Routing Algorithms in Traffic Assignment Modeling

Tuan Nam Nguyen and Gerhard Reinelt

University of Heidelberg
Institute of Computer Science
nam.nguyen@informatik.uni-heidelberg.de
gerhard.reinelt@informatik.uni-heidelberg.de

Workshop on Traffic Optimization
Heidelberg, October 8th, 2015

HGS MathComp

AG ComOpt
UNIVERSITÄT HEIDELBERG

Motivation

Figure: Traffic in Hanoi on October 8th, 2015. Source. Vnexpress.net

Outline

1. Introduction
   - Traffic Assignment Modeling (TAM)
   - Routing Problems in TAM
2. K Shortest Paths (KSP)
   - Some Exising Algorithms
   - New Heuristic Method HELF
   - Computational Results
3. K Dissimilar Shortest Paths (KDSP)
4. Application in TAM
   - A Case Study: Hanoi, Vietnam
   - Computational Results

Traffic Assignment Modeling

AG ComOpt
UNIVERSITÄT HEIDELBERG

Traffic assignment modeling

Traffic assignment modeling (TAM) aims at forecasting the number of trips on different links (road sections) of the network given the travel demand between different pairs of zones (or areas).

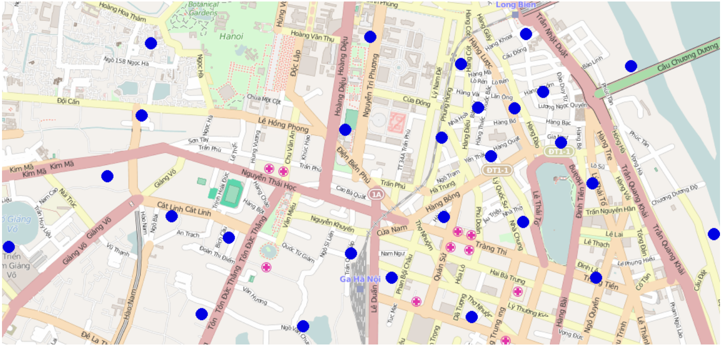


Figure: An example of traffic assignment. Source: TranOpt Plus

Traffic Assignment Modeling

Traffic assignment modeling

Traffic assignment modeling (TAM) aims at forecasting the number of trips on different links (road sections) of the network given the travel demand between different pairs of zones (or areas).

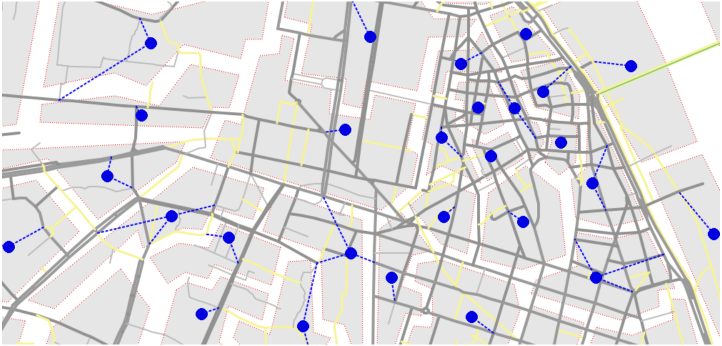


Figure: An example of traffic assignment. Source: TranOpt Plus

Traffic Assignment Modeling

Traffic assignment modeling

Traffic assignment modeling (TAM) aims at forecasting the number of trips on different links (road sections) of the network given the travel demand between different pairs of zones (or areas).

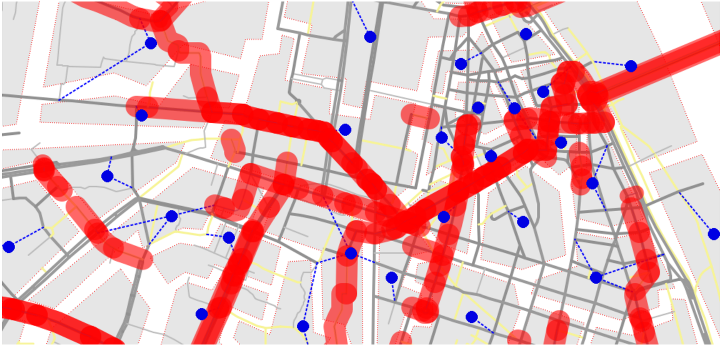


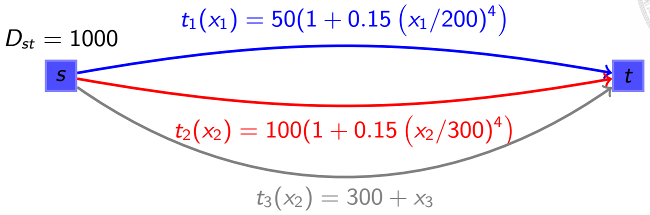
Figure: An example of traffic assignment. Source: TranOpt Plus

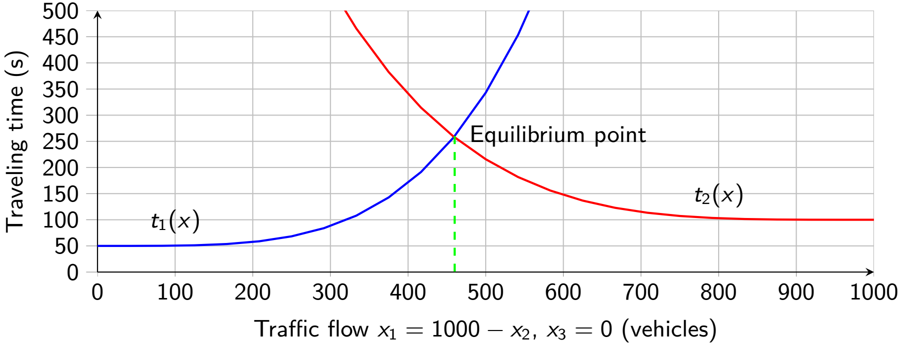
Traffic Assignment Models

Each TA model is made based on a number of assumptions on traffic behaviors, traffic networks, etc.

- *All or Nothing* (AON) model: Drivers choose the shortest path (in terms of length) to travel without consideration of traffic flow on the path.
- *Incremental* (INC) model: repeat the AON model for travel time, i.e. the travel time on links are updated regularly according to the traffic flow and drivers choose the shortest path (in terms of travel time) to travel.
- *System Equilibrium* model: drivers are cooperated to each other to minimize the total travel time on the whole system. This assumption based on the second principle of Wandrop.
- *User Equilibrium* (UE) model: Travel time on all used paths between two nodes are equal and less than those of un-used paths. This is based on the first principle of Wandrop.

The User Equilibrium Models





The User Equilibrium Models

- $t_{ij}(x_{ij})$: *travel time function on link* (i, j) where x_{ij} is the total traffic flow on the link (i, j),
- P_{rs} : *the set of potential paths from $r \in P$ to $s \in P$,*
- $\delta_{ij}^p = \begin{cases} 1 & \text{if link (i,j) is on path p} \\ 0 & \text{otherwise} \end{cases},$
- x_{rs}^p: *is the traffic flow on the potential path $p \in P^{rs}$.*

We have a *generalized user equilibrium (GUE)* model as following:

$$\text{Minimize } Z = \sum_{(i,j)\in E} \int_0^{x_{ij}} t_{ij}(\omega)d\omega, \qquad \text{(GUE)}$$

$$\text{Subject to } x_{ij} = \sum_{r,s\in P} \sum_{p\in P^{rs}} \delta_{ij}^p x_{rs}^p \qquad \forall (i,j) \in E \qquad (1)$$

$$\sum_{p\in P^{rs}} x_{rs}^p = d_{rs} \qquad \forall r, s \in P \qquad (2)$$

$$x_{rs}^p \geq 0 \qquad \forall r, s \in P, p \in P_{rs}. \qquad (3)$$

The User Equilibrium Models

For each pair of zones, the potential paths are possible paths between the nodes that satisfy some of routing behaviors, such as:

- Short in length
- Loop-less
- Dissimilar to each other
- Short time in congestion.

Routing Problems and Applications

AG ComOpt UNIVERSITÄT HEIDELBERG

Given a graph $G(V, E)$, two nodes $s, t \in V$.

- *Shortest path* (SP) problem
- *K shortest paths* (KSP) problem
  - Loop-less paths: KSLP problem
  - Non-loop-less paths: KSNLP problem
- *Dissimilar shortest paths* (DSP) problem
  - K shortest loop-less paths
  - $D(p_1, p_2) \geq \alpha$
- *Multi-objectvie shortest paths* (MOSP) problem

Popular algorithms: Floyd-Warshall, Bellman-Ford, Dijkstra, Yen, Martin, Eppstein, etc.

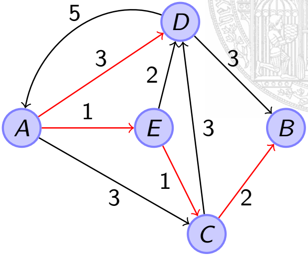


Figure: A example of graph.

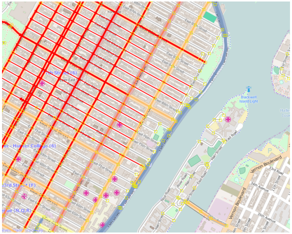


Figure: A part of map of NewYork.

Dissimilar paths with $minDifference = 30\%$. $N = 1$

Figure: Dissimilar shortest paths finding. Source: TranOpt Plus

Dissimilar paths with $minDifference = 30\%$. $N = 2$

Figure: Dissimilar shortest paths finding. Source: TranOpt Plus

Dissimilar paths with *minDifference* = 30%. $N = 3$

Figure: Dissimilar shortest paths finding. Source: TranOpt Plus

Dissimilar paths with $minDifference = 30\%$. $N = 10$

Figure: Dissimilar shortest paths finding. Source: TranOpt Plus

Applications of routing

- GPS navigation
- Logistic planning
- Games
- Routing services
- Army
- Traffic planning

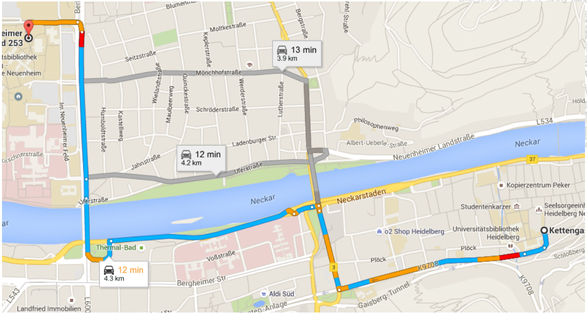


Figure: Routing service. Source: Google.

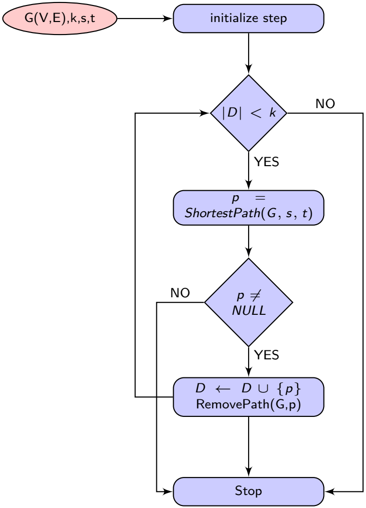
Note: Many paths routing problems are more and more importatnt in real applications.

K Shortest Paths

Table: Existing algorithms for KSP problem.

Year	Author	The Title
1971	J. Y. Yen	Finding the k shortest loopless paths
1975	K. Aihara	Enumerating elementary paths and cutsets by Gaussian elimination method
1976	T. D. Am et al.	An algorithm for generating all the paths between two vertices in a digraph and its application
1984	E. Q. V. Martins	An algorithm in paths removing approach. Next shortest path is found after removing previous shortest paths from the graph
1993	A. Aggarwal et al.	Finding a minimum weight K-link path in graphs with Monge property and applications
1997	D. Eppstein	Finding the k Shortest Paths
1999	Martin and Santos	Labeling approach as the extension of Dijkstra to find k shortest paths
2011	H. Aljazzar et al.	K^*: A heuristic search algorithm for finding the k shortest paths

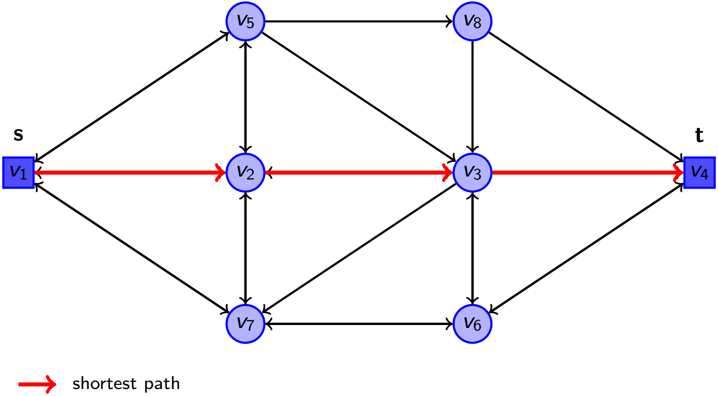
Paths Removing Approach



Principle

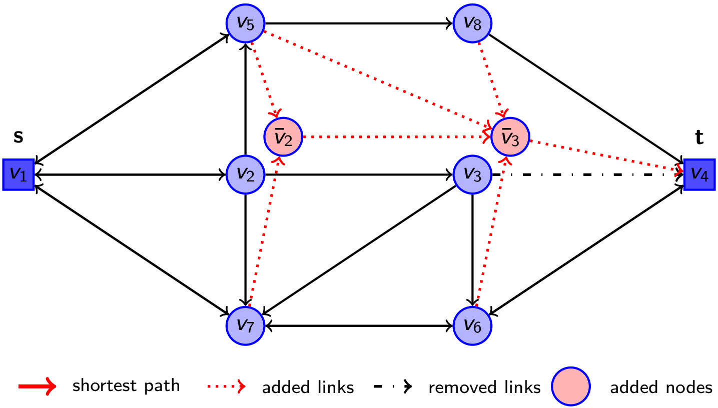
Given a graph $G(V, E)$ and a pair of nodes (s, t). If there are at least k possible different paths from s to t then the k^{th} shortest paths from s to t is the shortest path in the graph after removing only $k-1$ previous shortest paths from s to t.

Martin's Algorithm



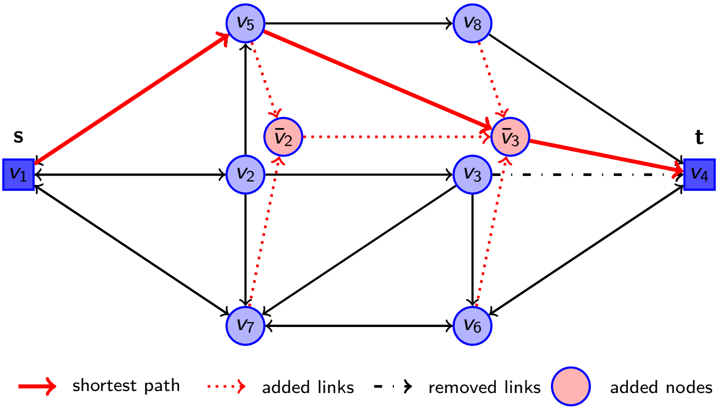
- Step1: get shortest path $p_1 = (v_1, v_2, v_3, v_4)$

Martin's Algorithm



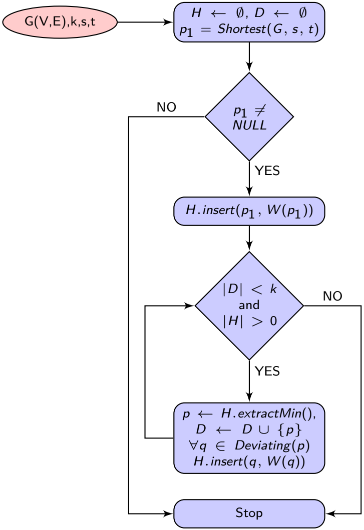
- Step1: get shortest path $p_1 = (v_1, v_2, v_3, v_4)$
- Step2: Remove p_1 from the graph

Martin's Algorithm



- Step1: get shortest path $p_1 = (v_1, v_2, v_3, v_4)$
- Step2: Remove p_1 from the graph
- Step3: Get shortest path in the new graph $\bar{p}_2 = (v_1, v_5, \bar{v}_3, v_4)$. This path refers to the original path $p_2 = (v_1, v_5, v_3, v_4)$

Paths Deviating Approach



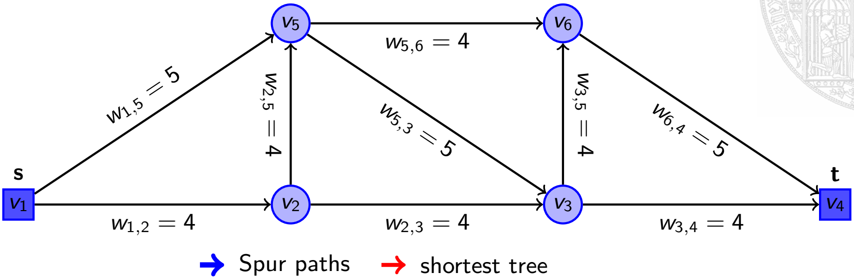
Idea

Generate a number of deviating paths from the previous found path and put them into a priority list, such that the next shortest path is always on the priority list.

Well-known algorithms

- Yen's algorithm: Proposed by Yen 1971. Find loop-less paths.
- Eppstein'a algorithm: proposed by Eppstein 1998. Find non-loop-less paths.

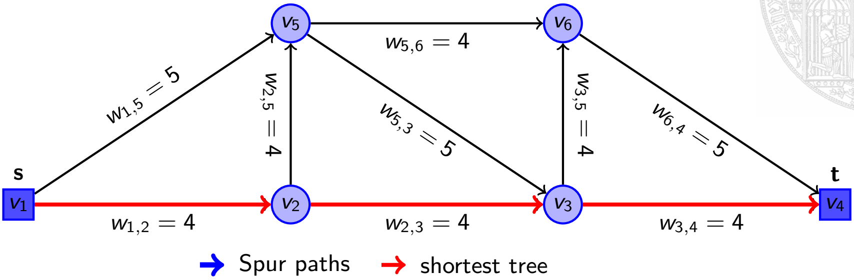
Yen's Algorithm



Found paths $D \leftarrow \emptyset$

Priority list of candidates H:

Yen's Algorithm

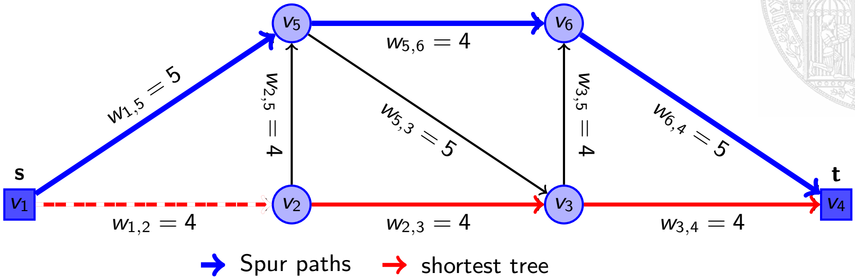


Found paths $D \leftarrow \emptyset$

$p_1 = (v_1, v_2, v_3, v_4)$

Priority list of candidates H:

Yen's Algorithm



Found paths $D \leftarrow \emptyset$

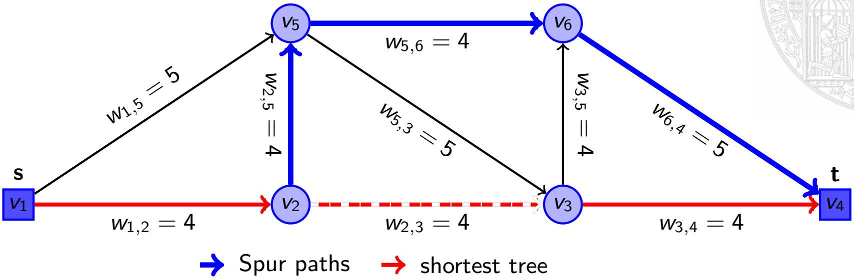
$p_1 = (v_1, v_2, v_3, v_4)$

Priority list of candidates H:

$p_2 = (v_1, v_5, v_6, v_4),\ w(p_2) = 14$

Yen's Algorithm

AG ComOpt UNIVERSITÄT HEIDELBERG



Found paths $D \leftarrow \emptyset$

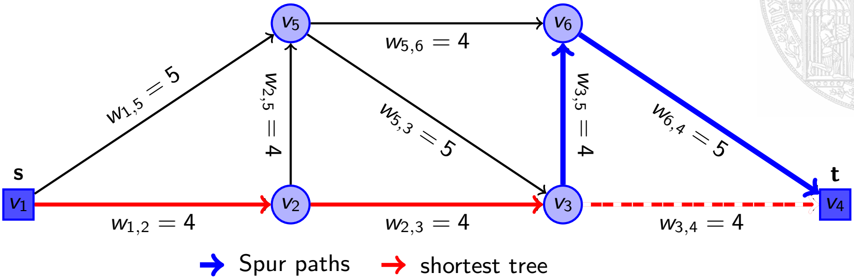
$p_1 = (v_1, v_2, v_3, v_4)$

Priority list of candidates H:

$p_2 = (v_1, v_5, v_6, v_4),\ w(p_2) = 14$

$p_3 = (v_1, v_2, v_5, v_6, v_4),\ w(p_3) = 17$

Yen's Algorithm



Found paths $D \leftarrow \emptyset$

$p_1 = (v_1, v_2, v_3, v_4)$

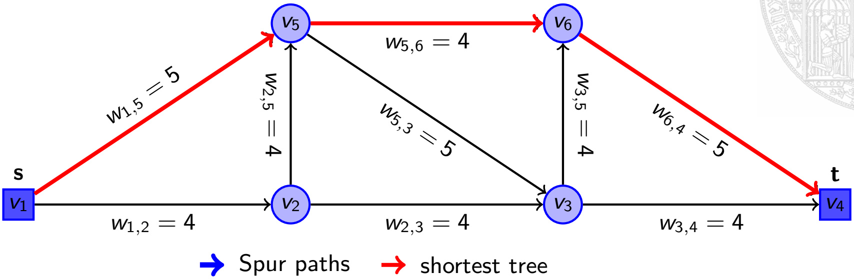
Priority list of candidates H:

$p_2 = (v_1, v_5, v_6, v_4),\ w(p_2) = 14$

$p_3 = (v_1, v_2, v_5, v_6, v_4),\ w(p_3) = 17$

$p_4 = (v_1, v_2, v_3, v_6, v_4),\ w(p_4) = 17$

Yen's Algorithm



Found paths $D \leftarrow \emptyset$

$p_1 = (v_1, v_2, v_3, v_4)$

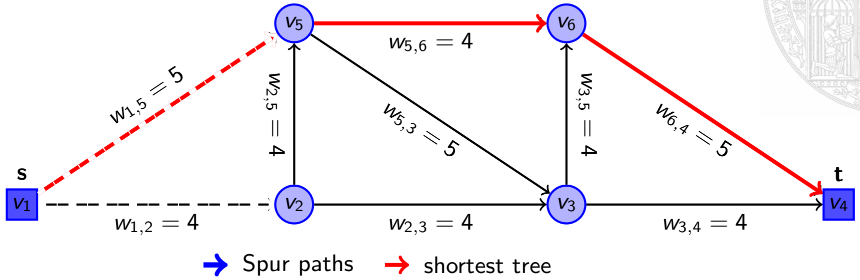
$p_2 = (v_1, v_5, v_6, v_4)$

Priority list of candidates H:

$p_3 = (v_1, v_2, v_5, v_6, v_4),\ w(p_3) = 17$

$p_4 = (v_1, v_2, v_3, v_6, v_4),\ w(p_4) = 17$

Yen's Algorithm



Found paths $D \leftarrow \emptyset$

$p_1 = (v_1, v_2, v_3, v_4)$

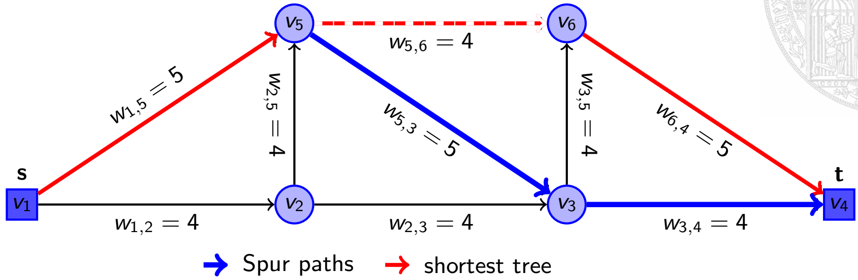
$p_2 = (v_1, v_5, v_6, v_4)$

Priority list of candidates H:

$p_3 = (v_1, v_2, v_5, v_6, v_4),\ w(p_3) = 17$

$p_4 = (v_1, v_2, v_3, v_6, v_4),\ w(p_4) = 17$

Yen's Algorithm



Found paths $D \leftarrow \emptyset$

$p_1 = (v_1, v_2, v_3, v_4)$

$p_2 = (v_1, v_5, v_6, v_4)$

Priority list of candidates H:

$p_3 = (v_1, v_2, v_5, v_6, v_4),\ w(p_3) = 17$

$p_4 = (v_1, v_2, v_3, v_6, v_4),\ w(p_4) = 17$

$p_5 = (v_1, v_5, v_3, v_4),\ w(p_4) = 14$

Eppstein's Algorithm

Give graph $G(V, E)$, s is the source node, and t is the destination node.

- Complexity $O(|E| + |V|lg(|V|) + k)$
- The found paths may contain cycles.

We denote:

- D_x (or $D(x, t)$) is the length of the shortest path from x to destination node t.
- $f_{ij} = w_{ij} + D_j - D_i$ is the increasing cost, $\forall (i, j) \in E$

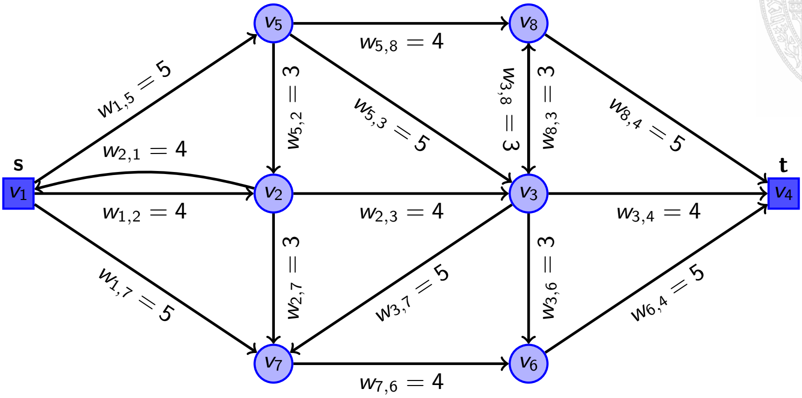
We have:

$$f_{ij} = \begin{cases} 0 & \text{if } (i,j) \text{ on the shortest path tree to } t \\ \geq 0 & \text{if } (i,j) \text{ is not in shortest path tree (called \textbf{side track})} \end{cases}$$

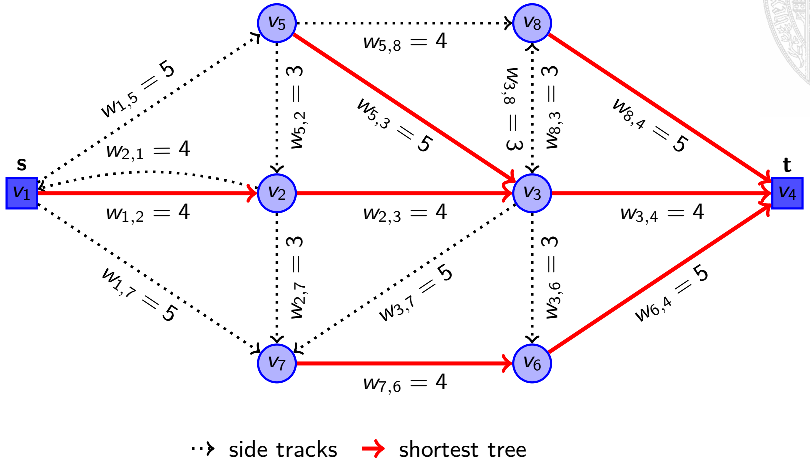
Proposition

Every path can be expressed as a sequence of side tracks.

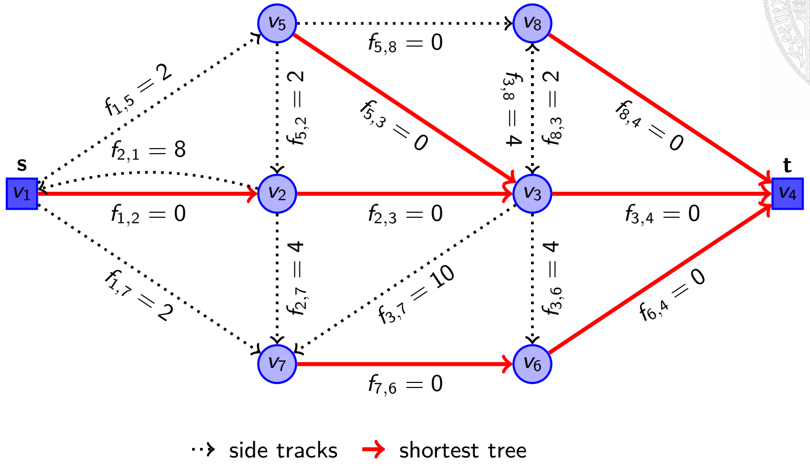
Eppstein's Algorithm



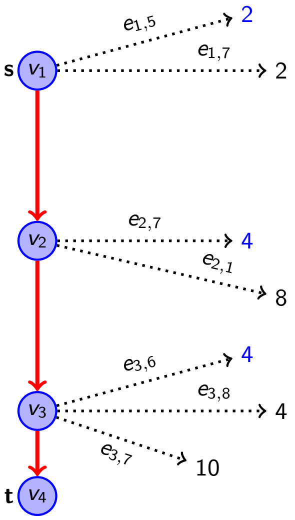
Eppstein's Algorithm



Eppstein's Algorithm

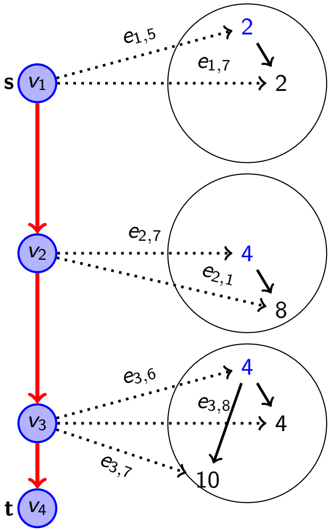


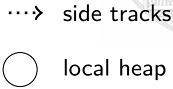
Eppstein's Algorithm: Heap Building



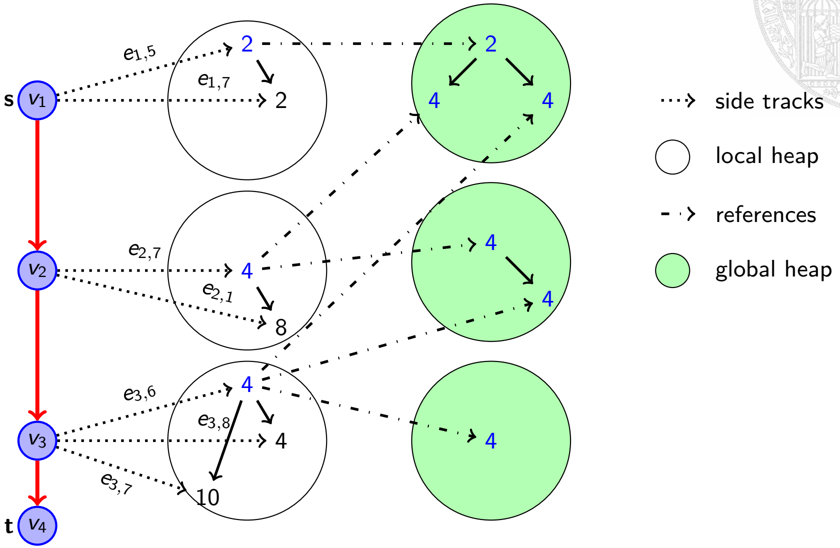
⋯› side tracks

Eppstein's Algorithm: Heap Building

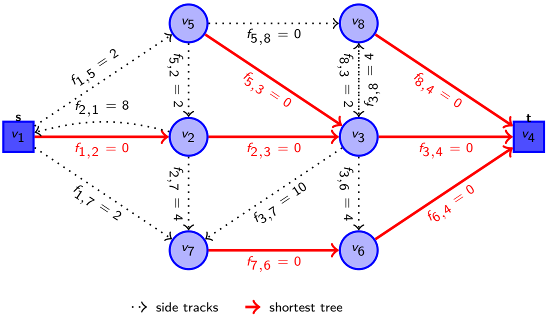




Eppstein's Algorithm: Heap Building



Eppstein's Algorithm: Selecting Paths



Found path:

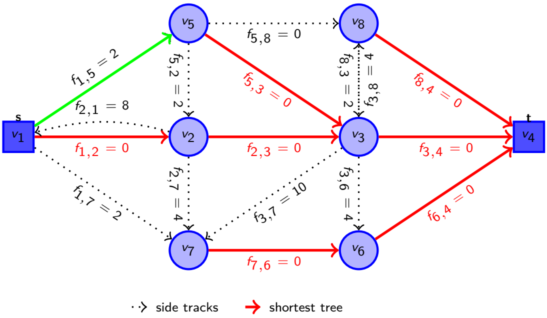
- $p_1 = (v_1, v_2, v_3, v_4)$,

Visiting side track:

Heap of candidate paths:

$$h_1 = (f_{1,5}), \qquad w(h_1) = L + 2$$

Eppstein's Algorithm: Selecting Paths



Heap of candidate paths:

$$h_2 = (f_{1,5} + f_{5,8}), \qquad w(h_2) = L + 2$$
$$h_3 = (f_{1,7}), \qquad w(h_3) = L + 2$$
$$h_4 = (f_{2,7}), \qquad w(h_4) = L + 4$$
$$h_5 = (f_{3,6}), \qquad w(h_5) = L + 4$$

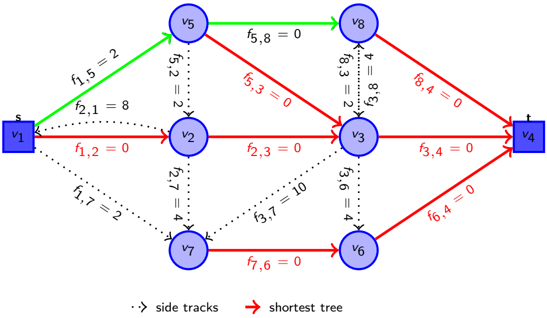
Found path:

- $p_1 = (v_1, v_2, v_3, v_4)$,
- $p_2 = h_1 = (f_{1,5}) = (v_1, v_5, v_3, v_4)$

Visiting side track:

$f_{1,5}$

Eppstein's Algorithm: Selecting Paths



Heap of candidate paths:

$$h_3 = (f_{1,7}), \qquad w(h_3) = L + 2$$
$$h_4 = (f_{2,7}), \qquad w(h_4) = L + 4$$
$$h_5 = (f_{3,6}), \qquad w(h_5) = L + 4$$
$$h_6 = (f_{1,5}, f_{5,2}), \qquad w(h_6) = L + 4$$

...

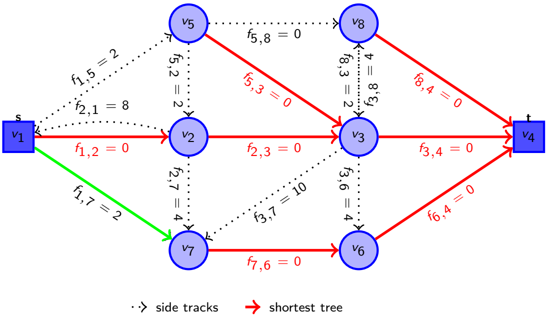
Found path:

- $p_1 = (v_1, v_2, v_3, v_4)$,
- $p_2 = h_1 = (f_{1,5}) = (v_1, v_5, v_3, v_4)$
- $p_3 = h_2 = (f_{1,5}, f_{5,8}) = (v_1, v_5, v_8, v_4)$

Visiting side track:

$f_{5,8}$

Eppstein's Algorithm: Selecting Paths



Heap of candidate paths:

$$h_4 = (f_{2,7}), \quad w(h_4) = L + 4$$
$$h_5 = (f_{3,6}), \quad w(h_5) = L + 4$$
$$h_6 = (f_{1,5}, f_{5,2}), \quad w(h_6) = L + 4$$

...

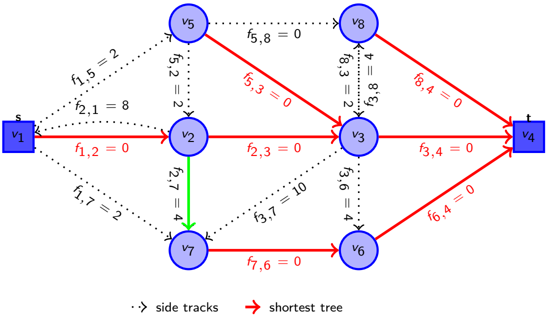
Found path:

- $p_1 = (v_1, v_2, v_3, v_4)$,
- $p_2 = h_1 = (f_{1,5}) = (v_1, v_5, v_3, v_4)$
- $p_3 = h_2 = (f_{1,5}, f_{5,8}) = (v_1, v_5, v_8, v_4)$
- $p_4 = h_3 = (f_{1,7}) = (v_1, v_7, v_6, v_4)$

Visiting side track:

$f_{1,7}$

Eppstein's Algorithm: Selecting Paths



Heap of candidate paths:

$h_5 = (f_{3,6})$, $w(h_5) = L + 4$

$h_6 = (f_{1,5}, f_{5,2})$, $w(h_6) = L + 4$

...

Found path:

- $p_1 = (v_1, v_2, v_3, v_4)$,
- $p_2 = h_1 = (f_{1,5}) = (v_1, v_5, v_3, v_4)$
- $p_3 = h_2 = (f_{1,5}, f_{5,8}) = (v_1, v_5, v_8, v_4)$
- $p_4 = h_3 = (f_{1,7}) = (v_1, v_7, v_6, v_4)$

Visiting side track:

$f_{2,7}$

Improvement for Eppstein's Algorithm

AG ComOpt UNIVERSITÄT HEIDELBERG

Inserting Loop-fess Filter

- Detect paths which generates only candidates with cycles, and elimiate it from the list of path without generating the cadidates from it.
- Reduce number of visited paths.

Original EA for KSLP

Step 1 Select the path with minimal value on the heap;

Step 2 Check if the the path is loop-less or not. If loop-less, put on the set of the found paths.

Step 3 Generate deviated paths from the previous selected path from the heap;

New heuristic method HELF

Step 1 Select the path p with minimal value on the heap;

Step 2 Check if p is loop-less or not. If loop-less, put on the set of the found paths.

Step 3 Apply loop-less filter. If p is not filtered by loop-less filter go to Step 4, else come back Step 1.

Step 4 Generate deviated paths from p

Computational Results

AG ComOpt UNIVERSITÄT HEIDELBERG

Table: Comparison between the heuristic based on the original EA and the new heuristic HELF.

City maps	Original Eppstein			HELF		
	Visited paths (Aver.)	Found paths (Aver.)	Time (Aver.)	Visited paths (Aver.)	Found paths (Aver.)	Time (Aver.)
HD-DE1k	437.33	10.00	0.0090	75.33	10.00	0.0106
HP-VN2k	1486.80	9.80	0.0206	91.50	10.00	0.0131
BH-VN4k	1648.73	9.40	0.0297	152.87	10.00	0.0197
NY-USA5k	1437.29	9.43	0.0423	20.71	10.00	0.0181
VT-VN5k	2895.27	9.27	0.0438	169.13	10.00	0.0237
MH-DE6k	1411.80	9.40	0.0445	63.87	10.00	0.0217
DN-VN8k	748.07	9.40	0.0610	37.80	10.00	0.0334
HN-VN9k	1348.93	9.40	0.0503	37.87	10.00	0.0389
PP-CB9k	12.33	10.00	0.0645	11.87	10.00	0.0393
MNL-PP12k	858.73	9.73	0.1000	38.00	10.00	0.0785
TP-TW21k	20.92	10.00	0.1023	18.75	10.00	0.1057
BK-TL22k	55.33	10.00	0.1071	25.87	10.00	0.0859
HCM-VN24k	44.53	10.00	0.2015	20.33	10.00	0.1351
Average	**954.31**	**9.68**	**0.0674**	**58.76**	**10.00**	**0.0480**

Computational Results

Table: Reduced time by using new heuristic in comparison to the original Eppstein.

Maps	k=5	k=10	k=20	k=30	k=40	k=50	k=60
	Diff.(%)	Diff.(%)	Diff.(%)	Diff.(%)	Diff.(%)	Diff.(%)	Diff.(%)
HD-DE1k	−14.67	*+17.78*	−83.75	−77.66	−90.63	−88.75	−83.95
HP-VN2k	−28.93	−36.41	−77.80	−79.15	−88.13	−88.24	−90.93
BH-VN4k	−12.17	−33.86	−75.35	−75.46	−77.94	−83.70	−82.36
NY-USA5k	−9.42	−57.09	−57.64	−66.53	−68.44	−64.08	−80.28
VT-VN5k	−12.93	−45.97	−75.36	−80.11	−79.56	−74.43	−86.90
MH-DE6k	−32.47	−51.35	−47.23	−66.74	−73.65	−75.10	−81.24
DN-VN8k	*+0.96*	−45.25	−51.85	−66.05	−61.34	−68.76	−85.13
HN-VN9k	−9.38	−22.55	−6.10	−33.28	−51.14	−62.18	−67.45
PP-CB9k	−16.15	−39.05	−15.15	−38.70	−29.18	−4.57	−45.27
MNL-PP12k	−27.05	−21.47	−14.57	−39.05	−38.12	−45.75	−67.17
TP-TW21k	−7.31	*+3.34*	−19.23	−0.78	−29.48	−5.06	−40.25
BK-TL22k	−13.37	−19.85	−9.32	−17.03	−16.35	−37.25	−68.25
HCM-VN24k	−6.73	−32.98	−35.70	*+1.54*	−2.35	−3.02	−42.53
Average	**-14.59**	**-29.59**	**-43.77**	**-49.15**	**-54.33**	**-53.91**	**-70.90**

Computational Results

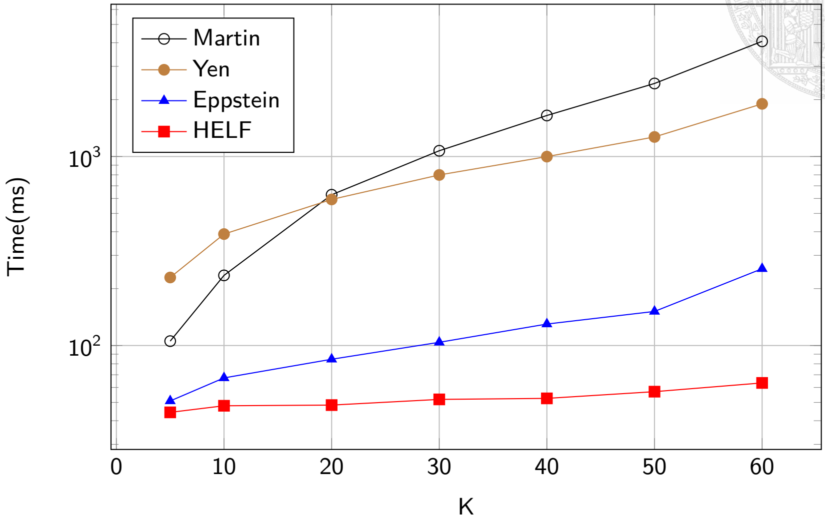


Figure: Average running time comparison for a set of city maps.

K Dissimilar Paths

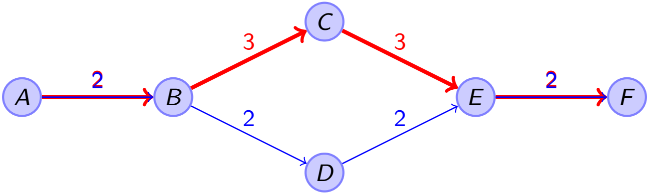
AG ComOpt UNIVERSITÄT HEIDELBERG

Definition of dissimilarity

Given two path p^1, p^2, the similarity and dissimilarity of two paths is defined as

$$S(p^1, p^2) = \frac{1}{2}\left(\frac{W(p^1 \cap p^2)}{W(p^1)} + \frac{W(p^1 \cap p^2)}{W(p^2)}\right)$$
$$D(p^1, p^2) = 1 - S(p^1, p^2),$$

where $W(p^1 \cap p^2)$ is the length (or weight) of the intersection between p^1 and p^2.



E. g.: $p^1 = (A, B, C, E, F)$ and $p^2 = (A, B, D, E, F)$.
$S(p^1, p^2) = \frac{1}{2}\left(\frac{2+2}{10} + \frac{2+2}{8}\right) = 0.45.$ $D(p^1, p^2) = 1 - 0.45 = 0.55.$

Common approaches

Off-line

Finding a list of N shortest loop-less paths, then select a sub-set of k paths such that as total dissimilarity of them are maximum.

- Min-max approach;
- $p-$dispersion approach.

On-line

Select one most suitable path in one iteration based on previous paths until got k paths.

- Penalty method;
- Gateway method.
- Heuristic methods using KSP algorithms

Disadvantage of the off-line approaches: can not guarantee to find enough k paths among N paths with the given minium dissimilarity between two paths.
Our approach: On-line approches: Penalty method and heuristic methods using HELF method.

Penalty Method and HELSF Method

AG ComOpt UNIVERSITÄT HEIDELBERG

Penalty Method

Step 1 $p = ShortestPath(s, t, G(V, E))$;

Step 2 If p is dissimilar with previous found paths in D. $D \leftarrow D \cup \{p\}$

Step 3 If link e on p, $e.weight += e.weight * penaltyFactor$

HELSF method

Step 1 Select the path p with minimal value on the heap;

Step 2 Check if p is loop-less or not. If loop-less, and p is dissimilar with found paths in D, $D \leftarrow D \cup \{p\}$

Step 3 Apply loop-less filter and similarity filter. If p does not satisfy the filters, go to Step 4, else come back Step 1.

Step 4 Generate deviated paths from p

Dissimilar paths: Results

Table: Comparison between HELSF and Penalty-Bidirectional Dijktra method with $k = 10$.

Min Diff.	HELDF		Penalty-Bidirectional Dijkstra					
			$\beta = 0.2$		$\beta = 0.4$		$\beta = 0.6$	
	Time(s)	Length	Time(s)	Length	Time(s)	Length	Time(s)	Length
0.10	0.1346	**1.0162**	0.0215	1.0909	0.3220	1.1336	0.0366	1.2163
0.15	0.2373	**1.0223**	0.0220	1.0935	0.0227	1.1336	0.0395	1.2163
0.20	0.5359	**1.0285**	0.0221	1.0989	0.0394	1.1336	0.0271	1.2163
0.25	1.3659	**1.0383**	0.0344	1.1165	0.0371	1.1381	0.0338	1.2163
0.30	1.9469	**1.0587**	0.0223	1.1165	0.0225	1.1514	0.0329	1.2163
0.35	3.2555	**1.0847**	0.0243	1.1323	0.0275	1.1721	0.0434	1.2163
0.40	8.4200	**1.1223**	0.0231	1.1411	0.0377	1.1784	0.0225	1.2326

Remark: HELSF give better average length of the found paths while Penalty method give better running time.

Case of Study: Hanoi, Vietnam

Figure: Traffic situation in Hanoi, Vietnam. Source. Vnexpress.net

Traffic Online Survey in Hanoi

Link of the online survey: http://goo.gl/forms/mdh6p8AKbr
Summary of results: Results

Hanoi traffic

- Mixed traffic dominated by motorcycles;
- High demands of traffic both cars and motorcycles;
- Poor infracstructure;
- Many drivers don't follow traffic rules.

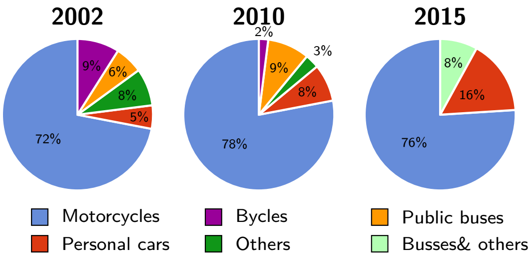


Figure: Vehicles shares in 2002, 2010 and 2015.

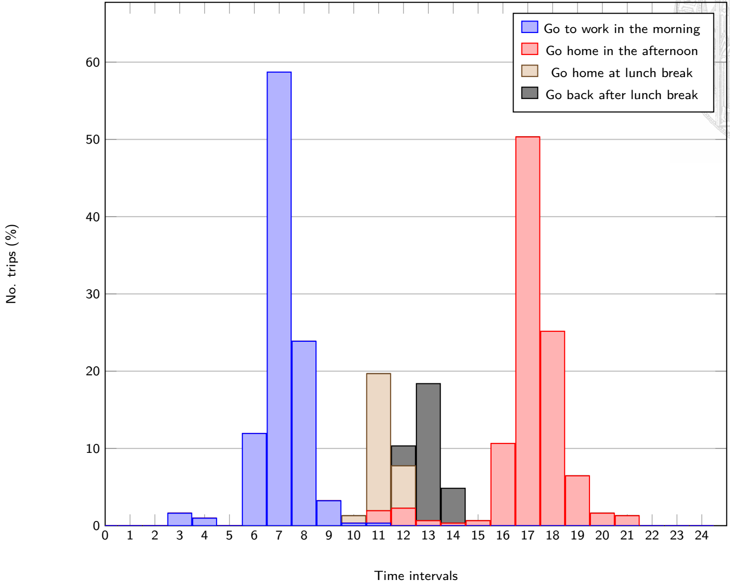


Figure: The trip distributions in a week day. Source: N.T. Nam, online survey 2015.

Table: The routing factors in Hanoi. Source: N. T. Nam, online survey in 2015.

Routing factors	Selection (%)
Probability of congestion on the path	**83.1**
The length of the path	**72.5**
The traffic density of the path	**51.1**
The cleanliness of the path	39.1
Number of traffic lights on the path	31.3
Number of traffic polices on the path	17.3
Other factors	8.1

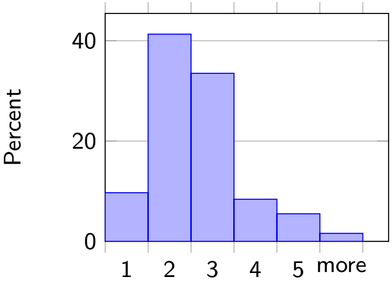


Figure: Number of alternative paths of drivers in Hanoi. Source: N. T. Nam, onine survey 2015.

Results in Traffic Assignment Modeling

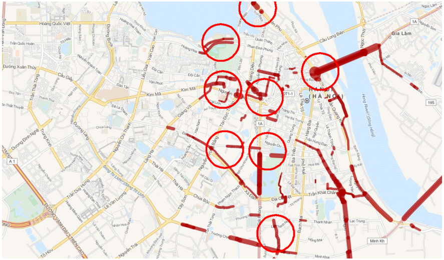


Figure: Assigned flows by MUE model.

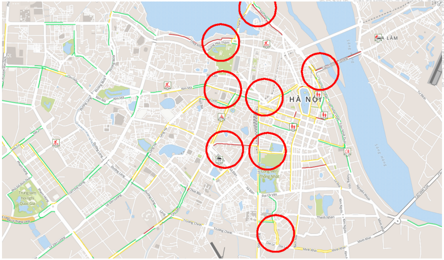


Figure: Real traffic flow provided by Remon-Hanoi project.

- High agreement between high-density areas predicted by UE assignment modeling, and real traffic congested areas.
- Finding set of potential paths is a very important task in traffic assignment modeling.

Appendix: TranOpt Plus Software

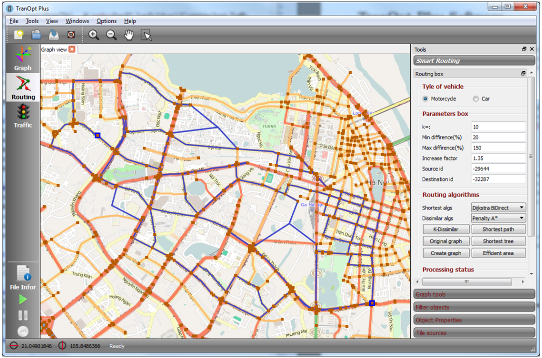


Figure: Screen shot of TranOpt Plus software.

- Written in $C++$ language, on Qt Creator
- Used Qt library to create interface and visualization
- Used open source code: MapGraphics and

The main features:

- Graph editor
  - Import openstreetmap
  - Create and edit graph objects
- Dynamic routing
  - k Shortest paths
  - Dissimilar paths
  - Shortest tree
- Traffic assignment modeling
  - Data handling
  - Mixed traffic assignment
  - Visualization tools
  - Results analysis

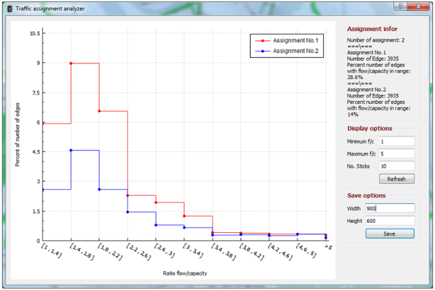
Appendix: TranOpt Plus Software

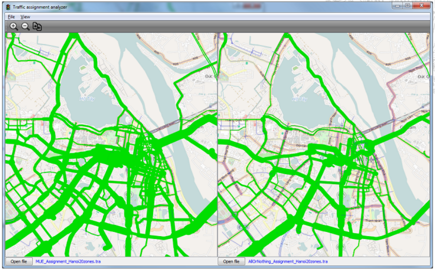
Table: Basic information about project TranOpt Plus Software.

	Infor	Comment
Language	$C++$	Version C++11
Time	From December 2013	Some algorithms was developed from 2012
IDE	Qt Creator	Use Qt Creator ver. 3 on Window 7
Version	Current ver. 1.0.2	The first version was released in March 2015
Platform	Window	Tested on window 7 and window 8
	Open source libraries	
Interface	Qt Library	Cross-platform application framework
Graphic	MapGraphics	Used MapGraphics as the core. Corrected and developed more.
Plot	QCustomPlot	Provided by Emanuel Eichhammer without changing
	Own developed libraries	
Tnga	Tnga/Routing	Containing dynamic routing algorithms
Tnga	Tnga/TA	Containing basic Traffic Assignment models

Website: https://nguyentuannam.wordpress.com/projects/tranopt-plus/

Appendix: TranOpt Plus Software





- Analyze the assigned flows
- Compare with other assignments

- Visualization of the assignment result
- Compare results of two assignment

Appendix: Testing Maps

Table: The information of of instances for implementation.

Name	City	Country	Nr. nodes	Nr. links
HD-DE1k	Heidelberg	Germany	1752	3517
HP-VN2k	Hai Phong	Vietnam	2731	6733
BH-VN4k	Bien Hoa	Vietnam	4749	11788
NY-USA5k	New York	USA	5653	11322
VT-VN5k	Vung Tau	Vietnam	5030	12978
MH-DE6k	Mannheim	Germany	6439	12504
DN-VN8k	Da Nang	Vietnam	8267	22927
HN-VN9k	Hanoi	Vietnam	9753	24205
PP-CB9k	Phnompenh	Cambodia	9896	26312
MNL-PP12k	Manila	The Philippines	12932	34638
TP-TW21k	Taipet	Taiwan	21137	49774
BK-TL22k	Bangkok	Thailand	22775	48139
HCM-VN24k	HoChiMinh City	Vietnam	24965	62566

Summary

- Basic about traffic assignment modeling;
- K shortest paths problem;
- K dissimilar shortest paths problem;
- Applications in Traffic Assignment modeling.
- TranOpt Plus software.

THANK YOU!