# Path-based multi-commodity flows in large networks


Rolf H. Möhring

Heiko Schilling

Ekkehard Köhler, Katharina Langkau, Martin Skutella

Technische Universität Berlin

Combinatorial Optimization and Graph Algorithms

# Why path-based flow models?

Flow models in practice have complicated side constraints

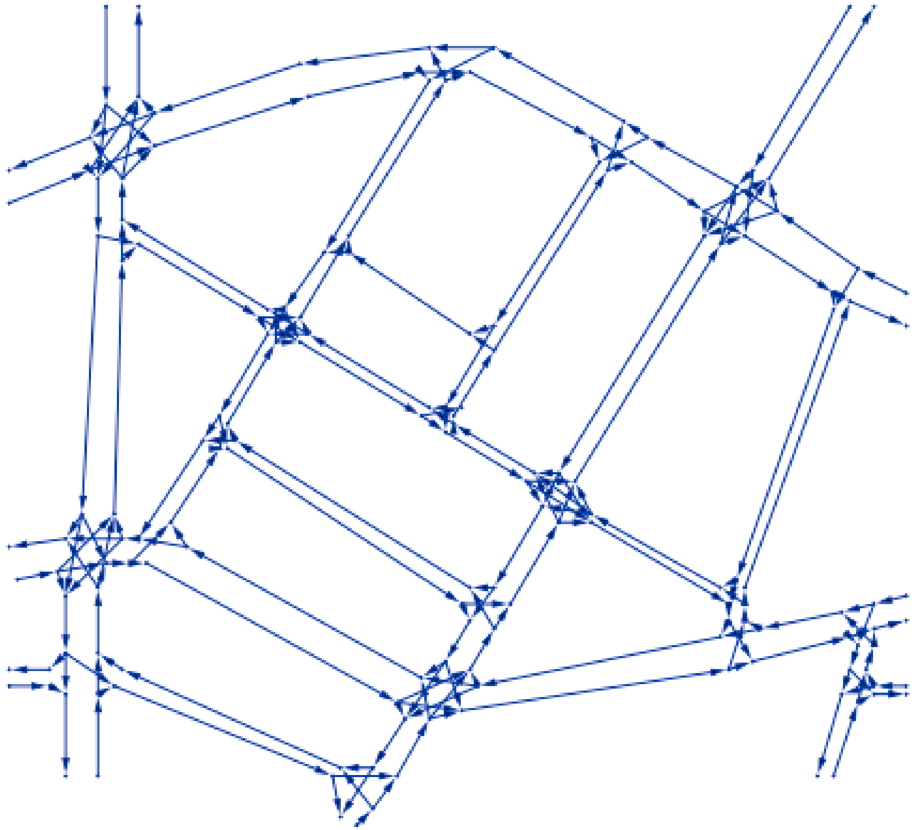
- individual path constraints
  - avoid turns and low level streets in traffic routes
  - restrict the length of routes (no long detours)
  - only way to describe flows over time
- global resource constraints
  - share the network as a common resource
  - path packing problems
- complicated evaluation functions
  - depends on paths (road pricing/track assignment)
  - non-linear in standard flow variables

# Overview

- An application: Traffic routing
- Project goals and working program
  - Static model for traffic flows
  - Fast basic algorithms for large networks
  - Algorithm engineering and software development
  - Dynamic models for traffic flows

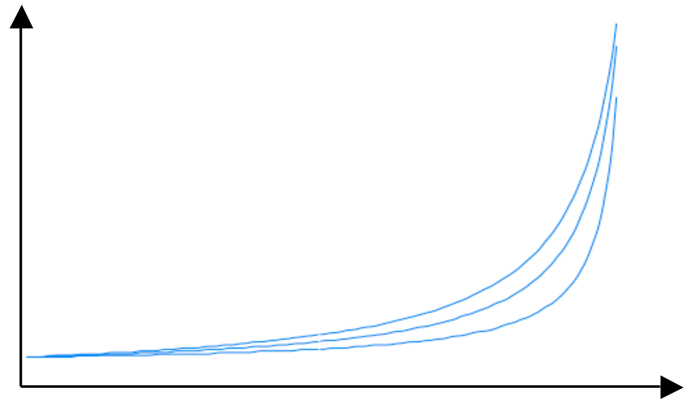
# Example: Traffic route guidance


a project with DaimlerChrysler



# Modeling congestion






Streets have capacities ... ... and traffic-dependent transit times

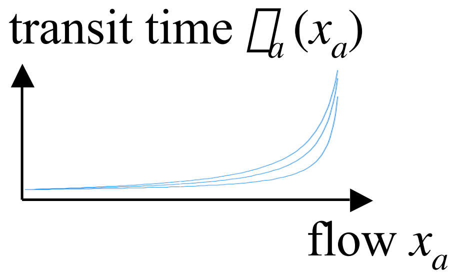
Minimize total travel time

# The underlying mathematical model

Static non-linear (fractional) multi-commodity flow model
Models rush hour traffic

- Street network (digraph) $G = (V, A)$

  Arcs $a \in A$ have a
  - capacity $u_a \geq 0$
  - geographic length $l_a \geq 0$
  - link delay function $\tau_a(x_a)$




- Demand: $k$ node pairs $(s_i, t_i)$ with demand $b_i \geq 0$

- Route the demand subject to the capacities such that

  total travel time $\sum_{a \in A} x_a \cdot \tau_a(x_a) \rightarrow \min$

# Selfish routing leads to user equilibrium

Users optimize independently and selfish $\Rightarrow$

- reach a Nash equilibrium (user equilibrium), i.e. nobody can improve his route just by himself

$\tau_p(x)$ = transit time of flow $x$ along path $p$

$\boxed{x \text{ is in Nash equilibrium} \quad \text{iff} \quad \tau_p(x) < \tau_q(x') \text{ for all paths } p, q \in P}$

transfer $\varepsilon$ flow units from path $p$ to path $q$

User equilibrium is not optimal and subject to unwanted effects

# Traffic management leads to system optimum

Centralized assignment of routes $\Rightarrow$

- reach the system optimum i.e.
  total travel time $\sum_{a \in A} x_a \cdot \tau_a(x_a)$ is minimum

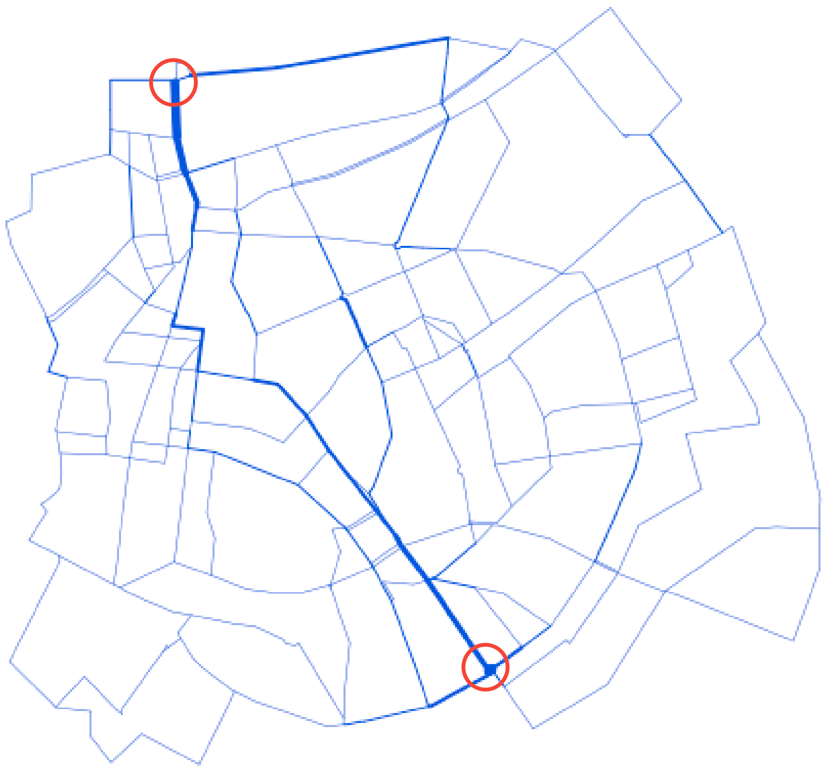
- Can be done in arc flow model, so standard methods available
- Beats user equilibrium Roughgarden & Tardos [STOC 2000]

But:

- Individual routes may be far too long

# Centralized assignment of routes

# Centralized assignment of routes



system optimum

restrict length of routes

# System optimum under length restriction

Requires path-oriented flow formulation

- $P_i$ set of paths from $s_i$ to $t_i$ that are "not too long"
  i.e., $l$ ( path ) $\leq (1+\varepsilon) \cdot l$ ( shortest path from $s_i$ to $t_i$ )
- $P = P_1 \cup \ldots \cup P_k$
- May represent flow $x$ as
  - vector $x^A$ of arc flow values or
  - vector $x^P$ of path flow values
- $\tau_p(x)$ = transit time of flow $x$ along path $p \in P$
- assign routes from restricted path set $P$ such that
  - all demands are satisfied
  - arc capacities are respected
  - total travel time $\sum_{a \in A} x_a \cdot \tau_a(x_a) = \sum_{p \in P} x_p \cdot \tau_p(x) \rightarrow \min$

# The optimization model

$\min\ (\square^{A}\ (\square\ x^{P}))^{T}\ \square\ x^{P}$

$\square$ = arc-path incidence matrix,
dimension $|A| \ \square\ |P|$ $\quad x^{A} = \square\ x^{P}$

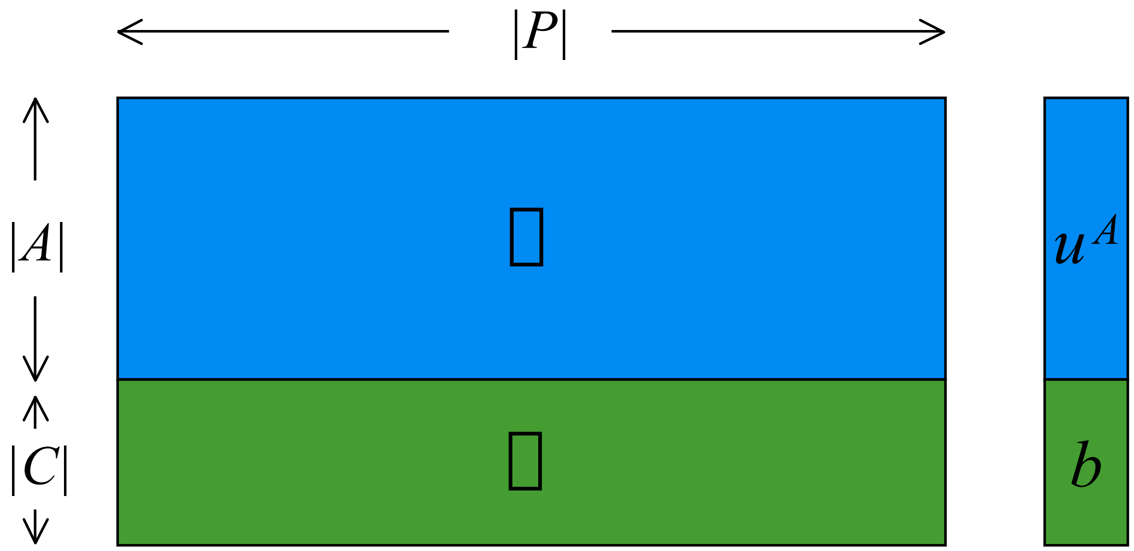
s.t. $\square\ x^{P} = b$

$\square$ = commodity-path incidence matrix
dimension $|C| \ \square\ |P|$ , $C = \{1,\ldots,k\}$

$\square\ x^{P} \leq u^{A}$

$x^{P} \geq 0$

$p \ \square\ P$

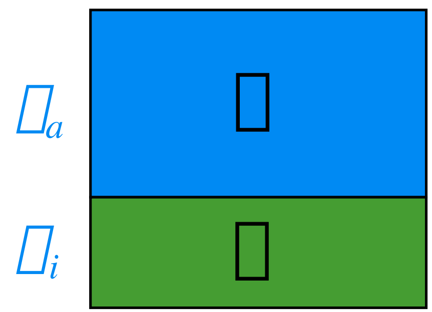



convex objective
linear constraints

# The linearized problem

Assume that transit times $\square^A$ = const

- Linear program with a huge number of variables $x_p$, $p \;\square\; P$
- Use revised simplex algorithm with column generation
  - solve LP only with few path variables $x_p$
    □ dual variable values $\square_a$, $\square_i$
  - optimality condition for whole LP is




$$\boxed{\square_{a \,\square\, p} (\square_a - \square_a) \geq \square_i \quad \text{for all } i \;\square\; C,\ p \;\square\; P_i}$$

□ constrained shortest path problem

find shortest path from $s_i$ to $t_i$ w.r.t. arc lengths $\square_a - \square_a$

such that $l$ ( path ) $\leq (1+\square) \cdot l$ ( shortest path from $s_i$ to $t_i$ )

# The constrained shortest path problem

$$\begin{aligned} &\min\ \square(p) \\ &\text{s.t.}\quad l(p) \leq L \\ &\qquad\ p \text{ is an } s-t \text{ path} \end{aligned}$$

- weakly NP-hard, there are full approximation schemes (of no practical use) [Warburton 1987]
- Branch & bound and Lagrangean relaxation [Beasley & Christofides1989]
- Labeling algorithm (Dijkstra-like) [Aneja, Aggarwal & Nair 1983]
- LP-guided combinatorial algorithms [Mehlhorn & Ziegelmann 2000]

# Main steps in the solution method

gradient method (Frank-Wolfe)

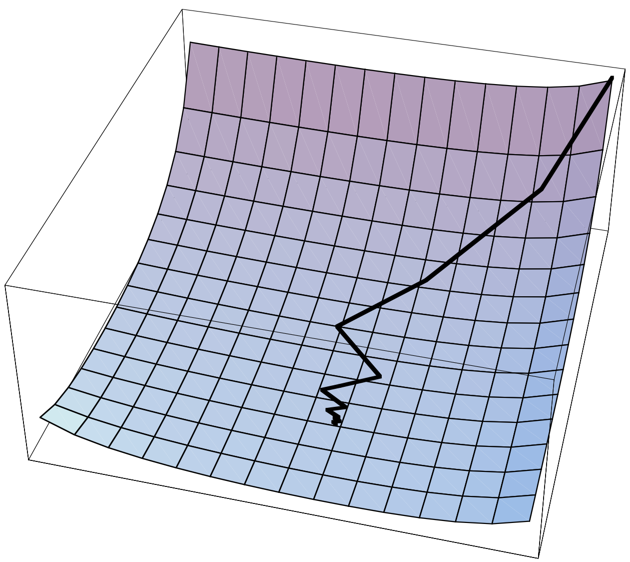
simplex algorithm

algorithm for constrained shortest paths

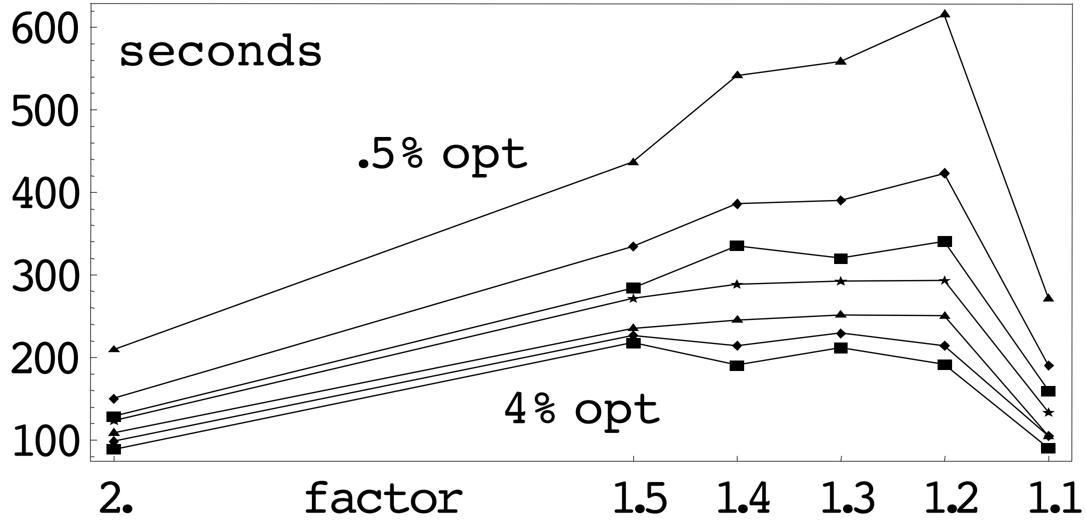
shortest path algorithm, e.g. Dijkstra

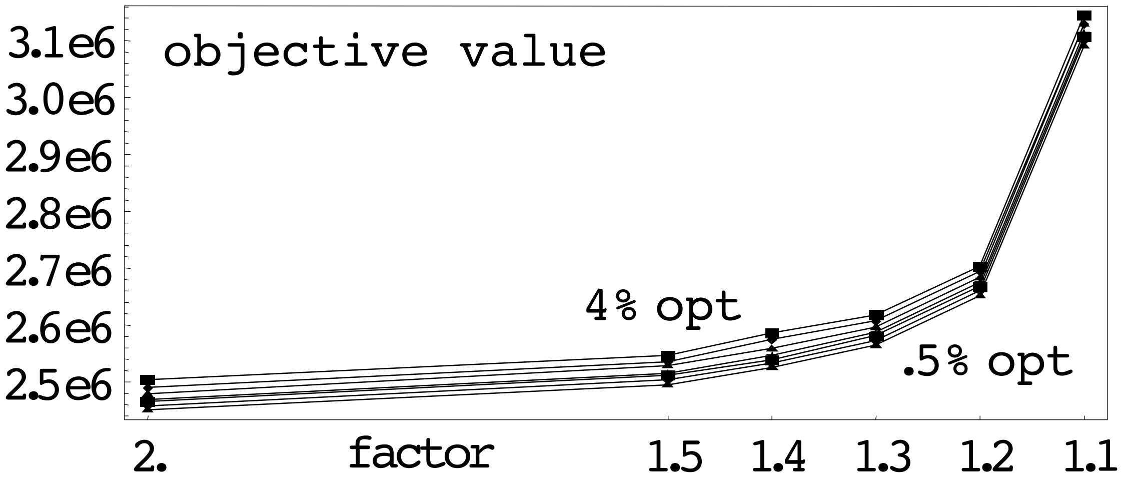
# General behavior

Instance REAL
|A| = 4040
|C| = 3166
Ultrasparc 333 MHz



50-200 iterations



# Project goals and working program

- **Static model for traffic flows**
  Evaluate quality of solution (compare to user equilibrium)
  Accelerate existing static algorithms (relax capacities, improve subalgorithms);
- **Fast basic algorithms for large networks**
  Design fast basic algorithms, in particular shortest path algorithms (acceleration methods, hierarchical, approximative)
- **Algorithm engineering and software development**
  Reuse of existing software, concepts for combining different libraries, data structures for flow related tasks
- **[Dynamic models for traffic flows]**
  Traffic is modeled as dynamic flow in a directed graph;
  travel times of arcs are time- and flow-dependent

# Measuring route guidance through (un)fairness

Unfairness of a route guidance strategy =

$$\max_{\boxed{\text{all OD pairs } i}} \frac{\text{max transit time } \square_p(x) \text{ of a used path } p \text{ for OD pair } i}{\text{min transit time } \square_p(x) \text{ of a used path } p \text{ for OD pair } i}$$

Unfairness ( user equilibrium ) = 1

Unfairness ( system optimum ) may be arbitrarily large

# Varying constraint factor

Instance REAL, Gap 0.5%, and geographic distances

| | | | | | Fairness | | |
|---|---|---|---|---|---|---|---|
| Factor | Value | # Iter. | # Paths | Sec. | geogr. | 0-flow | act. flow |
| $\infty$ | 2656640.05 | 214 | 10533 | 375 | 21.35 | 19.02 | 15.88 |
| 5.00 | 2657306.23 | 187 | 10370 | 291 | 4.885 | 5.881 | 5.835 |
| 2.00 | 2661115.89 | 255 | 10534 | 224 | 2 | 2.867 | 2.657 |
| 1.50 | 2782771.49 | 208 | 8952 | 376 | 1.5 | 1.963 | 1.927 |
| 1.40 | 2881963.13 | 203 | 8536 | 440 | 1.4 | 1.874 | 2.292 |
| 1.30 | 2988867.60 | 206 | 8152 | 435 | 1.3 | 1.782 | 2.463 |
| 1.20 | 3189798.45 | 217 | 7331 | 426 | 1.2 | 1.53 | 1.762 |
| 1.10 | 4108119.33 | 105 | 6051 | 181 | 1.1 | 1.53 | 2.811 |
| 1.05 | infeasible | | | | | | |
| UE | 2909704.41 | 72 | 6621 | 119 | 10.15 | 10.41 | |

# Make this approach scalable

- No use of Simplex algorithm
  - Lagrangian relaxation of capacity constraints
  - so far a factor of 2 faster
- Faster computation of constrained shortest paths
  - do standard speed-ups from shortest paths carry over?
- Exploit hierarchy and geography
  - natural in street networks

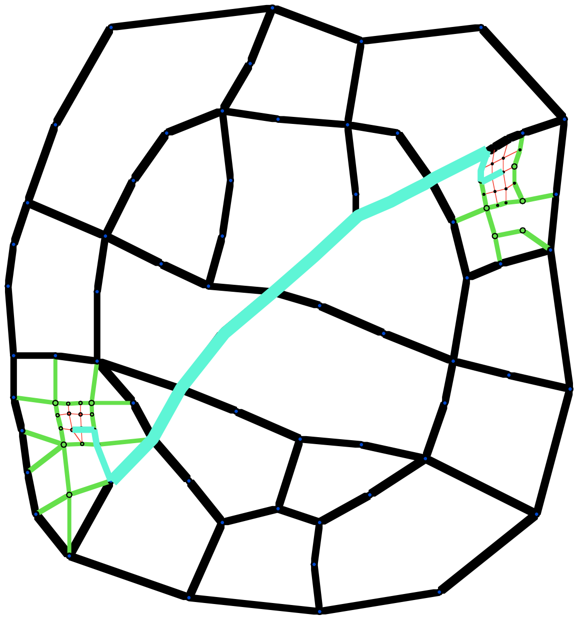
# Project goals and working program

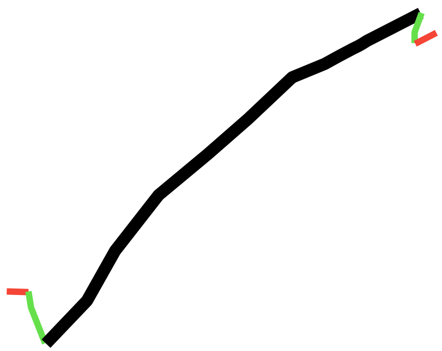
- **Static model for traffic flows**
  Evaluate quality of solution (compare to user equilibrium)
  Accelerate existing static algorithms (relax capacities, improve subalgorithms);
- **Fast basic algorithms for large networks**
  Design fast basic algorithms, in particular shortest path algorithms (acceleration methods, hierarchical, approximative)
- **Algorithm engineering and software development**
  Reuse of existing software, concepts for combining different libraries, data structures for flow related tasks
- **[Dynamic models for traffic flows]**
  Traffic is modeled as dynamic flow in a directed graph; travel times of arcs are time- and flow-dependent

# Shortest paths in large networks

- Hierarchy w.r.t. regions and street classes
- Hierarchy w.r.t. graph separators
- Preprocessing
- Acceleration methods

# Street-class approach

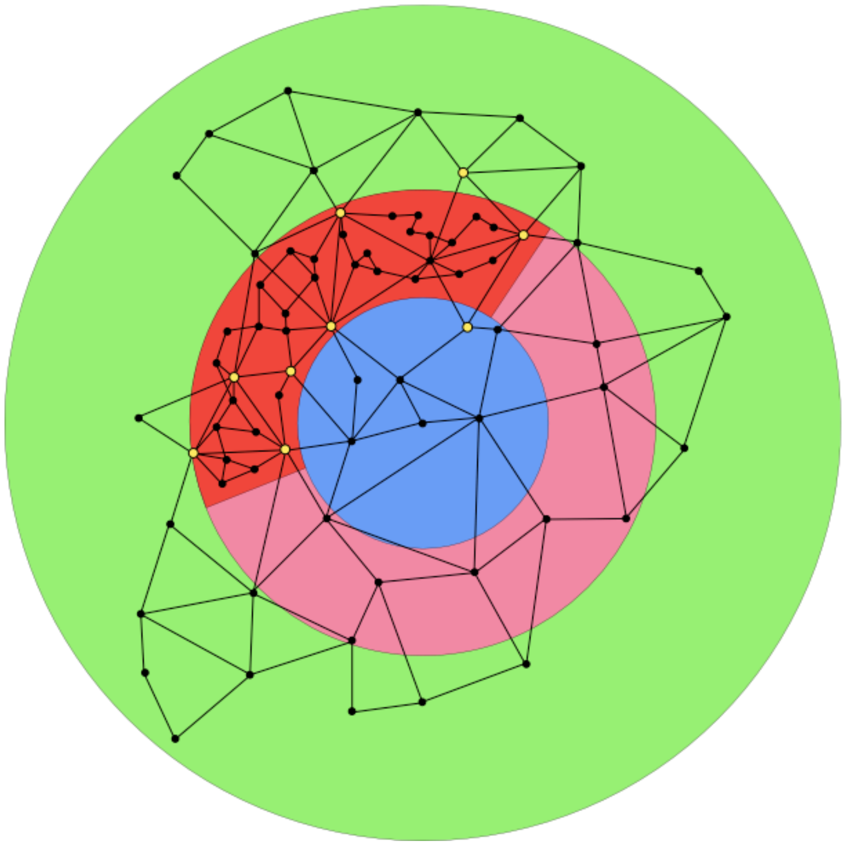


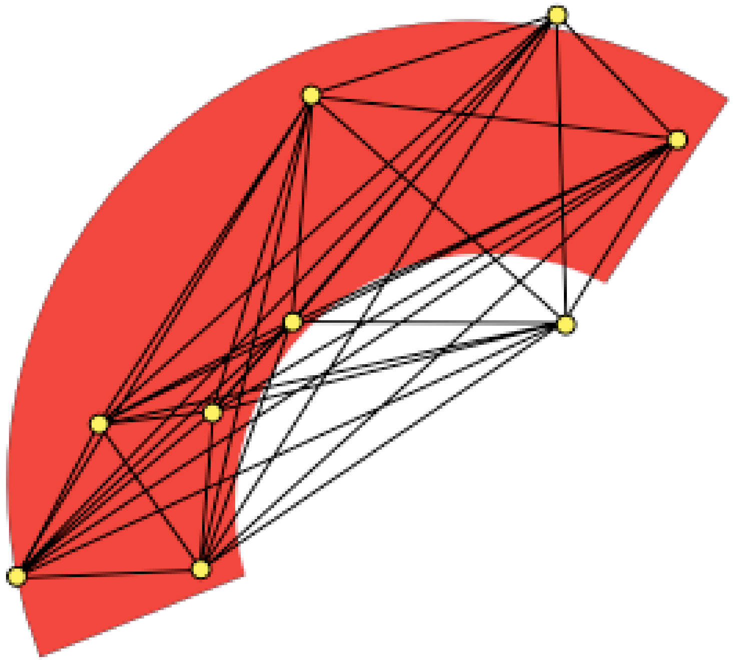


- Decompose the graph into a hierarchy of regions
- Search only for unimodal paths
  - street classes of arcs go first up and then down with the hierarchy
- First results
  - Run time ~ 60% of standard Dijkstra
  - Path lengths ~ 20% longer

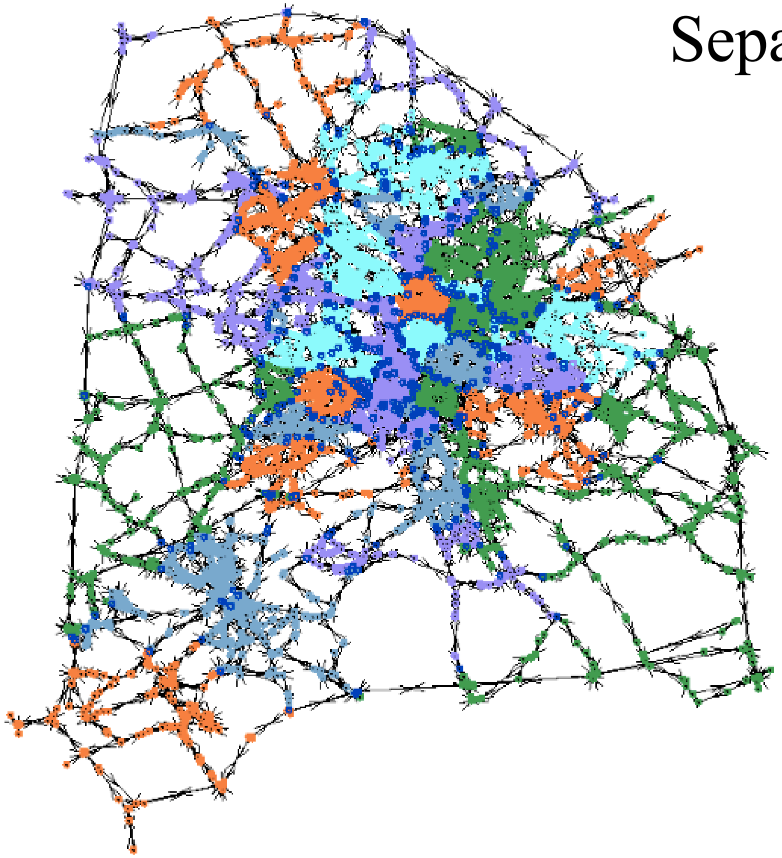
# Separator approach

- Find small graph separators that cut the graph into few regions
- Determine distance matrix for separator vertices of each region
- Compute shortest paths using the hierarchy induced by separators





# Separator approach



50% improvement on average

Separators constructed by simple heuristic

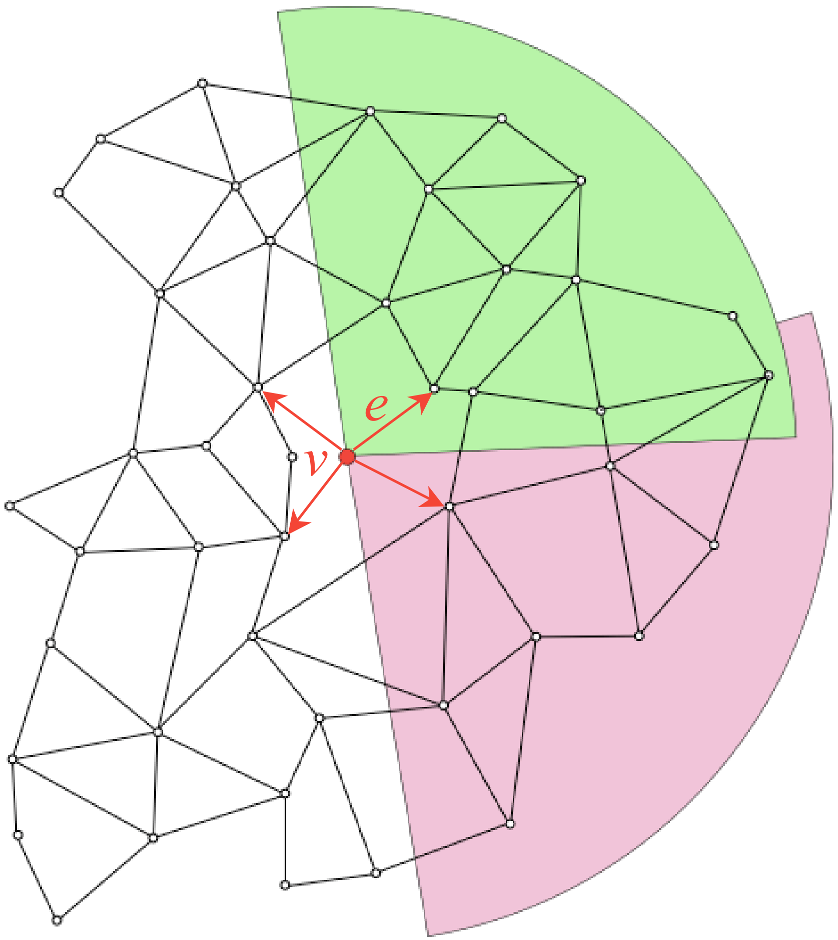
Graph 1:

- 12100 vertices and 19570 arcs
- 603 separator vertices
- 7691 additional arcs
- 54 regions
- 25 sec for preprocessing

Graph 2:

- 3810 vertices and 6027 arcs
- 231 separator vertices
- 2399 additional arcs
- 35 regions
- 2.3 sec for preprocessing

# Angle preprocessing

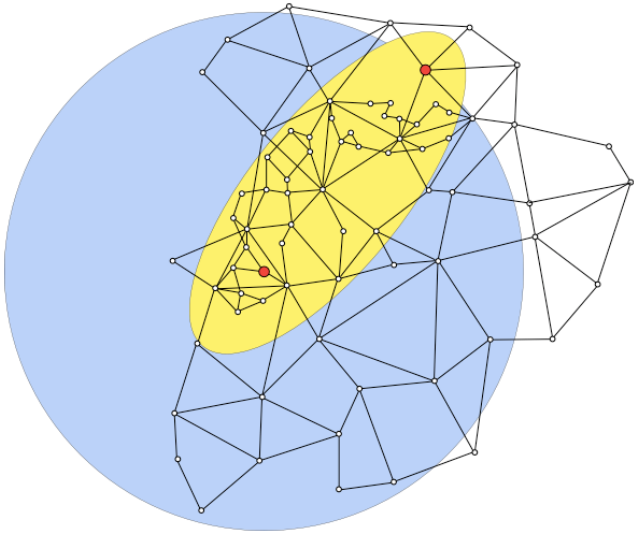



[Brandes, Schulz, Wagner & Wilhalm]

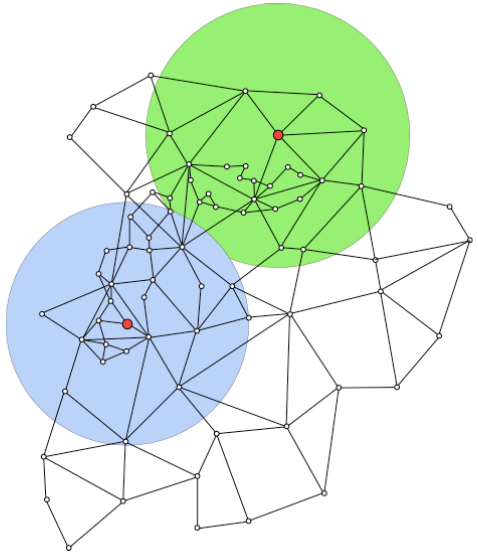
- For every vertex $v$ and edge $e$ leaving $v$, determine angle containing all vertices having $e$ on a shortest path from $v$.
- During algorithm:
  if target vertex not in angle of $e$, then "forget" $e$
- long preprocessing phase
  runtime ~ 40% of normal Dijkstra
- Combination of separator approach and angle preprocessing:
  ~ 25% of normal Dijkstra

# Constrained shortest paths: Acceleration

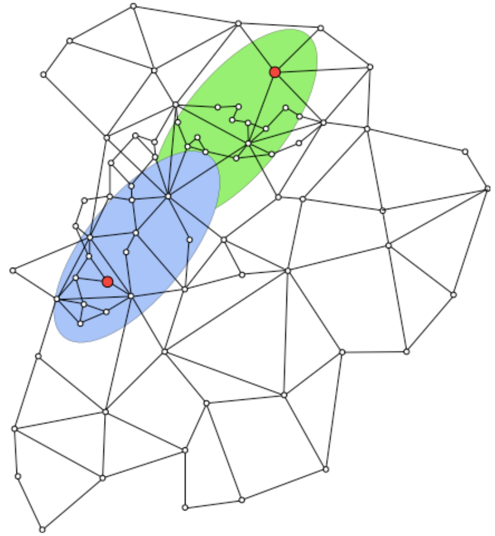
Destination oriented



Bi-directional



Combined



| fac | std | bi | bi-do | com |
|---|---|---|---|---|
| 1.50 | 1490 | 374 | 1.54 | 0.85 |
| 1.20 | 1471 | 390 | 9.89 | 3.47 |
| 1.10 | 1408 | 403 | 18.32 | 9.03 |
| 1.05 | 1435 | 430 | 10.38 | 4.92 |
| 1.00 | 1382 | 376 | 1.66 | 0.91 |

optimal

| fac | bi-do-br | do-br |
|---|---|---|
| 1.50 | 1.50 | 0.86 |
| 1.20 | 1.58 | 0.86 |
| 1.10 | 1.59 | 0.86 |
| 1.05 | 4.08 | 1.72 |
| 1.00 | 1.63 | 0.88 |

until first path found

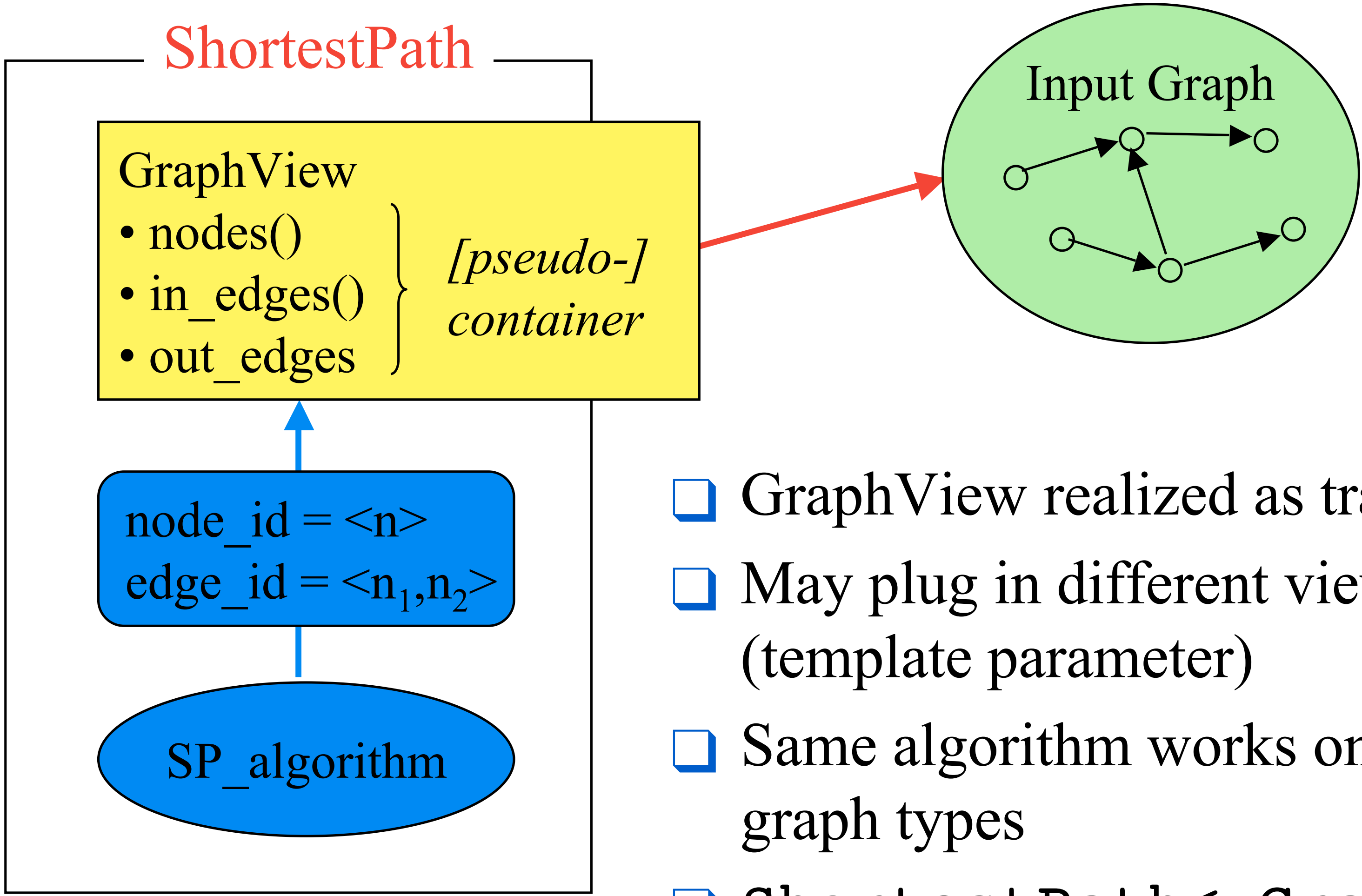
# Project goals and working program

- **Static model for traffic flows**
  Evaluate quality of solution (compare to user equilibrium)
  Accelerate existing static algorithms (relax capacities, improve subalgorithms);
- **Fast basic algorithms for large networks**
  Design fast basic algorithms, in particular shortest path algorithms (acceleration methods, hierarchical, approximative)
- **Algorithm engineering and software development**
  Reuse of existing software, concepts for combining different libraries, data structures for flow related tasks
- **[Dynamic models for traffic flows]**
  Traffic is modeled as dynamic flow in a directed graph; travel times of arcs are time- and flow-dependent

# Introduction

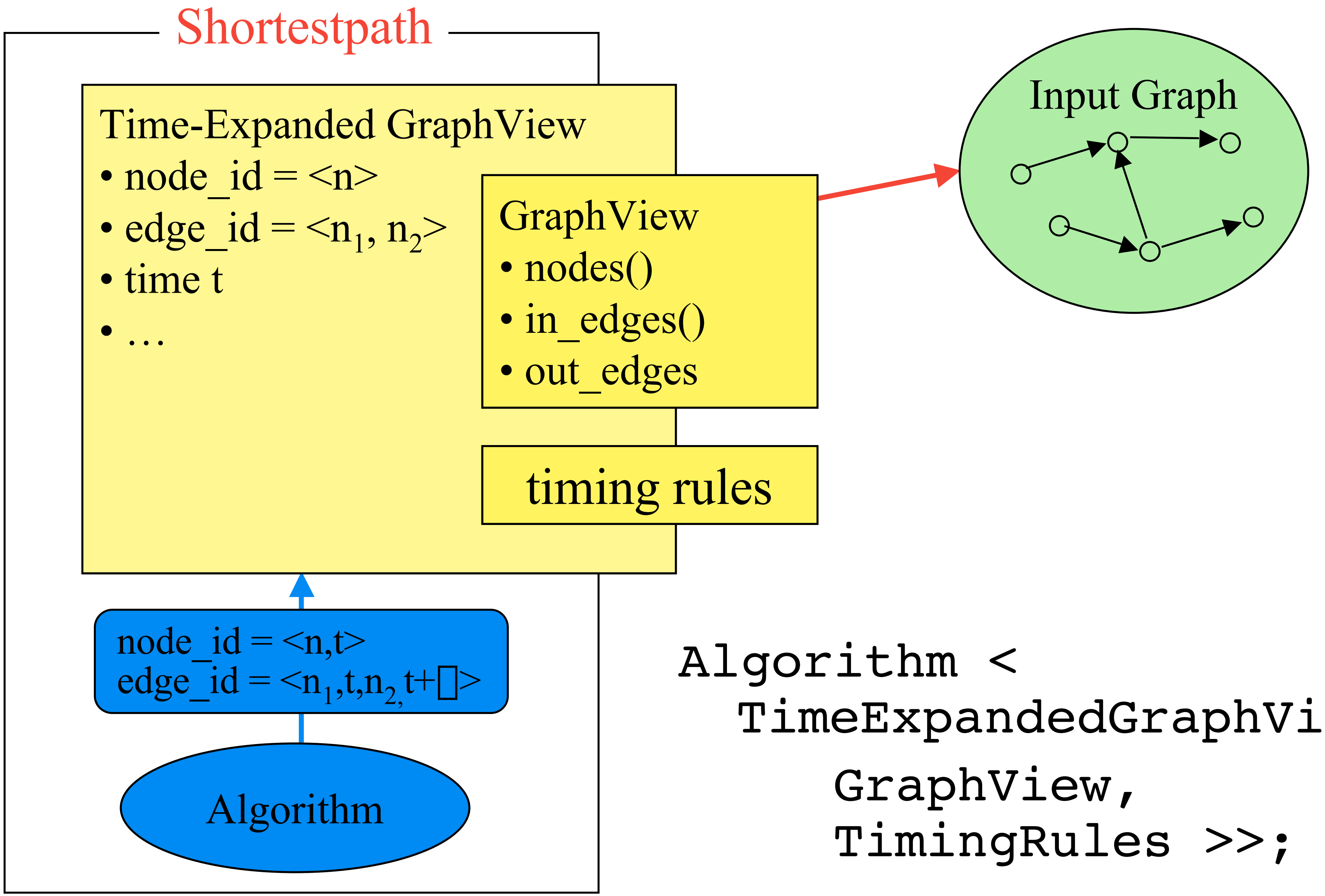
- Generic programming
  - separates data structures and algorithms
    via abstract requirement specifications
  - uses parametric polymorphism (templates) in C++
- Traits classes
  - provide mappings between types, functions and constants
    to meet specification/concept requirements
  - determine information about “unknown” types
  - “configure” templates

# GraphView



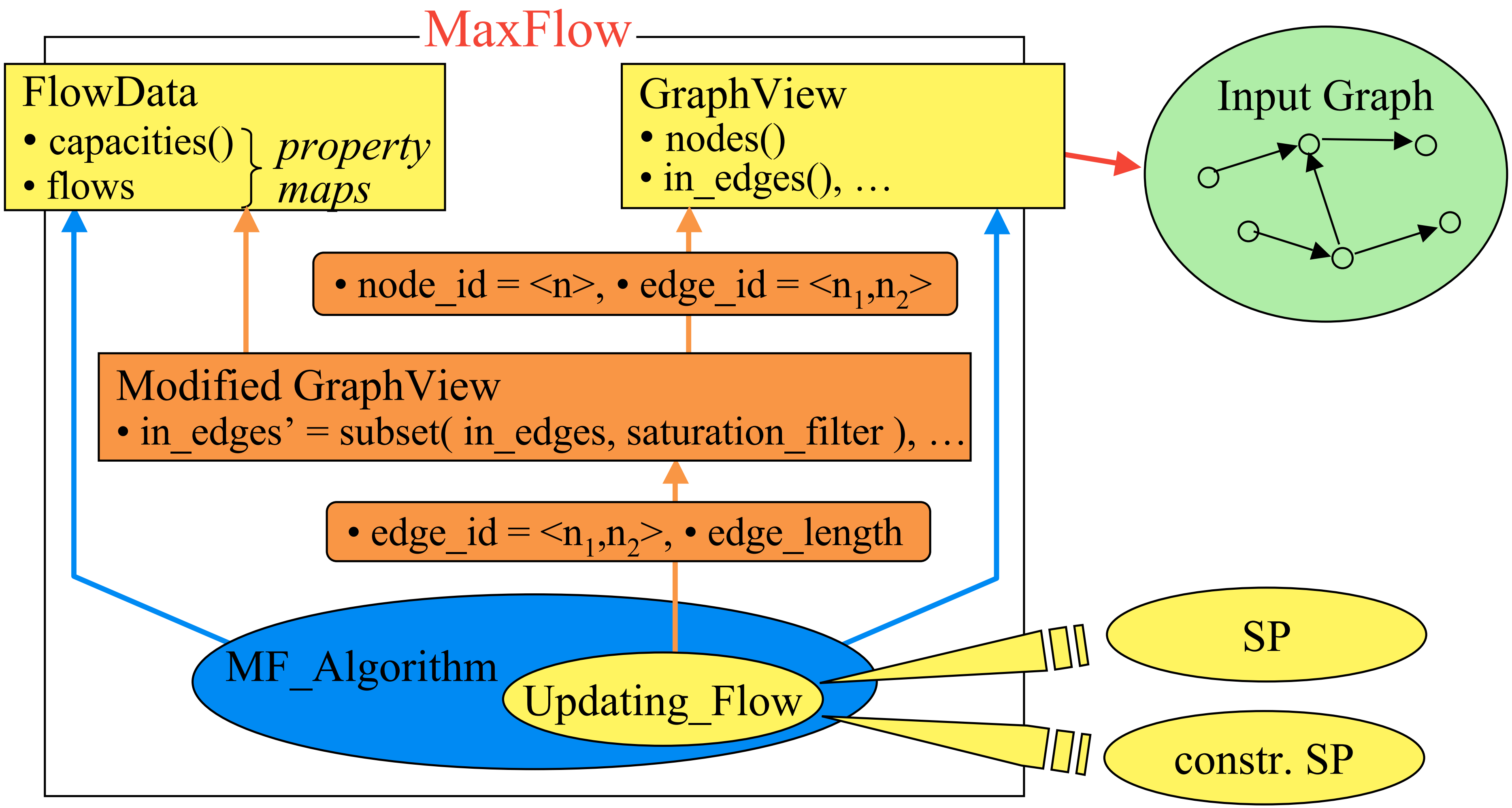

- GraphView realized as traits class
- May plug in different views (template parameter)
- Same algorithm works on different graph types
- `ShortestPath< GraphView >;`

# TimeExpanded GraphView




```
Algorithm <
  TimeExpandedGraphView<
     GraphView,
     TimingRules >>;
```

# Modified GraphView




```
MaxFlow< GraphView, FlowData,
         Updating_Flow_Builder = SP_Builder >;
```

# Algorithm generator

- Multicommodity flow algorithm [Garg & Könemann 1998]: plug in different updating flow algorithms
- Problem: sub-algorithm object needs access to data in main algorithm object, but cannot refer to partially created objects
- Solution: reverse_cast from Instancevariable to main object (e.g. from updating flow module to maxflow algorithm object) (uses pointer offsets in heap allocation table)
- Thus access to data within main object at run time