement of Alternative Configurations of Vehicles on Multiple Lanes". *Physical Review E* 79:066119+.

Osogami, T., T. Imamichi, H. Mizuta, T. Morimura, R. Raymond, T. Suzumura, R. Takahashi, and T. Idé. 2012. "IBM Mega Traffic Simulator". Technical Report RT0896, IBM Research – Tokyo.

Raymond, R., T. Morimura, T. Osogami, and N. Hirosue. 2012. "Map Matching with Hidden Markov Model on Sampled Road Network". In *21st International Conference on Pattern Recognition (ICPR2012)*, 2242–2245. Piscataway, New Jersey: IEEE.

Thulasidasan, S., and S. J. Eidenbenz. 2009. "Accelerating Traffic Microsimulations: A Parallel Discrete-event Queue-based Approach for Speed and Scale". In *Proceedings of the 2009 Winter Simulation Conference*, edited by M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, 2457–2466. Piscataway, New Jersey: IEEE.

Wang, P. T. R., and W. P. Niedringhaus. 1993. "Distributed/parallel Traffic Simulation for IVHS Applications". In *Proceedings of the 1993 Winter Simulation Conference*, edited by G. W. Evans, M. Mollaghasemi, E. C. Russell, and W. E. Biles, 1225–1230. Piscataway, New Jersey: IEEE.

Yuan, J., Y. Zheng, X. Xie, and G. Sun. 2011. "Driving with Knowledge from the Physical World". In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, 316–324. New York: ACM.

Yuan, J., Y. Zheng, C. Zhang, W. Xie, X. Xie, G. Sun, and Y. Huang. 2010. "T-drive: Driving Directions Based on Taxi Trajectories". In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, 99–108. New York: ACM.

AUTHOR BIOGRAPHIES

TAKASHI IMAMICHI is a researcher of IBM Research – Tokyo. He belongs to the Smarter Cities Solution group. He received the Bachelor of Engineering, Master of Informatics, and Doctor of Informatics degrees from Kyoto University in 2004, 2006, and 2009, respectively. His research interests include cutting and packing problems and metaheuristics. He is a member of the Operations Research Society of Japan and the Information Processing Society of Japan. His email address is imamichi@jp.ibm.com.

RUDY RAYMOND is a member of Analytics and Optimization group of IBM Research – Tokyo. He studied in Kyoto University and obtained the Bachelor of Engineering, Master, and Doctor of Informatics degrees in 2001, 2003, and 2006, respectively. In IBM Research, he has been working with various clients in telecommunication and manufacturing industries. His latest research deals with tracking vehicles from their sequences of noisy GPS points for mobility pattern recognition and traffic sensing. His email address is raymond@jp.ibm.com.