Bidirectional A^* on Time-Dependent Graphs

Giacomo Nannicini[1,2], Daniel Delling[3],
Leo Liberti[1], Dominik Schultes[3]

[1] *LIX, École Polytechnique, France*
[2] *Mediamobile, Paris, France*
[3] *Algoritmik, Universität Karlsruhe, Germany*

CTW 2008

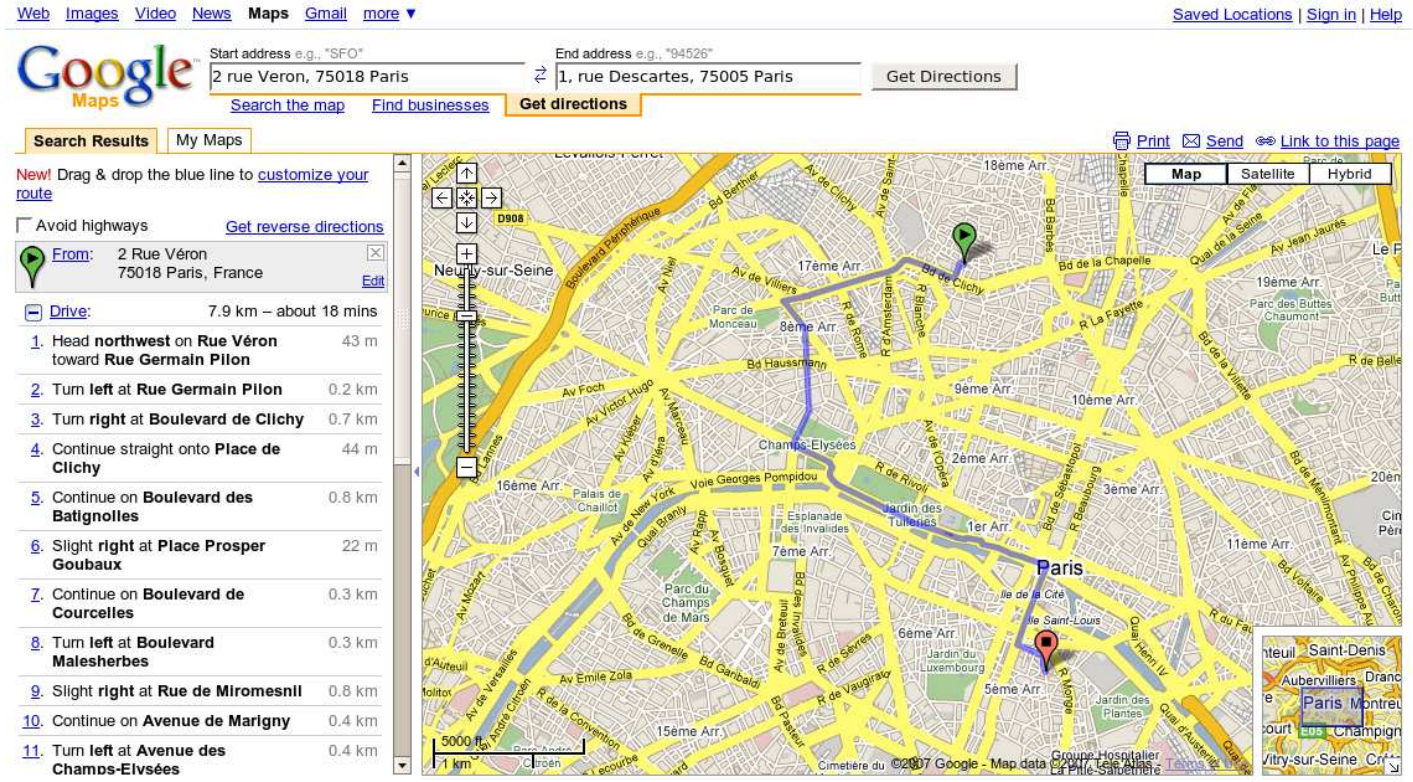
Summary of Talk

ÉCOLE POLYTECHNIQUE

vetrafic

Context Definition

- Computing point-to-point shortest paths is of great interest to many users:
  - GPS devices with path computing capabilities
  - Many web sites provide users with route planners

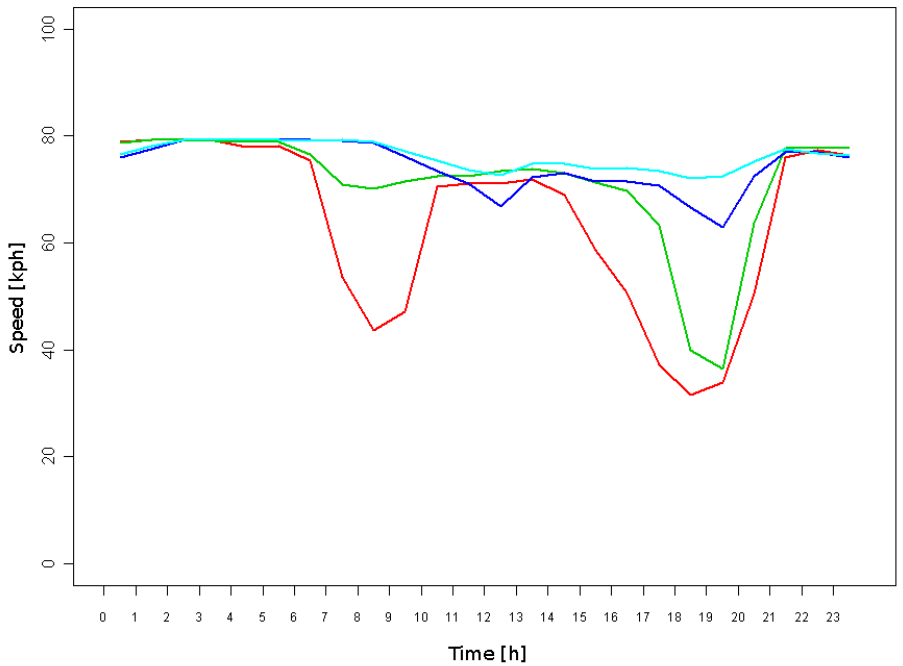


ÉCOLE POLYTECHNIQUE

vetrafic

Traffic is Time-Dependent

- Users are interested in the shortest path in terms of **travel time**, not of path length
- Traffic increases during peak hours
- Historical data can help in defining a cost function for each arc for each time instant of a day



ÉCOLE POLYTECHNIQUE
vetrafic

Problem Definition

- We are given a graph $G = (V, A)$ with a cost function $c : A \times T \longrightarrow \mathbb{R}$ which associates a travel time to each arc for each time instant, a source node s, a target node t and a departure time τ_0; we want to compute the shortest $s \longrightarrow t$ path leaving node s at τ_0 (**P**oint-to-**P**oint **T**ime-**D**ependent **S**hortest **P**ath **P**roblem)
- The time dependent cost function c is known for each arc and each time instant
- In FIFO networks, Dijkstra's algorithm applied to time-dependent graphs correctly solves the **PPTDSPP**

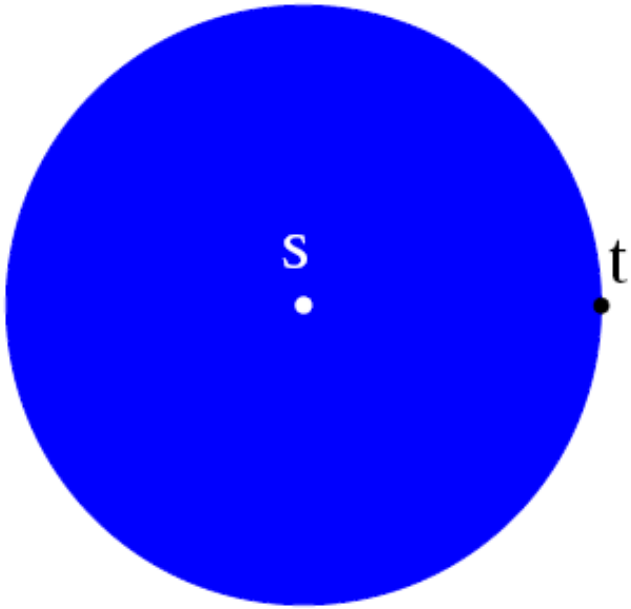
Time Constraints

- Road networks can be very large
  - Europe: roughly 18M nodes, 42M arcs
- We want to compute the shortest path in a low time
  - A very good implementation of Dijkstra's algorithm explores on average 9M nodes and takes more than 6 seconds (european road network): it's too long!
- There are several fast algorithms which compute exact solutions for the **PPSPP** on a static graph (e.g. Highway Hierarchies [Sanders and Schultes, 2006], Reach + *ALT* [Goldberg et al., 2005]), but they cannot be applied to time-dependent graphs

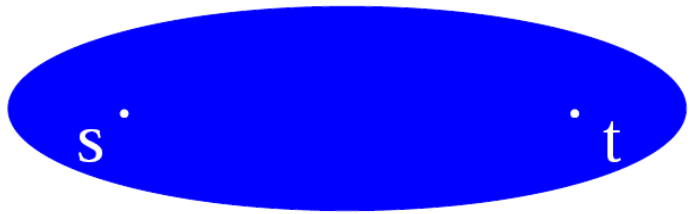
Goal Directed Search: A^*

- Same principle as Dijkstra's algorithm: extract minimum from a queue, explore adjacent nodes, update labels, repeat
- Main difference: add to the key of the priority queue a potential function $\pi(v)$ which estimates $d(v, t)$
- If $\pi(v) \leq d(v, t) \quad \forall v$ then A^* computes shortest paths
- If $\pi(v)$ is a good estimation of $d(v, t)$, A^* explores considerably fewer nodes than Dijkstra's algorithm

Goal Directed Search: A^*



Dijkstra's algorithm



A^*

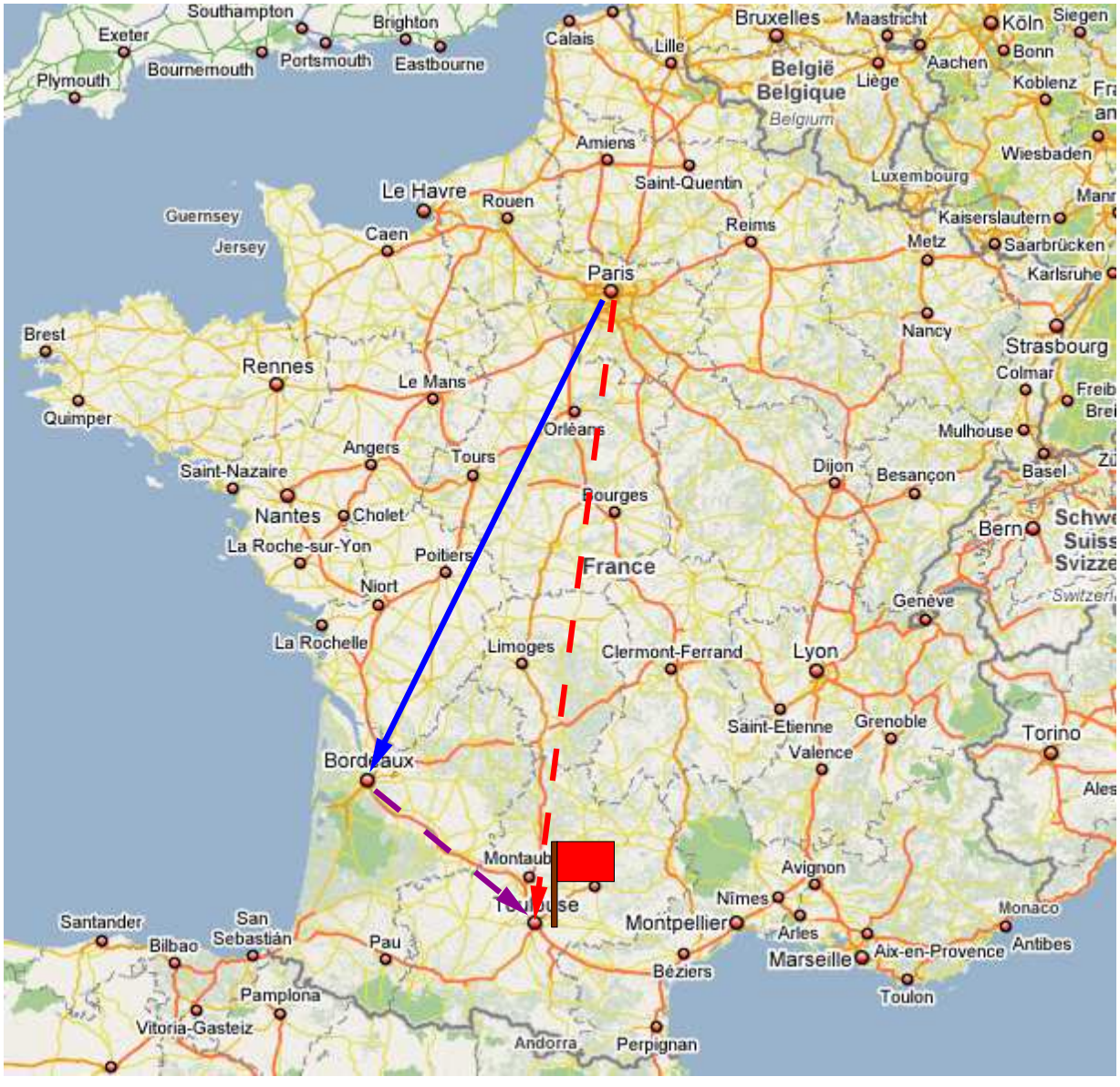
ÉCOLE POLYTECHNIQUE

vetrafic

A Good Lower Bound

- The quality of $\pi(v)$ is critic for performances: the closer to $d(v, t)$, the better
- Suppose we are in the static case
- Idea ([Goldberg and Harrelson, 2005]): use a few nodes as landmarks to compute distances within the graph
- Then triangle inequality comes to our help
  - ALT algorithm: A^*, Landmarks, Triangle inequality

A Good Lower Bound



ÉCOLE POLYTECHNIQUE

vetrafic

A Good Lower Bound

- Suppose we have a set $L \subset V$ of landmarks, i.e. we know $d(v, \ell), d(\ell, v) \quad \forall v \in V, \ell \in L$
- Then we have $d(v, \ell) \leq d(v, t) + d(t, \ell)$ and $d(\ell, t) \leq d(\ell, v) + d(v, t) \quad \forall v \in V, \ell \in L$

A Good Lower Bound

- Suppose we have a set $L \subset V$ of landmarks, i.e. we know $d(v, \ell), d(\ell, v) \quad \forall v \in V, \ell \in L$
- Then we have $d(v, \ell) \leq d(v, t) + d(t, \ell)$ and $d(\ell, t) \leq d(\ell, v) + d(v, t) \quad \forall v \in V, \ell \in L$

Lower bounding function:

$$\pi(v) = \max_{\ell \in L} \max\{d(v, \ell) - d(t, \ell), d(\ell, t) - d(\ell, v)\}$$

is a lower bound to $d(v, t)\, \forall v, t \in V$

Static ALT Algorithm

- Using some care in defining a valid potential function, bidirectional search can be applied on a static graph
  - The sum of forward and backward potential function should be constant $\forall v \in V$; we can use $p_f(v) = (\pi_f(v) - \pi_b(v))/2$ and $p_b(v) = -p_f(v)$ ([Delling and Wagner, 2007])
- If landmarks are well chosen, we get a speed-up factor of 50 with respect to bidirectional Dijkstra (which is already 2 times faster than the unidirectional version)
- But what about time-dependent graphs?

Static vs Time-Dependent

- Given a departure time, we do not know the exact arrival time (we need the optimal solution for that)

Static vs Time-Dependent

- Given a departure time, we do not know the exact arrival time (we need the optimal solution for that)

Problem:

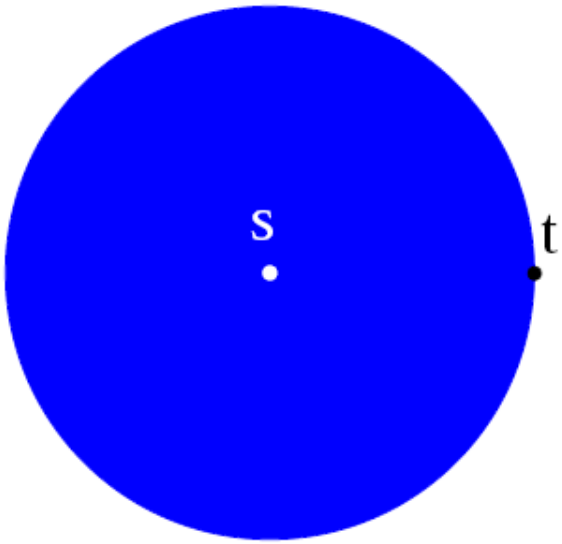
Bidirectional search cannot be applied "as it is", since we do not know which time instant we should use to compute time-dependent costs during the backward search

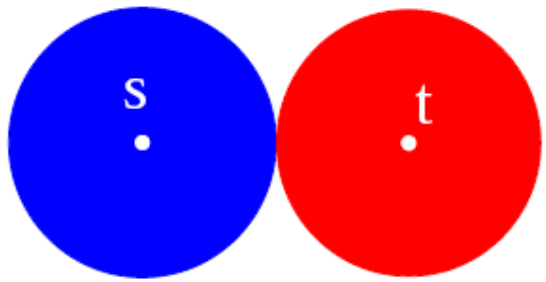
Static vs Time-Dependent

- Given a departure time, we do not know the exact arrival time (we need the optimal solution for that)

Problem:

Bidirectional search cannot be applied "as it is", since we do not know which time instant we should use to compute time-dependent costs during the backward search





ÉCOLE POLYTECHNIQUE

vetrafic

Time-Dependent ALT

- If we compute distances to and from landmarks using the lowest possible cost on each arc, then the lower bound $\pi(v)$ is still valid
- When arc costs are much higher than the value we used to compute landmark distances, performance decreases
- Using unidirectional ALT yields a speed-up of a factor 2.5 with respect to unidirectional Dijkstra – still in the order of seconds

Time-Dependent ALT

- If we compute distances to and from landmarks using the lowest possible cost on each arc, then the lower bound $\pi(v)$ is still valid
- When arc costs are much higher than the value we used to compute landmark distances, performance decreases
- Using unidirectional ALT yields a speed-up of a factor 2.5 with respect to unidirectional Dijkstra – still in the order of seconds

Question:

Can we apply bidirectional search on time-dependent graphs?

Bidirectional Search

- Idea: do a forward search using time-dependent costs, and a backward search using lower bounds on costs (i.e. the lowest possible cost on each arc)
- Then, as soon as we the two search scopes meet, compute the time-dependent cost of an $s \longrightarrow t$ path
- That path is a feasible solution, so its cost is an upper bound to the optimal cost

Bidirectional Search

- Continue both searches, alternating between forward search (with time-dependent costs) and backward search (with lower bounds on costs)
- The backward search cannot explore nodes already explored by the forward search

Bidirectional Search

- Continue both searches, alternating between forward search (with time-dependent costs) and backward search (with lower bounds on costs)
- The backward search cannot explore nodes already explored by the forward search

Question:

What happens when the minimum of the backward search queue becomes greater than the upper bound on the optimal cost?

Bidirectional Search

- The key of a node in the priority queue is a lower bound on the cost of an $s \longrightarrow t$ shortest path that passes through that node
- Let μ be our best upper bound and β be the smallest key in the backward search priority queue

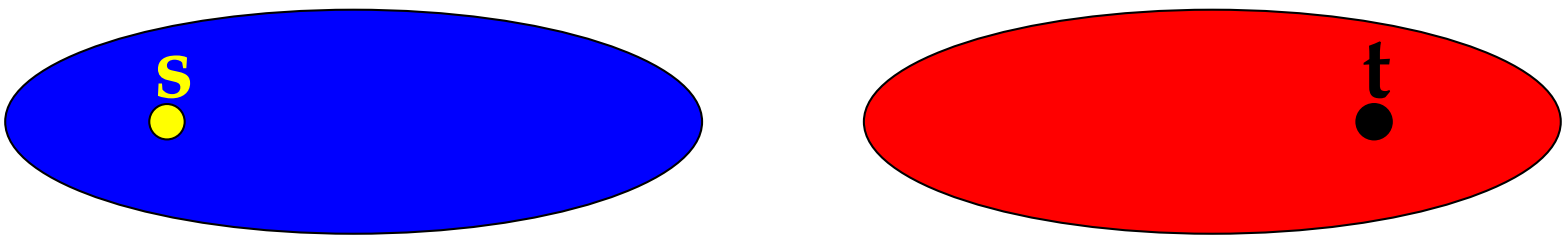
Bidirectional Search

- The key of a node in the priority queue is a lower bound on the cost of an $s \longrightarrow t$ shortest path that passes through that node
- Let μ be our best upper bound and β be the smallest key in the backward search priority queue

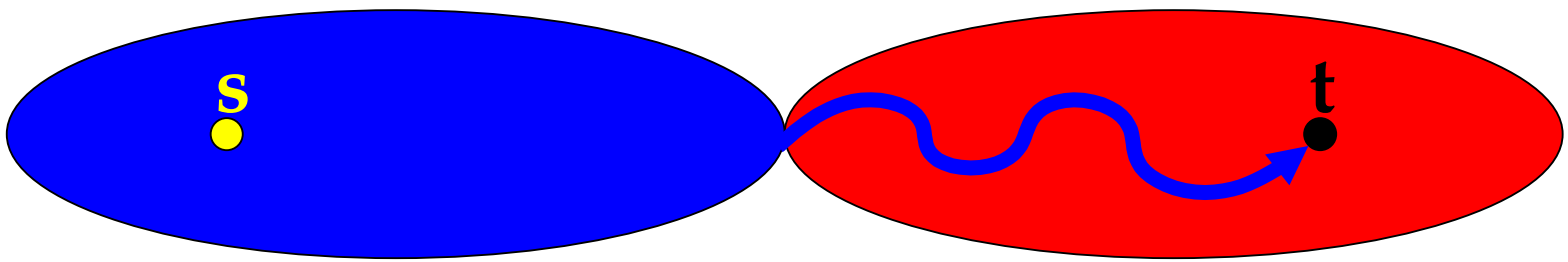
Theorem:

When $\beta \geq \mu$, the shortest path from s to t contains only nodes that have already been explored by the forward or backward search

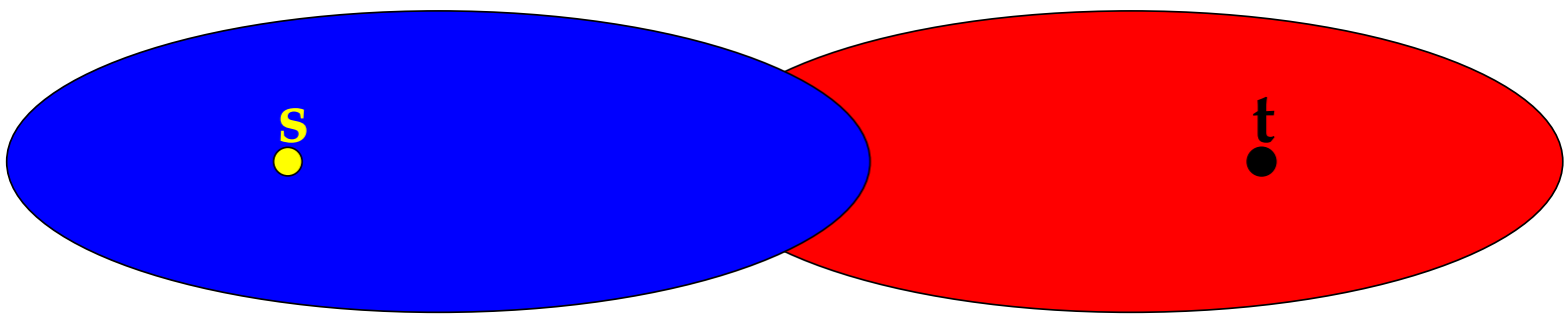
Bidirectional Search



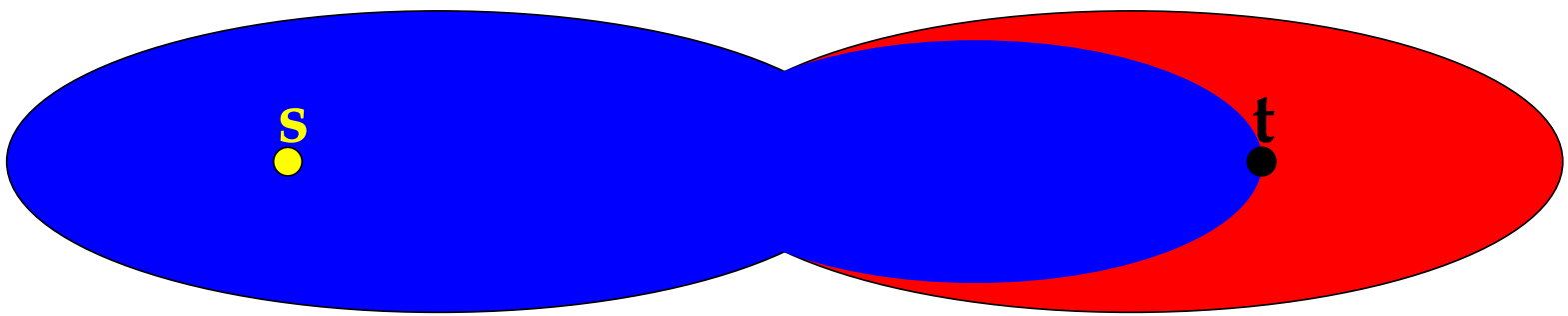
Bidirectional Search



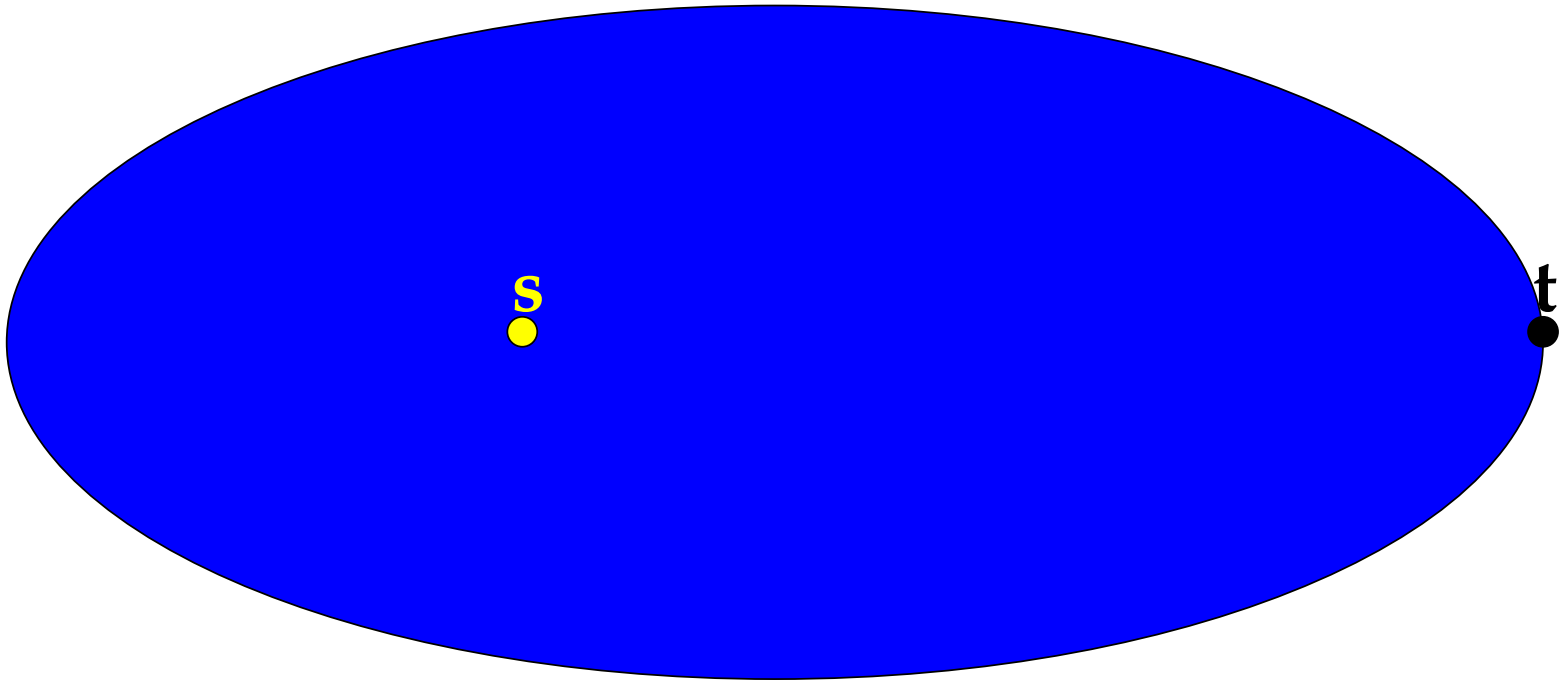
Bidirectional Search



Bidirectional Search



Bidirectional Search



Improvements

- If we artificially increase the lower bound β by a factor K, we get a K-approximated solution
  - When $K\beta \geq \mu$, the shortest path restricted to contain only nodes already explored by the forward or backward search has a cost which is no more than K times the optimum cost
  - The speed-up varies with K
- We can also tighten the bounds used by the backward search
  - For those nodes that have not been settled by the forward search, we can find a better bound than $\pi_b(v)$

ÉCOLE POLYTECHNIQUE

vetrafic

Random Queries

		Error			Query	
			relative		# settled	time
method	K	rate	avg	max	nodes	[ms]
Dijkstra	-	0.0%	0.000%	0.00%	8 908 300	6 325.8
uni-ALT	-	0.0%	0.000%	0.00%	2 192 010	1 775.8

Random Queries

		Error			Query	
			relative		# settled	time
method	K	rate	avg	max	nodes	[ms]
Dijkstra	-	0.0%	0.000%	0.00%	8 908 300	6 325.8
uni-ALT	-	0.0%	0.000%	0.00%	2 192 010	1 775.8
ALT	1.00	0.0%	0.000%	0.00%	3 117 160	3 399.3
	1.02	1.0%	0.003%	1.13%	2 560 370	2 723.3
	1.05	4.0%	0.029%	4.93%	1 671 630	1 703.6

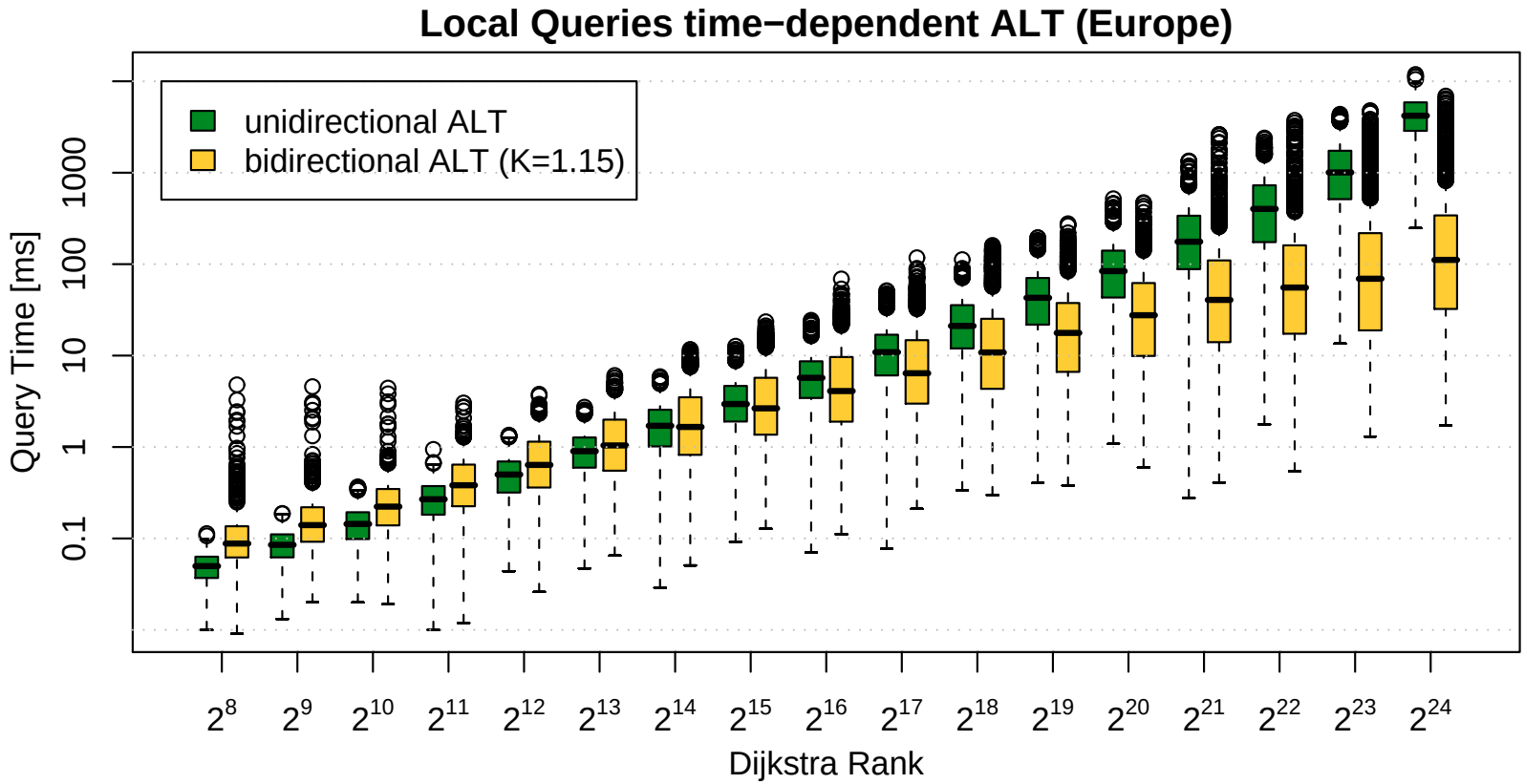
Random Queries

		Error			Query	
			relative		# settled	time
method	K	rate	avg	max	nodes	[ms]
Dijkstra	-	0.0%	0.000%	0.00%	8 908 300	6 325.8
uni-ALT	-	0.0%	0.000%	0.00%	2 192 010	1 775.8
ALT	1.00	0.0%	0.000%	0.00%	3 117 160	3 399.3
	1.02	1.0%	0.003%	1.13%	2 560 370	2 723.3
	1.05	4.0%	0.029%	4.93%	1 671 630	1 703.6
	1.10	18.7%	0.203%	8.10%	719 769	665.1
	1.13	30.5%	0.366%	12.63%	447 681	385.5
	1.15	36.4%	0.467%	13.00%	348 325	287.3

Random Queries

		Error			Query	
			relative		# settled	time
method	K	rate	avg	max	nodes	[ms]
Dijkstra	-	0.0%	0.000%	0.00%	8 908 300	6 325.8
uni-ALT	-	0.0%	0.000%	0.00%	2 192 010	1 775.8
ALT	1.00	0.0%	0.000%	0.00%	3 117 160	3 399.3
	1.02	1.0%	0.003%	1.13%	2 560 370	2 723.3
	1.05	4.0%	0.029%	4.93%	1 671 630	1 703.6
	1.10	18.7%	0.203%	8.10%	719 769	665.1
	1.13	30.5%	0.366%	12.63%	447 681	385.5
	1.15	36.4%	0.467%	13.00%	348 325	287.3
	1.20	44.7%	0.652%	18.19%	241 241	185.3
	1.30	48.2%	0.804%	23.63%	186 267	134.6
	1.50	48.8%	0.844%	25.70%	172 157	121.9
	2.00	48.9%	0.886%	48.86%	165 650	115.7

Local Queries



Comparison of uni- and bidirectional ALT using the Dijkstra rank methodology (box-and-whiskers plot).

ÉCOLE POLYTECHNIQUE

vetrafic

ÉCOLE POLYTECHNIQUE

vetrafic

Future Research

- Balancing forward and backward search
- Efficient updates of the upper bound
- Core-routing: extract a smaller subgraph (*core*) and carry out the computations on it
- Dynamic scenario: the coefficients of the piecewise linear time-dependent cost function may vary

The End

Thank you!

ÉCOLE POLYTECHNIQUE

vetrafic

References

Delling, D. and Wagner, D. (2007).
Landmark-based routing in dynamic graphs.
In Demetrescu, C., editor, *WEA 2007*, volume 4525 of *LNCS*, pages 52–65, New York. Springer.

Goldberg, A. and Harrelson, C. (2005).
Computing the shortest path: A^* meets graph theory.
In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005)*, pages 156–165, Philadelphia. SIAM.

Goldberg, A., Kaplan, H., and Werneck, R. (2005).
Reach for A^*: Efficient point-to-point shortest path algorithms.
In Demetrescu, C., Sedgewick, R., and Tamassia, R., editors, *Proceedings of the 7th Workshop on Algorithm Engineering and Experimentation (ALENEX 05)*, Philadelphia. SIAM.

Sanders, P. and Schultes, D. (2006).
Engineering highway hierarchies.
In *ESA 2006*, volume 4168 of *Lecture Notes in Computer Science*, pages 804–816. Springer.

ÉCOLE POLYTECHNIQUE

vetrafic