Karlsruhe Institute of Technology

Advanced Route Planning and Related Topics – *“Freiburg Update”*

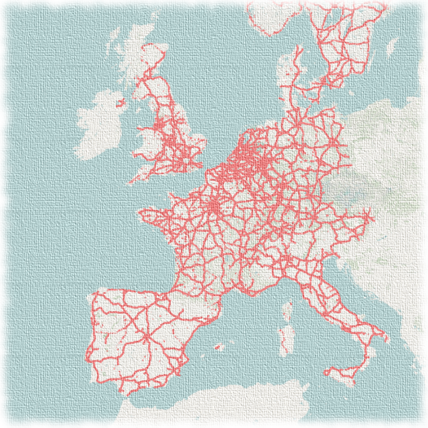
G. V. Batz, M. Kobitzsch, D. Luxen, P. Sanders, D. Schieferdecker

Institute of Theoretical Informatics - Algorithmics

Agenda

Route Planning

- Parallel Multi-Objective
- Transit Nodes
- Alternatives
- Stochastics



Label Setting Multiobjective SP

Procedure paretoSearch(G, s)

$L[v]{:=}\emptyset$ for all $v \in V$; $L[s]{:=}\ \{0^d\}$

PriorityQueue $Q = \{(s, 0^d)\}$

while $Q \neq \emptyset$ **do**

remove some Pareto optimal label L from Q

scan L

insert new locally nondominated labels $(v, \vec{\ell})$ into Q

and remove old locally dominated labels

Parallel Label Setting MOSP

Procedure paretoSearch(G, s)

$L[v] := \emptyset$ for all $v \in V$; $L[s] := \{0^d\}$

ParetoQueue $Q = \{(s, 0^d)\}$

while $Q \neq \emptyset$ **do**

remove all Pareto optimal labels L from Q

scan all labels in L in parallel

insert new locally nondominated labels $(v, \vec{\ell})$ into Q
and remove old locally dominated labels

Theorem: $\leq n$ iterations
Theorem: (super)linear speedup for bicriteria case
Implementation: ongoing Master thesis of Stefan Erb

CH based Transit Node Routing

[Arz, Luxen, Sanders, SEA submission]

- Preprocessing time: mostly CH construction
- Purely graph theoretical locality filter
  (Graph Voronoi Filter)
- Faster than HH-based TNR
- Much less space/preprocessing than hub based routing

Alternative Routes

[recap]

two main approaches

- penalty method (alternative graphs) [Bader et al. 11]
- plateau method (via node alternatives) [Abraham et al. 10a]

relation to our research

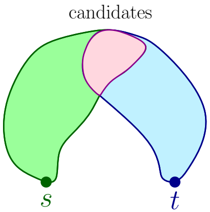
- Luxen, Schieferdecker:
  Candidate Sets for Alternatives (SEA 2012, JEA?)
- Kobitzsch:
  Hierarchy Implosion for Alternatives (SEA 2013?)

} plateau method via alternatives + graphs

- Kobitzsch, Radermacher, Schieferdecker:
  Fast Alternative Graphs with CRP (SEA 2013?)

} penalty method alternative graphs

Alternative Routes
[candidate sets method]

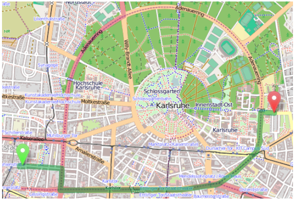


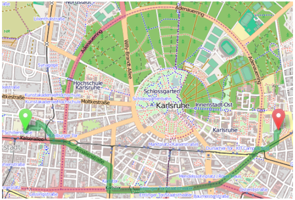
shortcoming of [Abraham et al. 10a]

- graph exploration + candidate evaluation costly
  $\rightarrow$ precompute sparse set of candidates to test

working assumption

- few shortest paths between two regions [Bast et al. 07, Abraham et al. 10b]
  $\rightarrow$ few good alternatives
  $\rightarrow$ covered by few via nodes



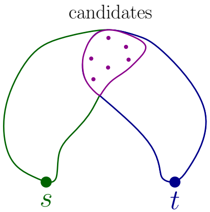


Alternative Routes

[candidate sets method]

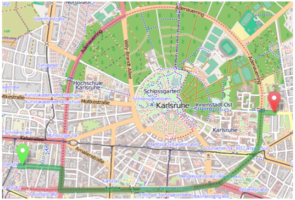
shortcoming of [Abraham et al. 10a]

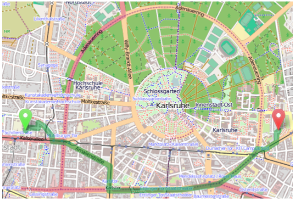
- graph exploration + candidate evaluation costly
  $\rightarrow$ precompute sparse set of candidates to test



working assumption

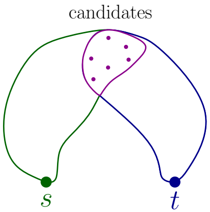
- few shortest paths between two regions [Bast et al. 07, Abraham et al. 10b]
  $\rightarrow$ few good alternatives
  $\rightarrow$ covered by few via nodes





Alternative Routes

[candidate sets method]

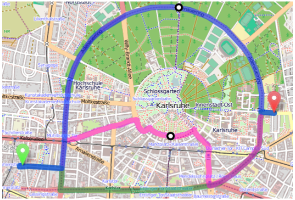


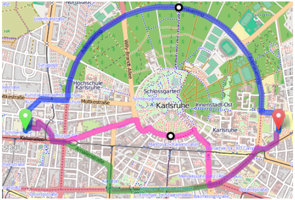
shortcoming of [Abraham et al. 10a]

- graph exploration + candidate evaluation costly
  $\rightarrow$ precompute sparse set of candidates to test

working assumption

- few shortest paths between two regions [Bast et al. 07, Abraham et al. 10b]
  $\rightarrow$ few good alternatives
  $\rightarrow$ covered by few via nodes





Alternative Routes

[candidate sets method]

approach

- partition graph into regions (single-, multi-level variant)
- compute via node candidate set for each region pair (bootstrapping)
  - alternative for each pair of border nodes with our query algorithm
  - not successful $\rightarrow$ new via node with baseline algorithm and add to set
- only evaluate these candidates during query

results

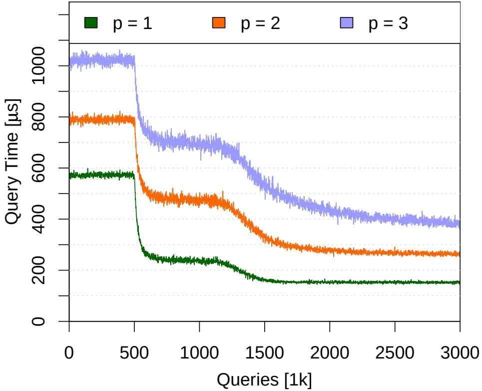
- alternatives in sub-milliseconds ($0.1ms$ – $0.3ms$ – $0.5ms$)
  $\rightarrow$ *more than* one order of magnitude faster than previous methods
- high success rates (90% – 70% – 44%)

Alternative Routes
[candidate sets method – applications]

Online Setting

- learn candidate sets on-the-fly
- applicable to legacy system
  (plus additional graph partitioning)

→ quick convergence
→ quality like offline variant



Alternative Routes

[candidate sets method – applications]

Alternative Graphs

- use full candidate sets to build reduced alternative graph

→ quick to compute: 1-2 ms

→ average quality: 2.5

(according to Bader et al.)

Alternative Routes

Hierarchy Implosion for higher Quality routes

Classic Via Routes

- low success rates in pruned search space
  - → relaxed search space for larger candidate selection
  - → relaxation adds more vertices with invalid distance
- each candidate to be evaluated by four point to point queries
  - → requires severe filtering methods to be practical
  - → filtering based on (incorrect) CH search space data

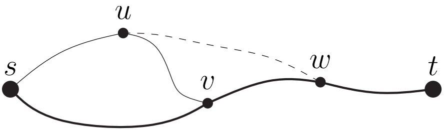
→ contradicting goals

New Approach

- clarify 2nd degree admissibility
- evaluate all possible candidates at once without additional queries

Alternative Routes

Hierarchy Implosion for higher Quality routes

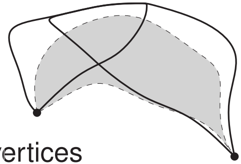


Admissibility

- $\mathcal{L}(P_{s,t} \cap P_{s,v,t}) \leq \gamma \cdot \mathcal{L}(P_{s,t})$ (limited sharing)
- $\forall v_i, v_j \in P_{s,t}, \mathcal{L}(\langle v_i, \ldots, v_j \rangle) \leq \alpha \cdot P_{s,t}$: $\mathcal{L}(\langle v_i, \ldots, v_j \rangle) = d(v_i, v_j)$ (local optimality)
- $\forall v_i, v_j \in P_{s,t} : \mathcal{L}(\langle v_i, \ldots, v_j \rangle) \leq (1 + \epsilon) d(v_i, v_j)$ (bounded stretch)

Alternative Routes

Hierarchy Implosion for higher Quality routes



Algorithm

- calculate forwards and backwards shortest path trees (localized PHAST)
  - → all necessary distance values
- perform hierarchy implosion to find important vertices
  - → plateaus eliminate necessity for T-test
- compress graph for small representation
  nearly for free if correct information generated during implosion

Results

- significantly higher quality, especially local optimality
- comparable runtime for first alternative
  compared to algorithms with similar preprocessing overhead
- further alternatives for free
- SEA submission

Alternative Graphs

Fast Implementation of the Penalty Method

Penalty Method

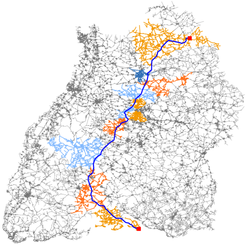
1. compute shortest path
2. possibly add to output
3. add penalties to path and connected arcs
4. repeat until satisfied

Problems

- highly dynamic
  - → requires Dijkstras algorithm (?)
  - → inherently slow (?)

Alternative Graphs

Fast Implementation of the Penalty Method



Bachelor Thesis: Marcel Radermacher

- CRP based implementation
  - → parallel updates
  - → fast query
- currently alternative graphs in 0.75 to 0.8 seconds on single i7
- ongoing efforts to speed calculation up even further
- still possibly SEA submission(?)

Route Corridors

Karlsruhe Institute of Technology

Idea

- allow for deviations from shortest path
- calculate all necessary shortest path distances in advance
- developed in context of hybrid scenario

  → also potential alternative route candidates (?)

Properties

- iterative construction possible (n-turn corridor)
  - size about doubles with each turn
- fast
  - below 100 ms even for six turns
- hard to leave corridor
  - random drive simulation gives high success rate

Route Corridors

Algorithm, Search Space Extraction and Sweep

Algorithm

- perform backwards search from *target*
- repeat until number of deviations reached
  1. determine necessary forward search space
  2. sweep calculated search space
  3. unpack and update

Dependency Sets

- compute reachable nodes with distance label not set
- bucket queue allows for in-order extraction
  $\rightarrow$ efficient due to small number of levels

Sweep

- sweep calculated node set in generated order
- set usually very small due to early pruning

Route Corridors

Algorithm, Search Space Extraction and Sweep

Algorithm

- perform backwards search from *target*
- repeat until number of deviations reached
  1. determine necessary forward search space
  2. sweep calculated search space
  3. unpack and update

Dependency Sets

- compute reachable nodes with distance label not set
- bucket queue allows for in-order extraction
  $\rightarrow$ efficient due to small number of levels

Sweep

- sweep calculated node set in generated order
- set usually very small due to early pruning

Route Corridors

Algorithm, Search Space Extraction and Sweep

Algorithm

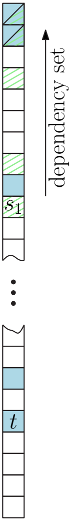
- perform backwards search from *target*
- repeat until number of deviations reached
  1. determine necessary forward search space
  2. sweep calculated search space
  3. unpack and update

Dependency Sets

- compute reachable nodes with distance label not set
- bucket queue allows for in-order extraction
  $\rightarrow$ efficient due to small number of levels

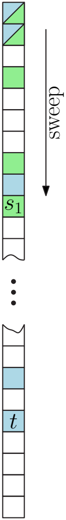
Sweep

- sweep calculated node set in generated order
- set usually very small due to early pruning



Route Corridors

Algorithm, Search Space Extraction and Sweep



Algorithm

- perform backwards search from *target*
- repeat until number of deviations reached
  1. determine necessary forward search space
  2. sweep calculated search space
  3. unpack and update

Dependency Sets

- compute reachable nodes with distance label not set
- bucket queue allows for in-order extraction
  $\rightarrow$ efficient due to small number of levels

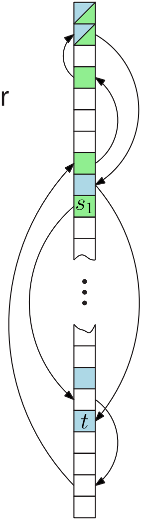
Sweep

- sweep calculated node set in generated order
- set usually very small due to early pruning

Route Corridors

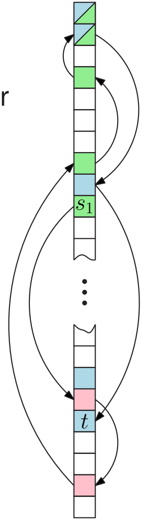
Unpacking and Update

- unpack shortest paths up to the first node within the corridor
  - keep corridor as small as possible
  - no equal sized paths whenever possible
  - avoid unpacking of identical information as much as possible
- set distance values along the path
  - sets lower level values $\rightarrow$ pruning for search space extraction
  - traversal necessary for deviation vertex calculation anyways
- remember new deviation vertices



Route Corridors
Unpacking and Update

- unpack shortest paths up to the first node within the corridor
  - keep corridor as small as possible
  - no equal sized paths whenever possible
  - avoid unpacking of identical information as much as possible
- set distance values along the path
  - sets lower level values $\rightarrow$ pruning for search space extraction
  - traversal necessary for deviation vertex calculation anyways
- remember new deviation vertices



SOTA

Stochastic on time arrival

Definition

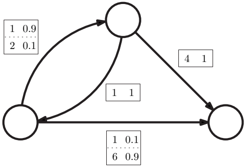
Given a graph $G = (V, A)$, $|V| = n$, $|A| = m$, for each $a \in A$ a probability distribution $p : \mathbb{N} \mapsto [0, 1]$, depicting the cumulative probability to traverse an arc in a given time $T \in \mathbb{N}$, as well as a time budget B and a source node s as well as a target node t:

Goal: Find the optimal strategy, starting at s, maximizing the probability of arriving at t within the budget B.

SOTA

Problems

- strategy depends on travel times experienced so far
  - → optimal strategy might contain loops
  - → computation has to look at each node multiple times
- each edge relaxation represents a costly convolution



Solved (Berkeley Results)

- optimal computation order
  - → reinserting Dijkstra
- zero delay convolutions for faster edge relaxations
  - → low overhead for updating cumulative distribution functions

SOTA

Status of ongoing research project

Algorithm

- ported JAVA implementation (Berkeley) to C++
  - tuned C++ implementation
  - handles larger inputs
  - faster

Next Goals

- extract meaningful distributions from your data
- evaluate alternative routes in comparison to plain SOTA
  - via routes
  - penalty method
  - corridors
- combine and adjust alternative techniques for efficient SOTA

  → extensible to other algorithms as well