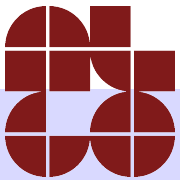
Combination of Speed-Up Techniques

for Dijkstra's Algorithm

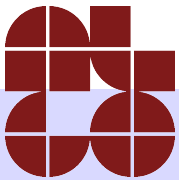
Dennis Schieferdecker (schiefer@ira.uka.de)

ITI Sanders, Universtiy of Karlsruhe (TH)

05.02.2009



- I -
Introduction
speed-up techniques



Motivation

dijkstra's algorithm

Common Problems

- finding connections in a network
- finding provably correct shortest paths

Definition

- given graph $G = (V, E)$ with weights $w : E \to (R)$ and two nodes $s, t \in V$, find a shortest path from s to t in G
- solved by Dijkstra's algorithm (1959)

Problem

- Dijkstra's algorithm is very slow (several seconds on large networks)

$\hookrightarrow$ Solution: speed-up techniques

Motivation

dijkstra's algorithm

Common Problems

- finding connections in a network
- finding provably correct shortest paths

Definition

- given graph $G = (V, E)$ with weights $w : E \to (R)$ and two nodes $s, t \in V$, find a shortest path from s to t in G
- solved by Dijkstra's algorithm (1959)

Problem

- Dijkstra's algorithm is very slow (several seconds on large networks)

$\hookrightarrow$ Solution: speed-up techniques

Motivation

dijkstra's algorithm

Common Problems

- finding connections in a network
- finding provably correct shortest paths

Definition

- given graph $G = (V, E)$ with weights $w : E \to (R)$ and two nodes $s, t \in V$, find a shortest path from s to t in G
- solved by Dijkstra's algorithm (1959)

Problem

- Dijkstra's algorithm is very slow (several seconds on large networks)

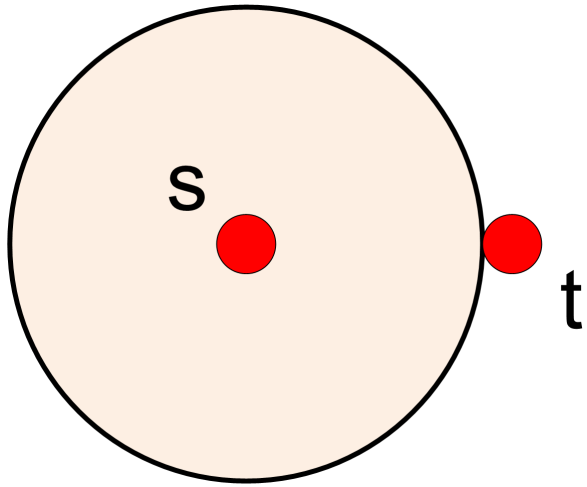
$\hookrightarrow$ Solution: **speed-up techniques**

Speed-Up Techniques

basics

Basic Idea

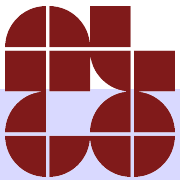
- reduce search space of Dijkstra's algorithm
- three strategies:
  - prune unimportant edges
  - visit promising nodes first
  - introduce shortcuts



Common Setup: Two-Step Algorithm

- preprocess data: compute additional information
- perform query: use this information to speed-up the search

$\hookrightarrow$ transformation: online-effort $\rightarrow$ preprocessing time + space consumption

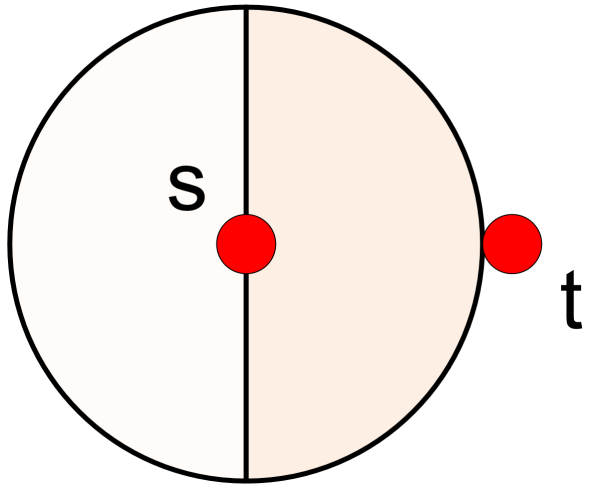


Speed-Up Techniques

basics

Basic Idea

- reduce search space of Dijkstra's algorithm
- three strategies:
  - prune unimportant edges
  - visit promising nodes first
  - introduce shortcuts



Common Setup: Two-Step Algorithm

- preprocess data: compute additional information
- perform query: use this information to speed-up the search

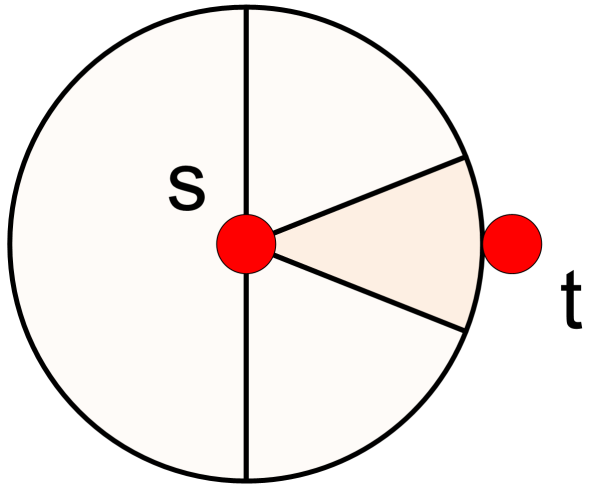
$\hookrightarrow$ transformation: online-effort $\rightarrow$ preprocessing time + space consumption

Speed-Up Techniques

basics

Basic Idea

- reduce search space of Dijkstra's algorithm
- three strategies:
  - prune unimportant edges
  - visit promising nodes first
  - introduce shortcuts



Common Setup: Two-Step Algorithm

- preprocess data: compute additional information
- perform query: use this information to speed-up the search

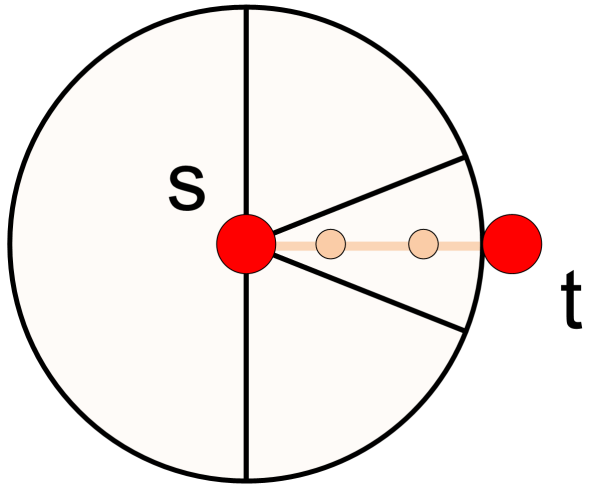
$\hookrightarrow$ transformation: online-effort $\rightarrow$ preprocessing time + space consumption

Speed-Up Techniques

basics

Basic Idea

- reduce search space of Dijkstra's algorithm
- three strategies:
  - prune unimportant edges
  - visit promising nodes first
  - introduce shortcuts



Common Setup: Two-Step Algorithm

- preprocess data: compute additional information
- perform query: use this information to speed-up the search

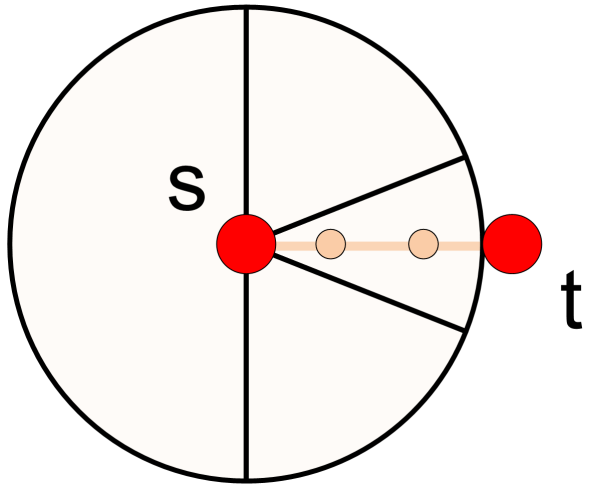
$\hookrightarrow$ transformation: online-effort $\rightarrow$ preprocessing time + space consumption

Speed-Up Techniques

basics

Basic Idea

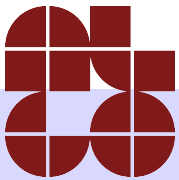
- reduce search space of Dijkstra's algorithm
- three strategies:
  - prune unimportant edges
  - visit promising nodes first
  - introduce shortcuts



Common Setup: Two-Step Algorithm

- preprocess data: compute additional information
- perform query: use this information to speed-up the search

$\hookrightarrow$ transformation: online-effort $\rightarrow$ preprocessing time + space consumption

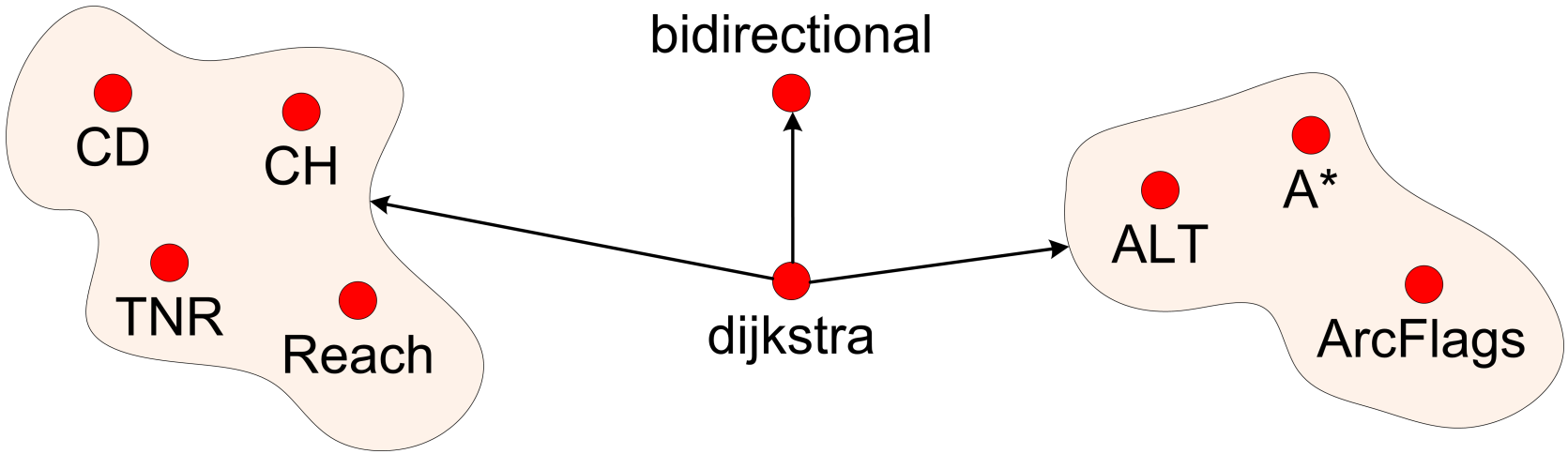


Speed-Up Techniques

algorithms

Types of Algorithms

- bidirectional search
- goal-directed search
- hierarchical search



Speed-Up Techniques

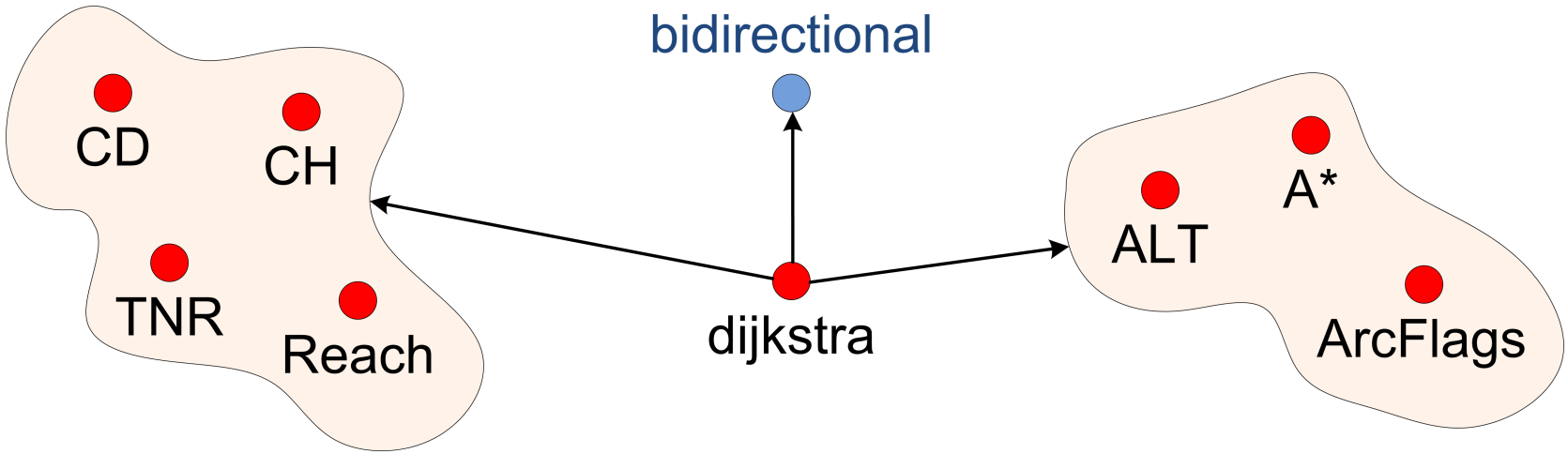
algorithms

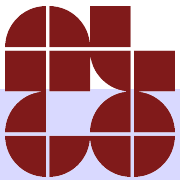
Types of Algorithms

- bidirectional search
- goal-directed search
- hierarchical search

Bidirectional Search

- start search from s and t
- perform query alternating in both directions, until search spaces meet





Speed-Up Techniques

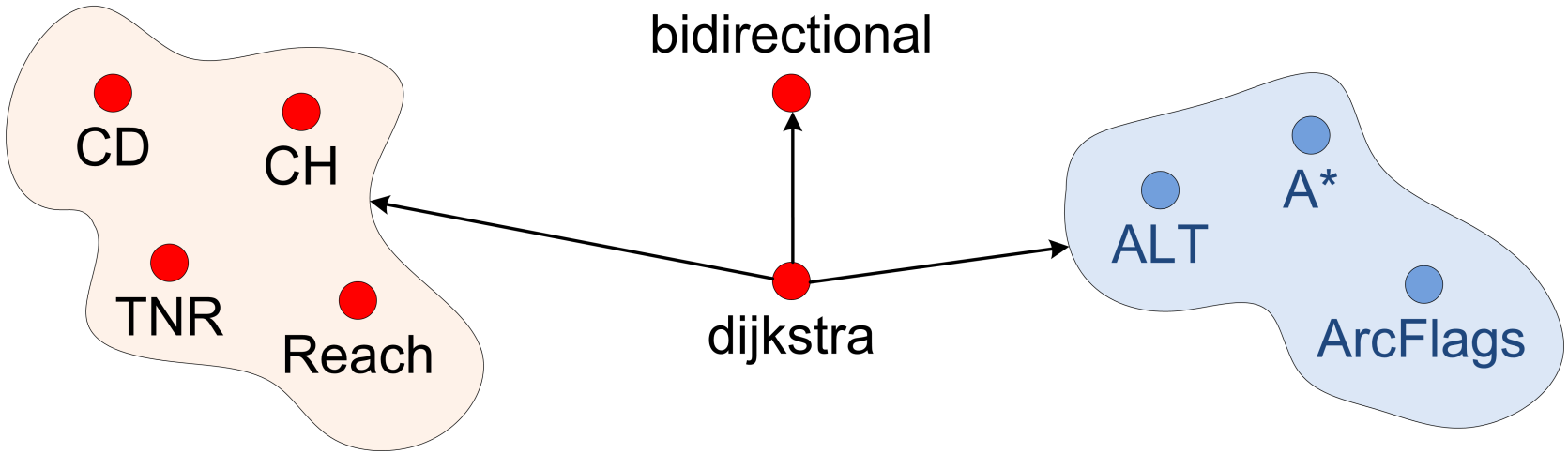
algorithms

Types of Algorithms

- bidirectional search
- goal-directed search
- hierarchical search

Goal-Directed Search

- push search in direction of t
- do not visit unimportant nodes or visit them later



Speed-Up Techniques

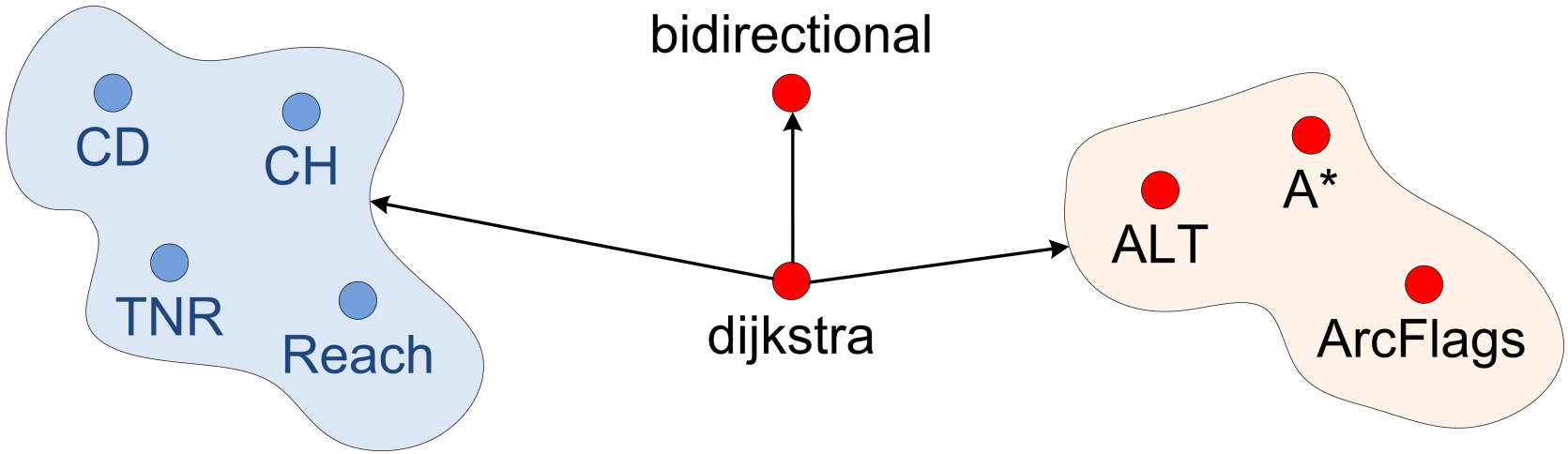
algorithms

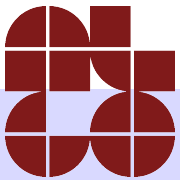
Types of Algorithms

- bidirectional search
- goal-directed search
- hierarchical search

Hierarchical Search

- organize graph in levels
- propagate search to the highest possible level and perform search there





ArcFlags

goal-directed approaches

basic idea

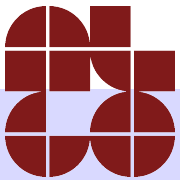
- partition graph into regions
- add labels to indicate shortest paths into regions
- apply **Dijkstra** with respect to labels

pro

- small overhead
- very fast long-range queries

contra

- long preprocessing times
- slow queries within a region



ArcFlags

goal-directed approaches

basic idea

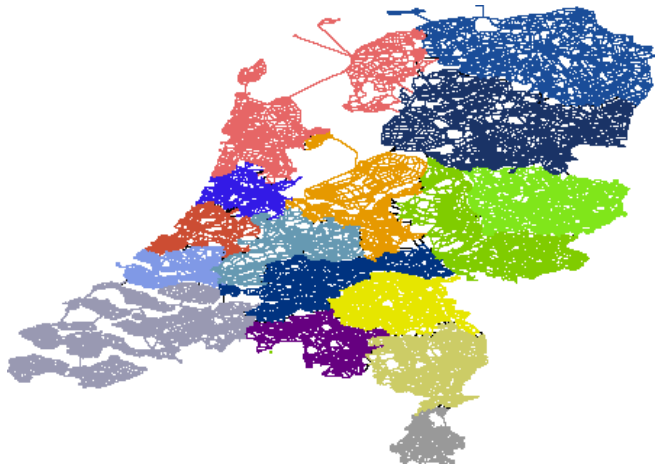
- partition graph into regions
- add labels to indicate shortest paths into regions
- apply **Dijkstra** with respect to labels

pro

- small overhead
- very fast long-range queries

contra

- long preprocessing times
- slow queries within a region



ArcFlags

goal-directed approaches

basic idea

- partition graph into regions
- add labels to indicate shortest paths into regions
- apply **Dijkstra** with respect to labels

pro

- small overhead
- very fast long-range queries

contra

- long preprocessing times
- slow queries within a region

ArcFlags

goal-directed approaches

basic idea

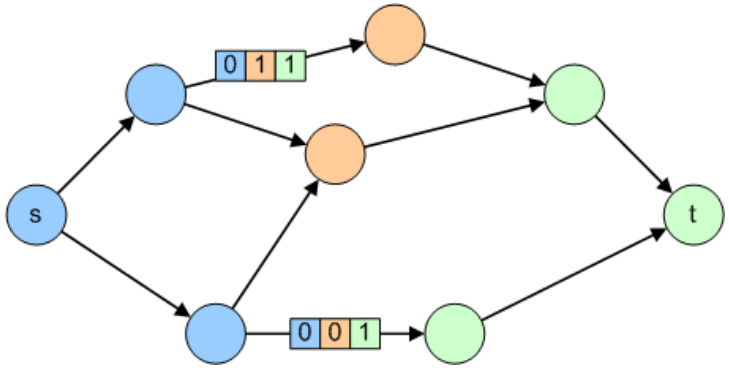
- partition graph into regions
- add labels to indicate shortest paths into regions
- apply **Dijkstra** with respect to labels

pro

- small overhead
- very fast long-range queries

contra

- long preprocessing times
- slow queries within a region



ArcFlags

goal-directed approaches

basic idea

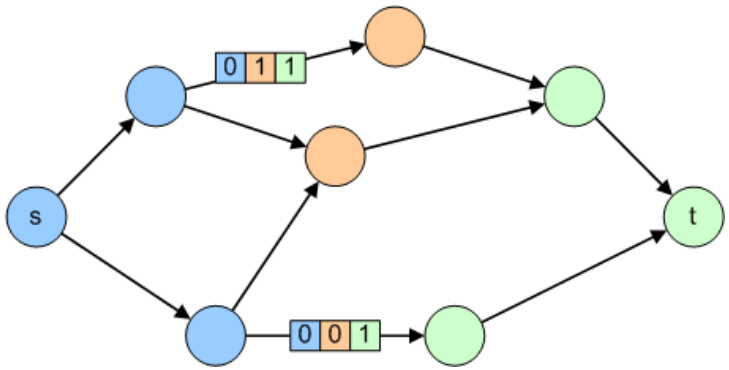
- partition graph into regions
- add labels to indicate shortest paths into regions
- apply **Dijkstra** with respect to labels

pro

- small overhead
- very fast long-range queries

contra

- long preprocessing times
- slow queries within a region



ArcFlags

goal-directed approaches

basic idea

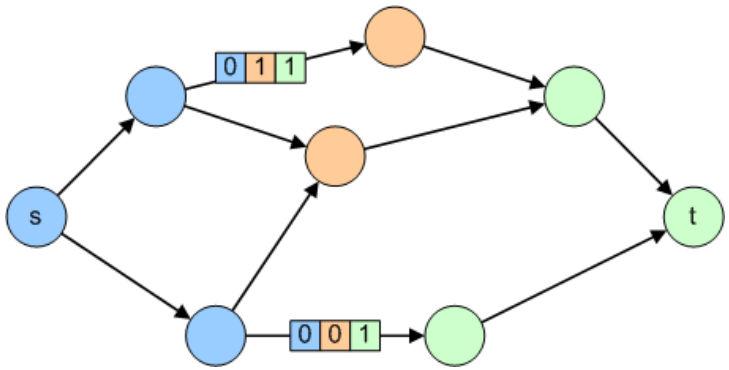
- partition graph into regions
- add labels to indicate shortest paths into regions
- apply **Dijkstra** with respect to labels

pro

- small overhead
- very fast long-range queries

contra

- long preprocessing times
- slow queries within a region



A*, ALT

goal-directed approaches

basic idea

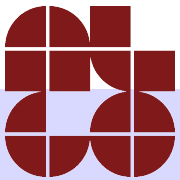
- add potential function $\pi(u)$ to graph nodes
  $\hookrightarrow$ lower bound on distance $u \longrightarrow t$
- apply **Dijkstra**, choose path with the shortest lower bound on the overall distance to proceed

pro

- fast preprocessing
- fast queries

contra

- large overhead



A*, ALT

goal-directed approaches

basic idea

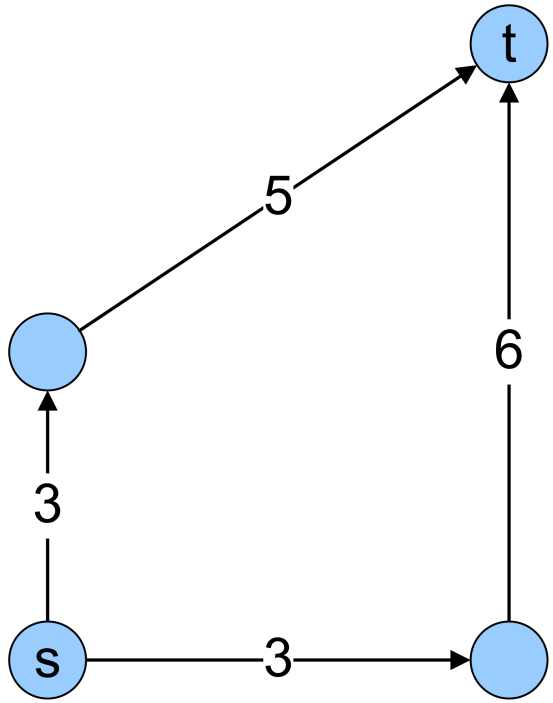
- add potential function $\pi(u)$ to graph nodes
  $\hookrightarrow$ lower bound on distance $u \longrightarrow t$
- apply **Dijkstra**, choose path with the shortest lower bound on the overall distance to proceed

pro

- fast preprocessing
- fast queries

contra

- large overhead



A*, ALT

goal-directed approaches

basic idea

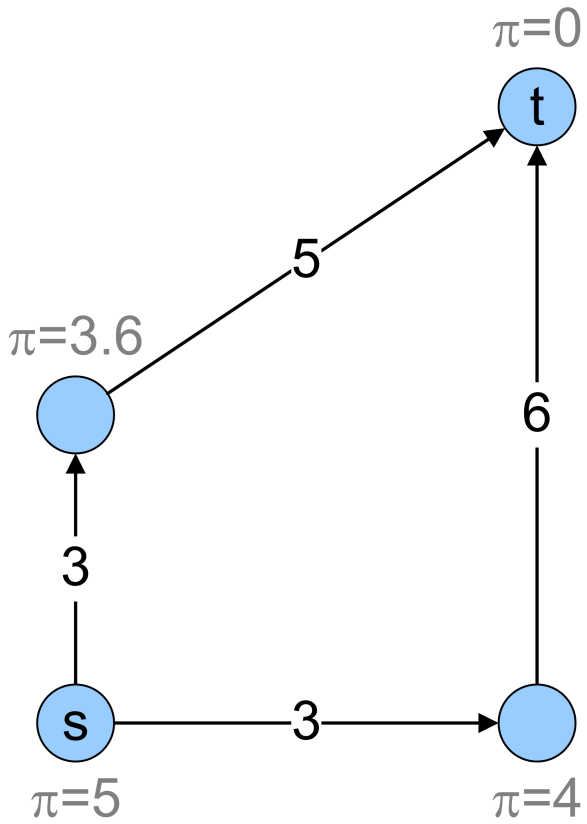
- add potential function $\pi(u)$ to graph nodes
  $\hookrightarrow$ lower bound on distance $u \rightarrow t$
- apply **Dijkstra**, choose path with the shortest lower bound on the overall distance to proceed

pro

- fast preprocessing
- fast queries

contra

- large overhead



A*, ALT

goal-directed approaches

basic idea

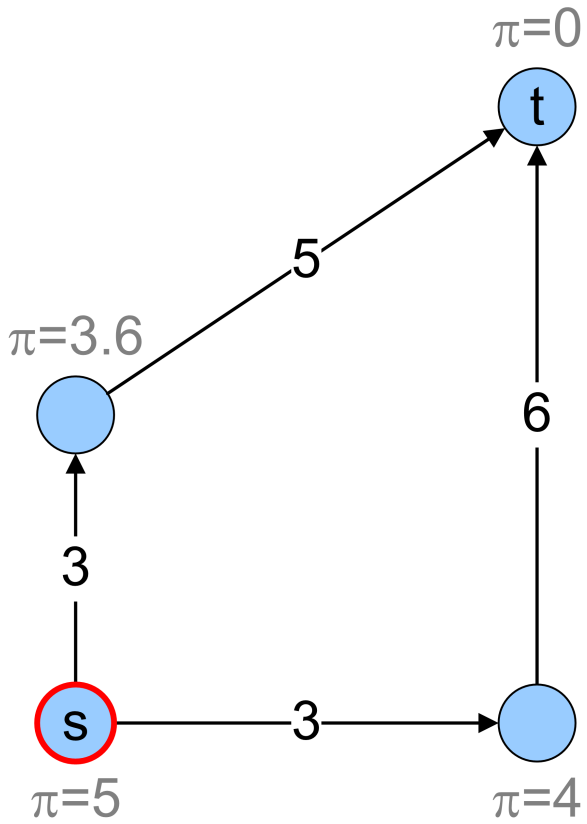
- add potential function $\pi(u)$ to graph nodes
  $\hookrightarrow$ lower bound on distance $u \rightarrow t$
- apply **Dijkstra**, choose path with the shortest lower bound on the overall distance to proceed

pro

- fast preprocessing
- fast queries

contra

- large overhead



A*, ALT

goal-directed approaches

basic idea

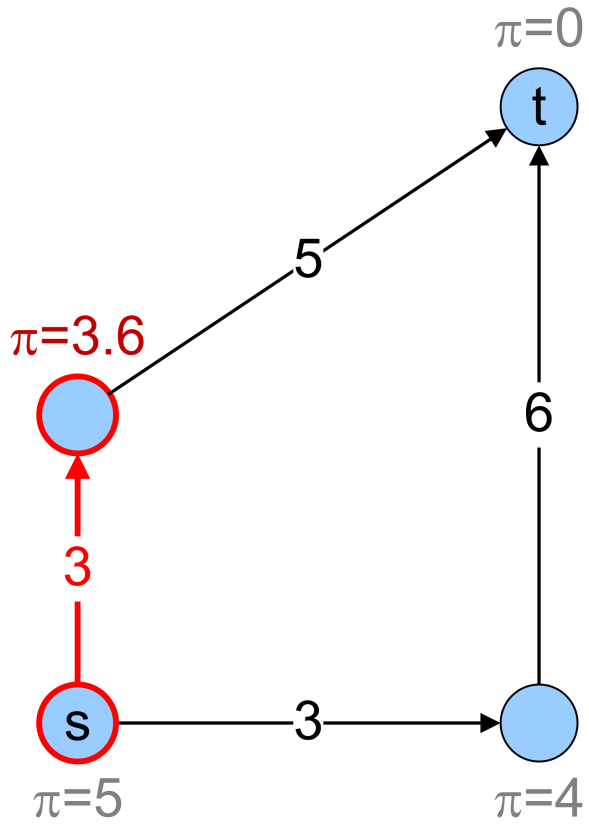
- add potential function $\pi(u)$ to graph nodes
  $\hookrightarrow$ lower bound on distance $u \rightarrow t$
- apply **Dijkstra**, choose path with the shortest lower bound on the overall distance to proceed

pro

- fast preprocessing
- fast queries

contra

- large overhead



A*, ALT

goal-directed approaches

basic idea

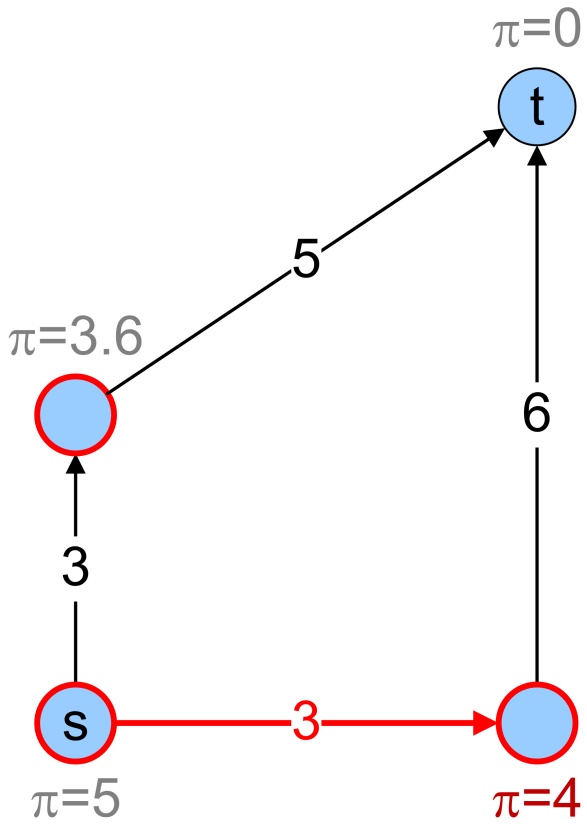
- add potential function $\pi(u)$ to graph nodes
  $\hookrightarrow$ lower bound on distance $u \longrightarrow t$
- apply **Dijkstra**, choose path with the shortest lower bound on the overall distance to proceed

pro

- fast preprocessing
- fast queries

contra

- large overhead



A*, ALT

goal-directed approaches

basic idea

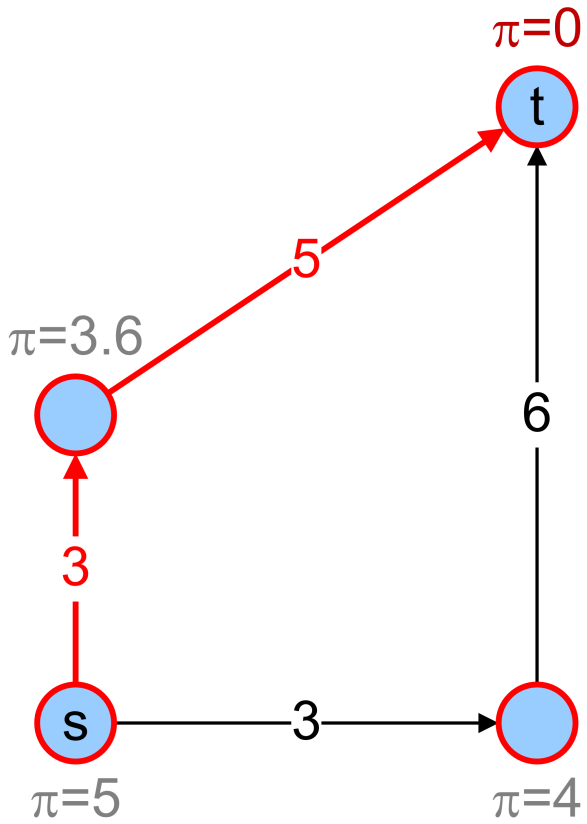
- add potential function $\pi(u)$ to graph nodes
  $\hookrightarrow$ lower bound on distance $u \longrightarrow t$
- apply **Dijkstra**, choose path with the shortest lower bound on the overall distance to proceed

pro

- fast preprocessing
- fast queries

contra

- large overhead



A*, ALT

goal-directed approaches

basic idea

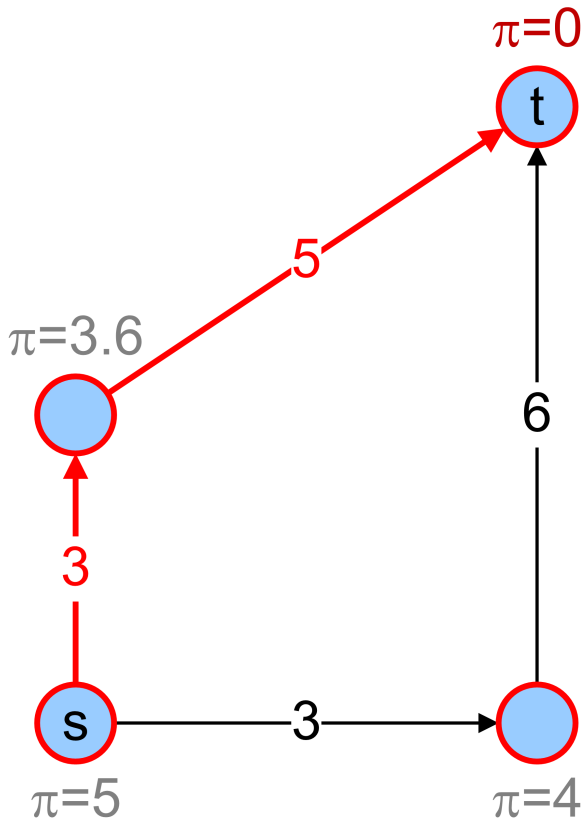
- add potential function $\pi(u)$ to graph nodes
  $\hookrightarrow$ lower bound on distance $u \longrightarrow t$
- apply **Dijkstra**, choose path with the shortest lower bound on the overall distance to proceed

pro

- fast preprocessing
- fast queries

contra

- large overhead



A*, ALT

goal-directed approaches

basic idea

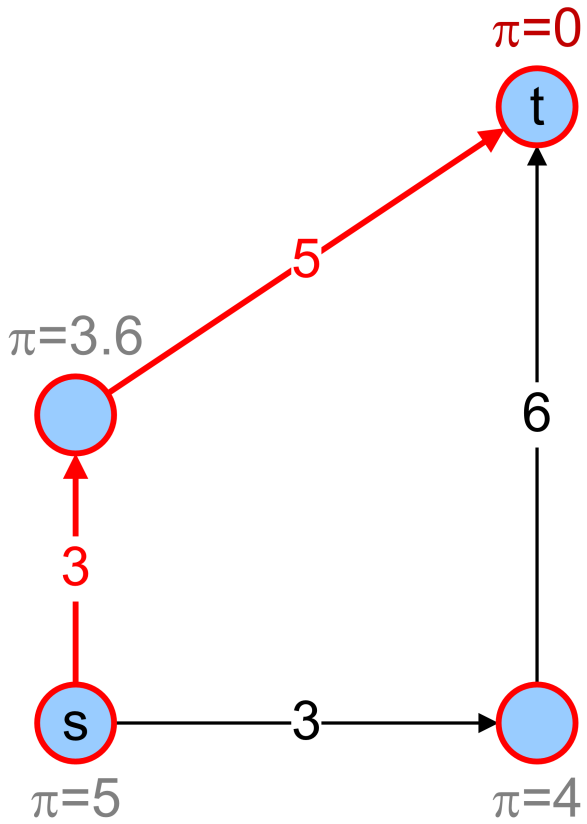
- add potential function $\pi(u)$ to graph nodes
  $\hookrightarrow$ lower bound on distance $u \longrightarrow t$
- apply **Dijkstra**, choose path with the shortest lower bound on the overall distance to proceed

pro

- fast preprocessing
- fast queries

contra

- large overhead



Highway / Contraction Hierarchies

hierarchy-aware approaches

basic idea

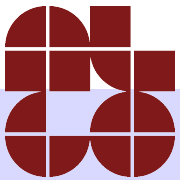
- organize graph into layers, and introduce shortcuts
- apply **Dijkstra**, never take edges to lower layers

pro

- fast preprocessing
- fast queries

contra

- bidirectional query mandatory
- graph needs inherent hierarchy



Highway / Contraction Hierarchies

hierarchy-aware approaches

basic idea

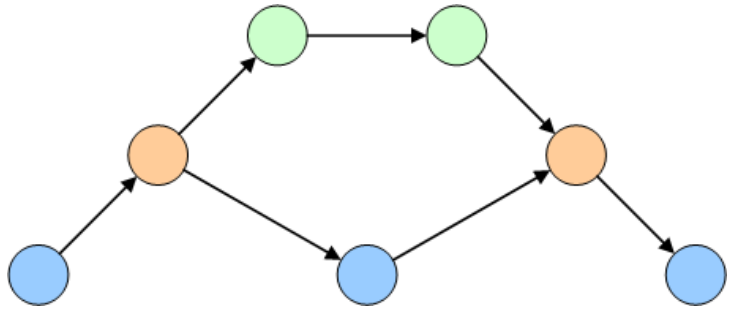
- organize graph into layers, and introduce shortcuts
- apply **Dijkstra**, never take edges to lower layers

pro

- fast preprocessing
- fast queries

contra

- bidirectional query mandatory
- graph needs inherent hierarchy



Highway / Contraction Hierarchies

hierarchy-aware approaches

basic idea

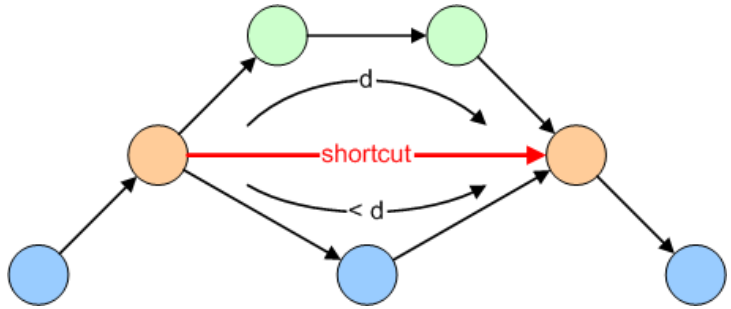
- organize graph into layers, and introduce shortcuts
- apply **Dijkstra**, never take edges to lower layers

pro

- fast preprocessing
- fast queries

contra

- bidirectional query mandatory
- graph needs inherent hierarchy



Highway / Contraction Hierarchies

hierarchy-aware approaches

basic idea

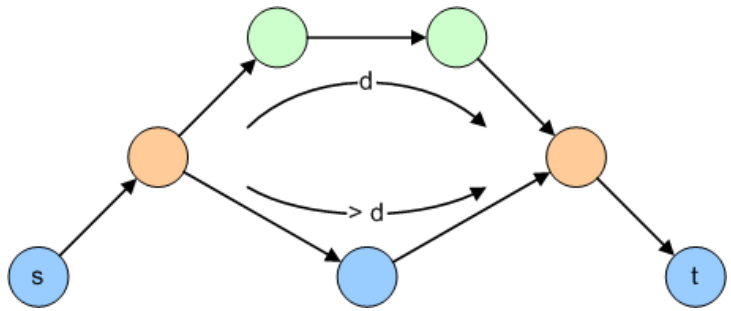
- organize graph into layers, and introduce shortcuts
- apply **Dijkstra**, never take edges to lower layers

pro

- fast preprocessing
- fast queries

contra

- bidirectional query mandatory
- graph needs inherent hierarchy



Highway / Contraction Hierarchies

hierarchy-aware approaches

basic idea

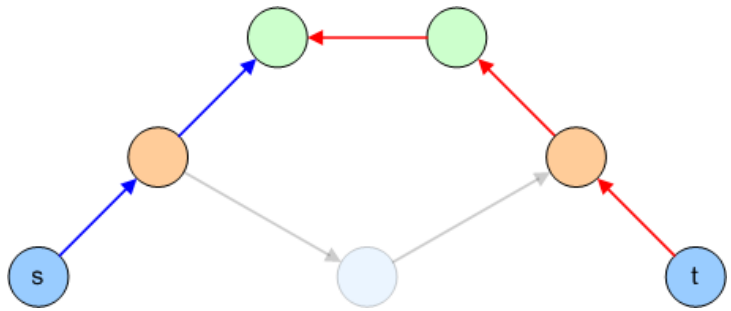
- organize graph into layers, and introduce shortcuts
- apply **Dijkstra**, never take edges to lower layers

pro

- fast preprocessing
- fast queries

contra

- bidirectional query mandatory
- graph needs inherent hierarchy



Highway / Contraction Hierarchies

hierarchy-aware approaches

basic idea

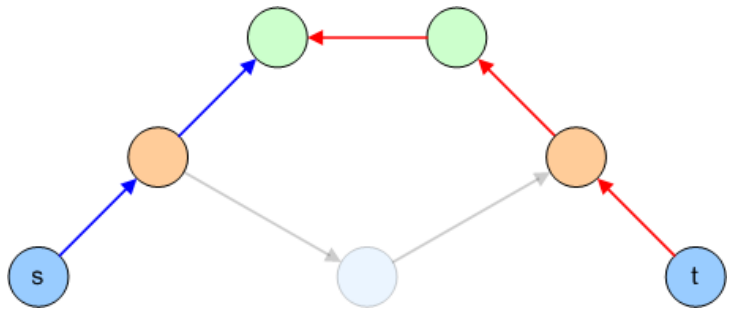
- organize graph into layers, and introduce shortcuts
- apply **Dijkstra**, never take edges to lower layers

pro

- fast preprocessing
- fast queries

contra

- bidirectional query mandatory
- graph needs inherent hierarchy



Highway / Contraction Hierarchies

hierarchy-aware approaches

basic idea

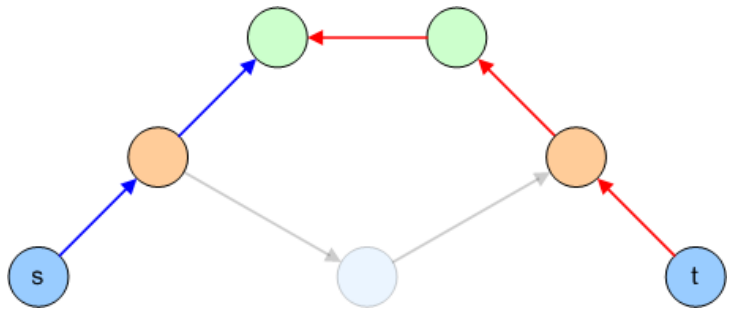
- organize graph into layers, and introduce shortcuts
- apply **Dijkstra**, never take edges to lower layers

pro

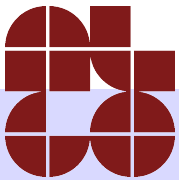
- fast preprocessing
- fast queries

contra

- bidirectional query mandatory
- graph needs inherent hierarchy



- II -
Combinations
of speed-up techniques



Combination of Speed-Up Techniques

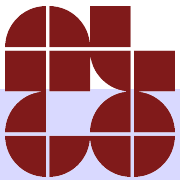
basics

Basic Goals

- higher speed-ups
- less overhead
  (preprocessing time, space consumption)

Basic Considerations

- different speed-up techniques exploit different properties of the graph
  ↪ combine them to profit from more features of the graph
- take advantage of additional synergy effects
  ↪ apply results of one algorithm to the other
  ↪ use less powerful variants of the individual techniques
- shortcuts are always a favourable additive



Combination of Speed-Up Techniques

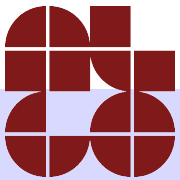
basics

Basic Goals

- higher speed-ups
- less overhead
  (preprocessing time, space consumption)

Basic Considerations

- different speed-up techniques exploit different properties of the graph
  ↪ combine them to profit from more features of the graph
- take advantage of additional synergy effects
  ↪ apply results of one algorithm to the other
  ↪ use less powerful variants of the individual techniques
- shortcuts are always a favourable additive



Combination of Speed-Up Techniques

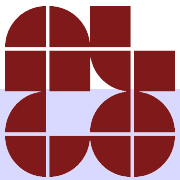
basics

Basic Goals

- higher speed-ups
- less overhead
  (preprocessing time, space consumption)

Basic Considerations

- different speed-up techniques exploit different properties of the graph
  ↪ combine them to profit from more features of the graph
- take advantage of additional synergy effects
  ↪ apply results of one algorithm to the other
  ↪ use less powerful variants of the individual techniques
- shortcuts are always a favourable additive



Combination of Speed-Up Techniques

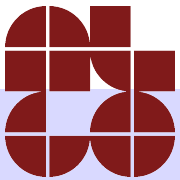
basics

Basic Goals

- higher speed-ups
- less overhead
  (preprocessing time, space consumption)

Basic Considerations

- different speed-up techniques exploit different properties of the graph
  ↪ combine them to profit from more features of the graph
- take advantage of additional synergy effects
  ↪ apply results of one algorithm to the other
  ↪ use less powerful variants of the individual techniques
- shortcuts are always a favourable additive



Bidirectional + X

basic combinations

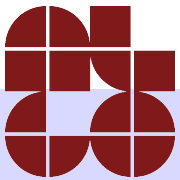
Goals

- decrease search space
- speed-up of queries

Basic Algorithm

- start a query from $s \rightarrow t$ and from $t \rightarrow s$
- advance both searches alternately
- stop search after both search spaces have met and no shorter path can be found

- Almost all speed-up techniques are coded bidirectionally
- Mandatory for most hierarchical techniques

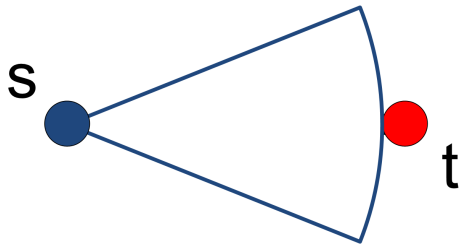


Bidirectional + X

basic combinations

Goals

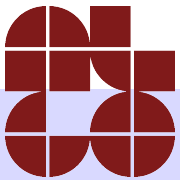
- decrease search space
- speed-up of queries



Basic Algorithm

- start a query from $s \rightarrow t$ and from $t \rightarrow s$
- advance both searches alternately
- stop search after both search spaces have met and no shorter path can be found

- Almost all speed-up techniques are coded bidirectionally
- Mandatory for most hierarchical techniques

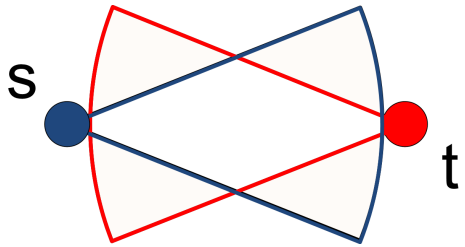


Bidirectional + X

basic combinations

Goals

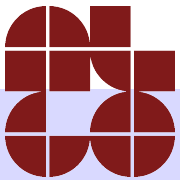
- decrease search space
- speed-up of queries



Basic Algorithm

- start a query from $s \rightarrow t$ and from $t \rightarrow s$
- advance both searches alternately
- stop search after both search spaces have met and no shorter path can be found

- Almost all speed-up techniques are coded bidirectionally
- Mandatory for most hierarchical techniques

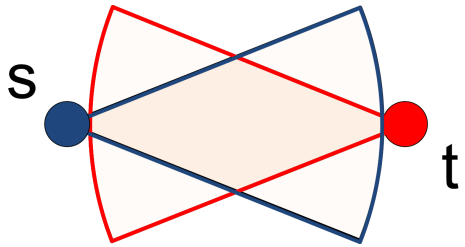


Bidirectional + X

basic combinations

Goals

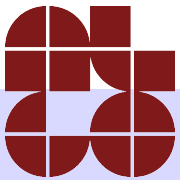
- decrease search space
- speed-up of queries



Basic Algorithm

- start a query from $s \rightarrow t$ and from $t \rightarrow s$
- advance both searches alternately
- stop search after both search spaces have met and no shorter path can be found

- Almost all speed-up techniques are coded bidirectionally
- Mandatory for most hierarchical techniques

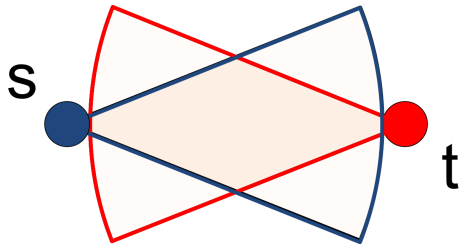


Bidirectional + X

basic combinations

Goals

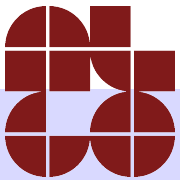
- decrease search space
- speed-up of queries



Basic Algorithm

- start a query from $s \longrightarrow t$ and from $t \longrightarrow s$
- advance both searches alternately
- stop search after both search spaces have met and no shorter path can be found

- Almost all speed-up techniques are coded bidirectionally
- Mandatory for most hierarchical techniques

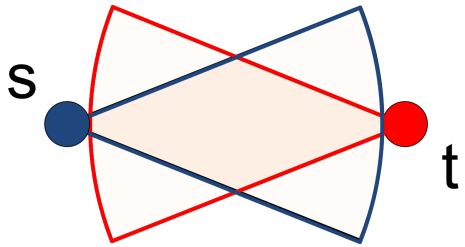


Bidirectional + X

basic combinations

Goals

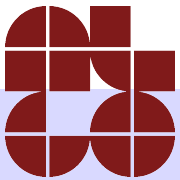
- decrease search space
- speed-up of queries



Basic Algorithm

- start a query from $s \longrightarrow t$ and from $t \longrightarrow s$
- advance both searches alternately
- stop search after both search spaces have met and no shorter path can be found

- Almost all speed-up techniques are coded bidirectionally
- Mandatory for most hierarchical techniques



ArcFlags + ALT (AALT)

basic combinations

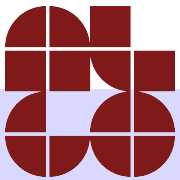
Goals

- speed-up of local queries within a region
- use less regions / landmarks
  ↪ faster preprocessing, smaller memory overhead

Basic Algorithm

- perform bidirectional ALT query
- prune edges according to ArcFlags

- time-expanded timetable queries [pajor08]
  - ArcFlags: geographical search
  - ALT: temporal search
- road maps [schiefer08]



ArcFlags + ALT (AALT)

basic combinations

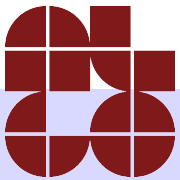
Goals

- speed-up of local queries within a region
- use less regions / landmarks
  ↪ faster preprocessing, smaller memory overhead

Basic Algorithm

- perform bidirectional ALT query
- prune edges according to ArcFlags

- time-expanded timetable queries [pajor08]
  - ArcFlags: geographical search
  - ALT: temporal search
- road maps [schiefer08]



ArcFlags + ALT (AALT)

basic combinations

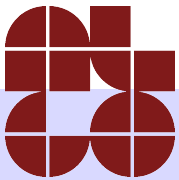
Goals

- speed-up of local queries within a region
- use less regions / landmarks
  ↪ faster preprocessing, smaller memory overhead

Basic Algorithm

- perform bidirectional ALT query
- prune edges according to ArcFlags

- time-expanded timetable queries [pajor08]
  - ArcFlags: geographical search
  - ALT: temporal search
- road maps [schiefer08]



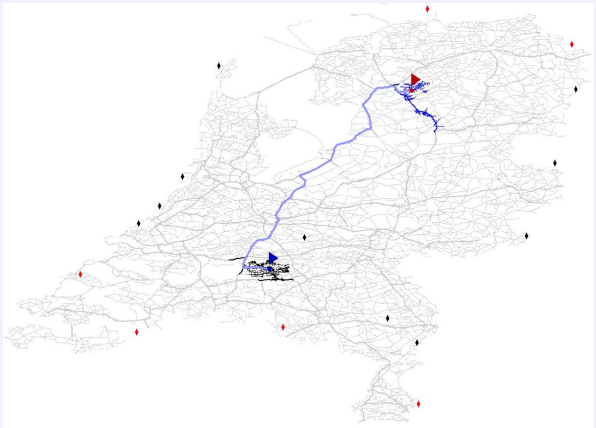
ArcFlags + ALT (AALT)

basic combinations

Observations for Road Networks

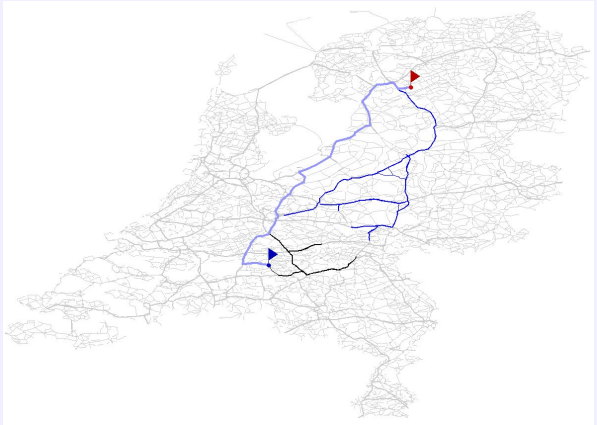
- no speed-up of local queries
- speed-up of longer queries
- higher speed-up for distance metrik than for travel-times

ALT



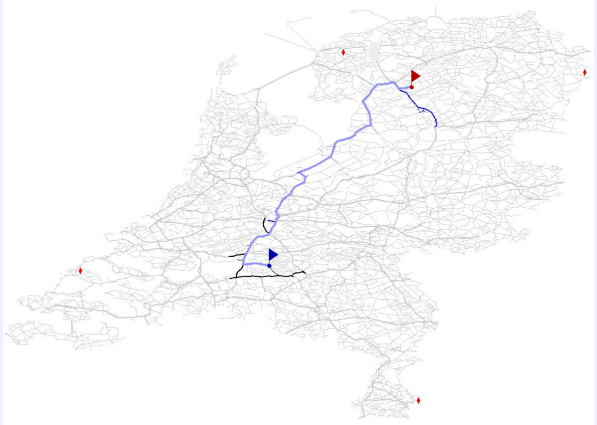
16 landmarks

ArcFlags



16 regions

ArcFlags + ALT



2 regions + 4 landmarks

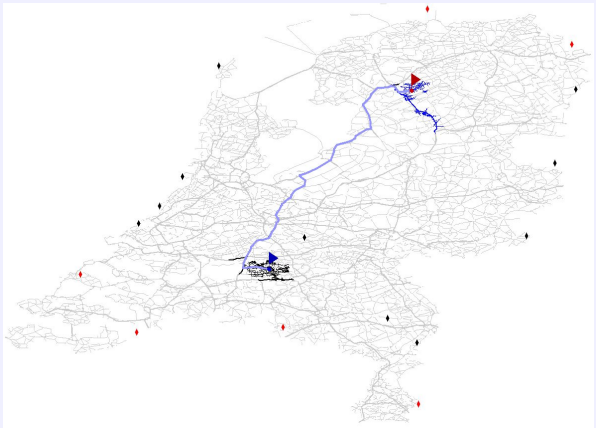
ArcFlags + ALT (AALT)

basic combinations

Observations for Road Networks

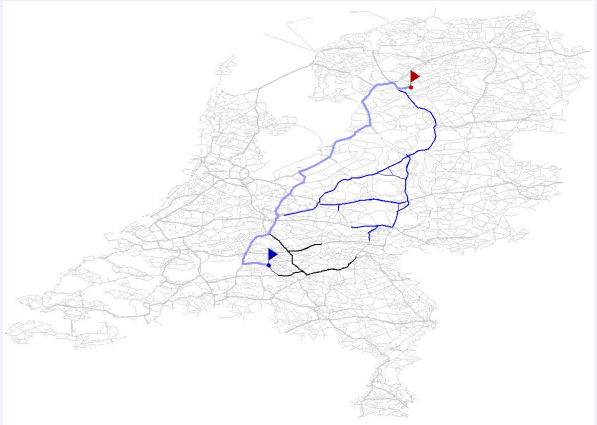
- no speed-up of local queries
- speed-up of longer queries
- higher speed-up for distance metrik than for travel-times

ALT



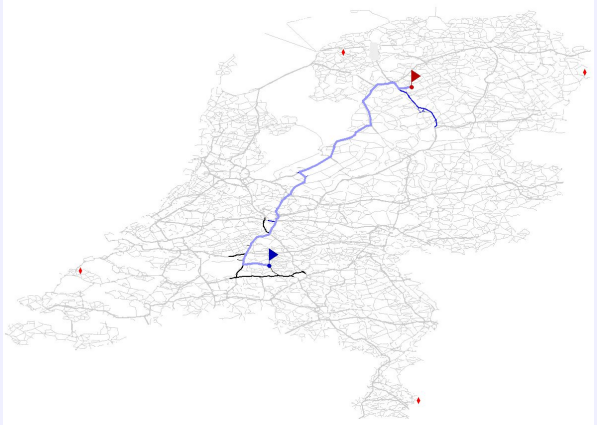
16 landmarks

ArcFlags



16 regions

ArcFlags + ALT



2 regions + 4 landmarks

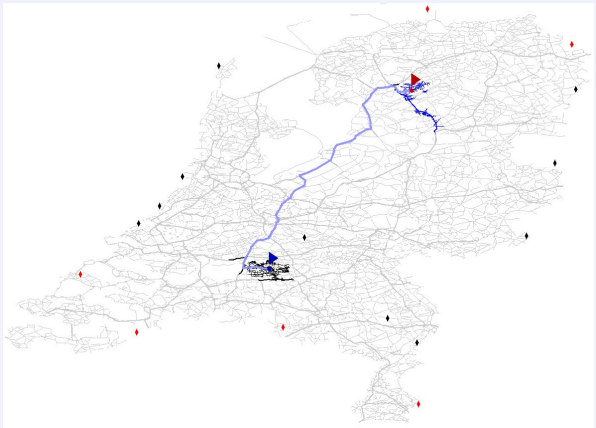
ArcFlags + ALT (AALT)

basic combinations

Observations for Road Networks

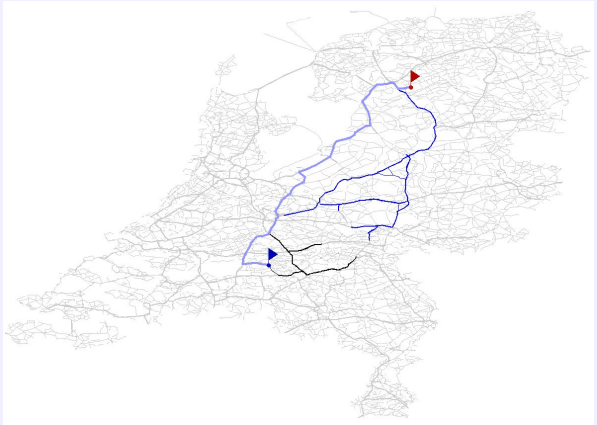
- no speed-up of local queries
- speed-up of longer queries
- higher speed-up for distance metrik than for travel-times

ALT



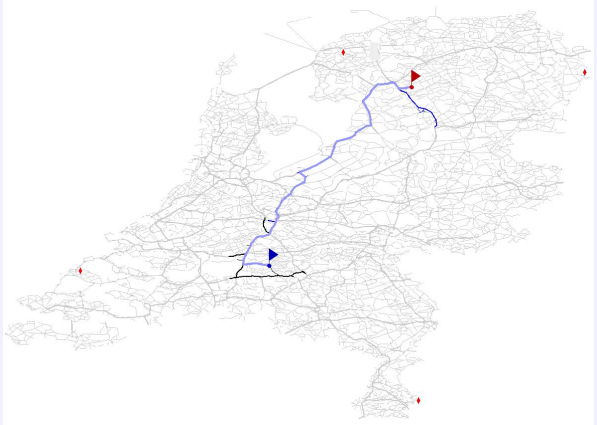
16 landmarks

ArcFlags



16 regions

ArcFlags + ALT



2 regions + 4 landmarks

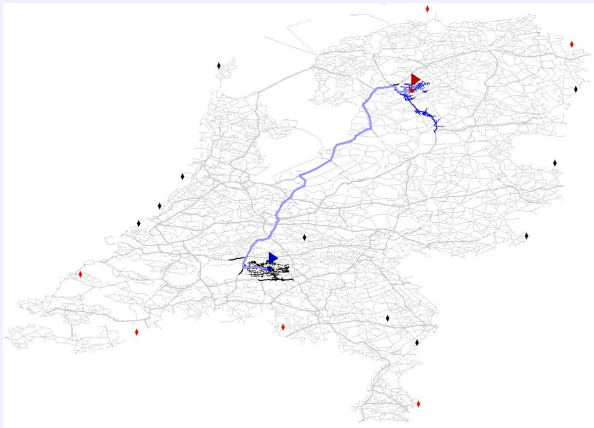
ArcFlags + ALT (AALT)

basic combinations

Observations for Road Networks

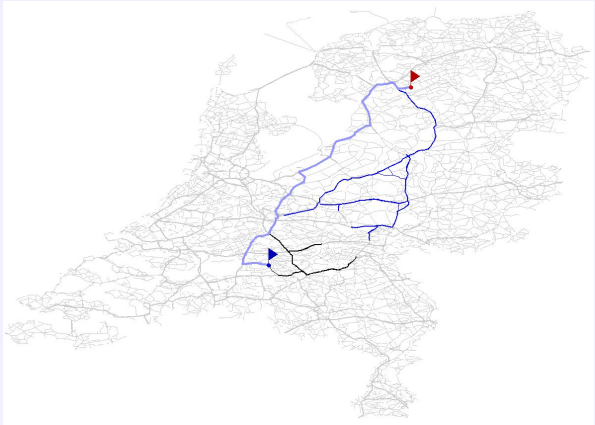
- no speed-up of local queries
- speed-up of longer queries
- higher speed-up for distance metrik than for travel-times

ALT



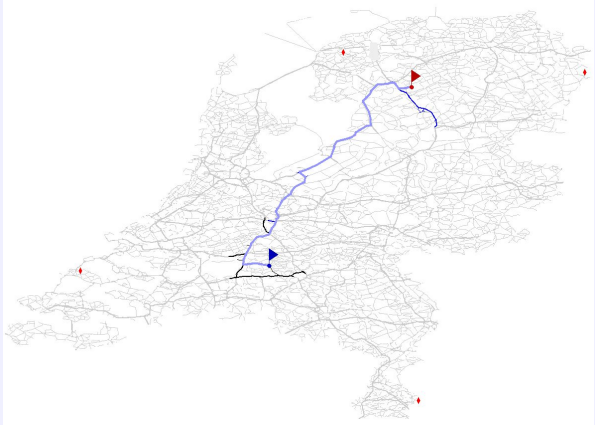
16 landmarks

ArcFlags



16 regions

ArcFlags + ALT



2 regions + 4 landmarks

Goal Direction + Hierarchies

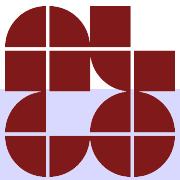
advanced combinations

Goals

- profit from two completely different properties of the graph
  - inherent structural hierarchies
  - (usually) geometric relations

General Ideas

- use hierarchical technique as basis
- two approaches for goal-direction
  - blindly add goal-direction on the whole graph (REAL, HH*)
  - use level information and add goal-direction only on higher levels: core-based routing (CALT, HiFlags)



Goal Direction + Hierarchies

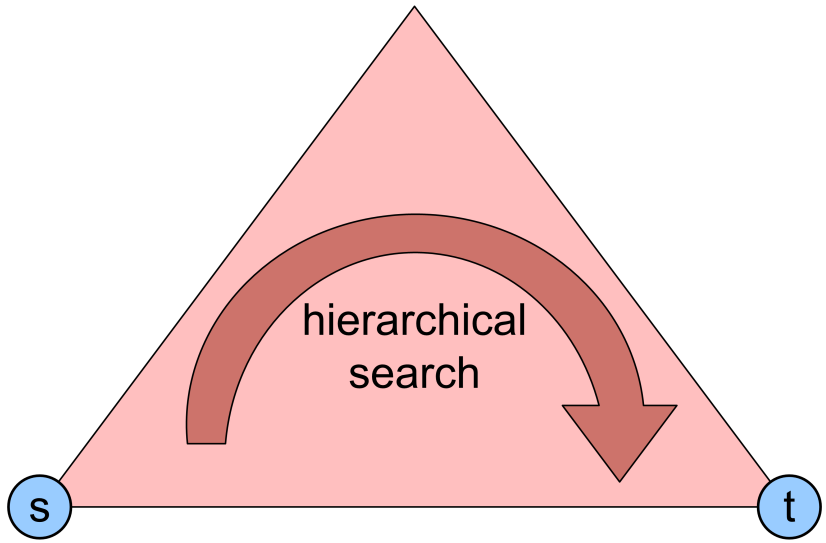
advanced combinations

Goals

- profit from two completely different properties of the graph
  - inherent structural hierarchies
  - (usually) geometric relations

General Ideas

- use hierarchical technique as basis
- two approaches for goal-direction
  - blindly add goal-direction on the whole graph (REAL, HH*)
  - use level information and add goal-direction only on higher levels: core-based routing (CALT, HiFlags)



Goal Direction + Hierarchies

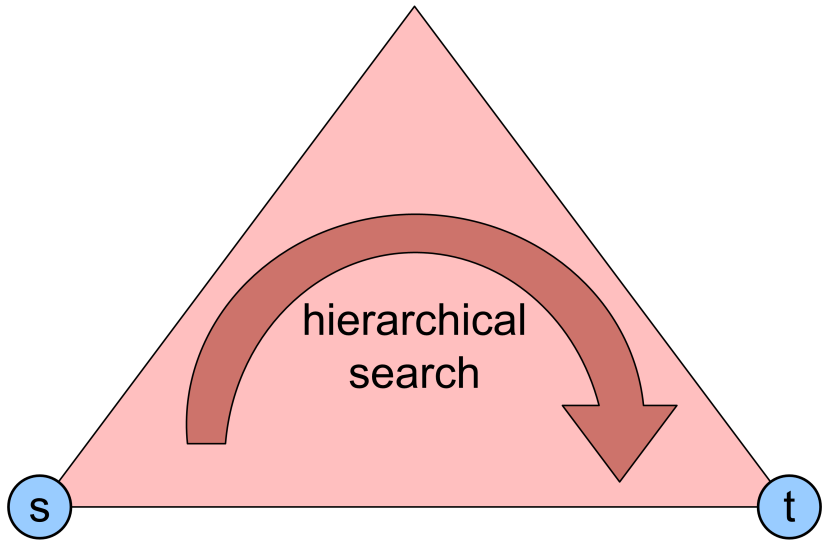
advanced combinations

Goals

- profit from two completely different properties of the graph
  - inherent structural hierarchies
  - (usually) geometric relations

General Ideas

- use hierarchical technique as basis
- two approaches for goal-direction
  - blindly add goal-direction on the whole graph (REAL, HH*)
  - use level information and add goal-direction only on higher levels: core-based routing (CALT, HiFlags)



Goal Direction + Hierarchies

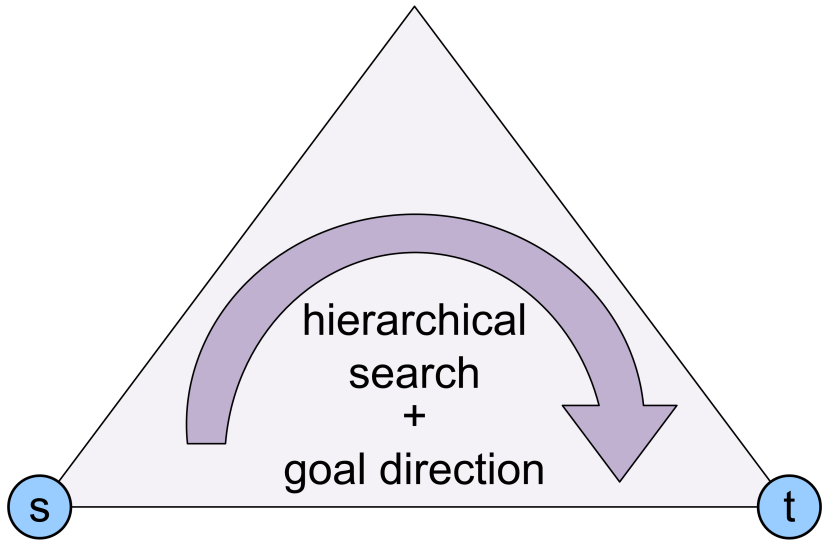
advanced combinations

Goals

- profit from two completely different properties of the graph
  - inherent structural hierarchies
  - (usually) geometric relations

General Ideas

- use hierarchical technique as basis
- two approaches for goal-direction
  - blindly add goal-direction on the whole graph (REAL, HH*)
  - use level information and add goal-direction only on higher levels: core-based routing (CALT, HiFlags)



Goal Direction + Hierarchies

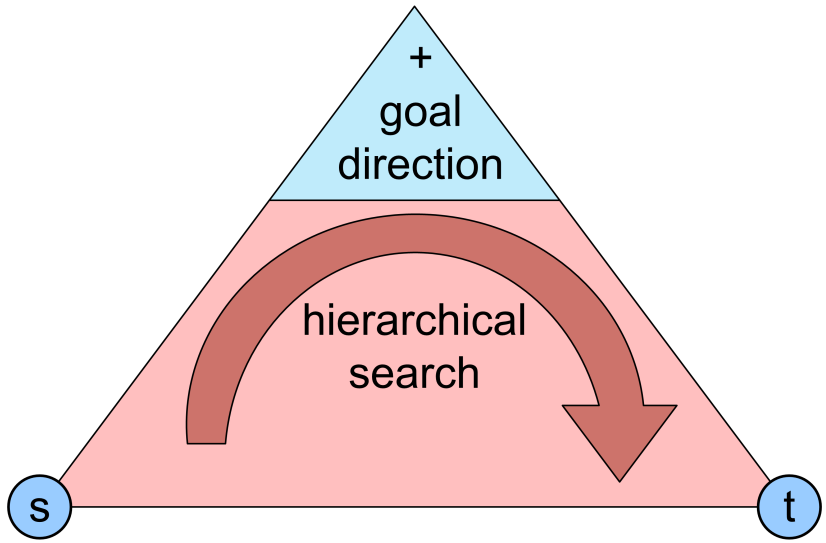
advanced combinations

Goals

- profit from two completely different properties of the graph
  - inherent structural hierarchies
  - (usually) geometric relations

General Ideas

- use hierarchical technique as basis
- two approaches for goal-direction
  - blindly add goal-direction on the whole graph (REAL, HH*)
  - use level information and add goal-direction only on higher levels: **core-based routing** (CALT, HiFlags)

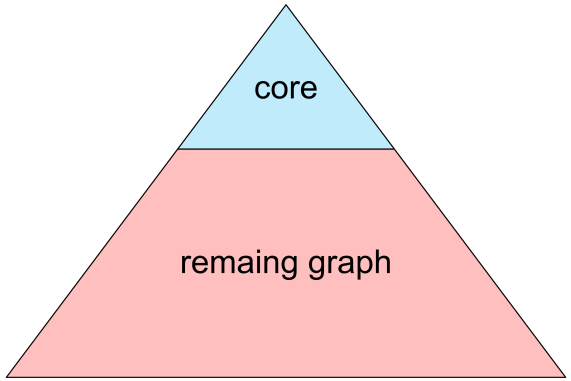


Goal Direction + Hierarchies

advanced combinations

Definiton: k%-Core of a Graph

- subgraph of the original graph
- induced by the k% of all nodes with the highest hierarchical levels
- one connected component by construction



Pitfalls when using Core-based Routing

- auxiliary data for goal-direction only available for core nodes / edges (landmark distances, regions, ...)
- source s and target t are usually not core nodes

Further Use of Hierarchies

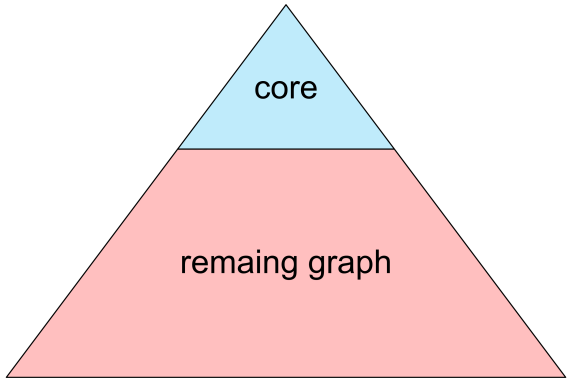
- Having to select "important" parts of a graph, can be reduced to looking inside the core (i.e. landmark selection)

Goal Direction + Hierarchies

advanced combinations

Definiton: k%-Core of a Graph

- subgraph of the original graph
- induced by the k% of all nodes with the highest hierarchical levels
- one connected component by construction



Pitfalls when using Core-based Routing

- auxiliary data for goal-direction only available for core nodes / edges (landmark distances, regions, ...)
- source s and target t are usually not core nodes

Further Use of Hierarchies

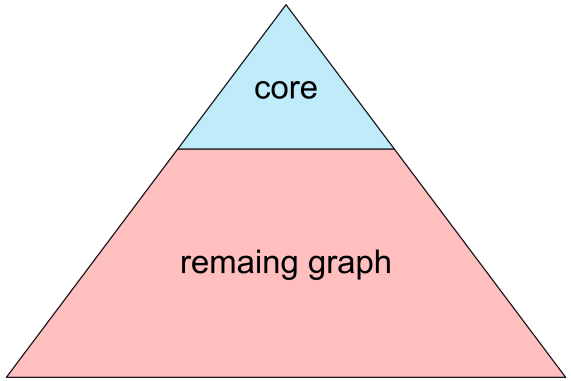
- Having to select "important" parts of a graph, can be reduced to looking inside the core (i.e. landmark selection)

Goal Direction + Hierarchies

advanced combinations

Definiton: k%-Core of a Graph

- subgraph of the original graph
- induced by the k% of all nodes with the highest hierarchical levels
- one connected component by construction



Pitfalls when using Core-based Routing

- auxiliary data for goal-direction only available for core nodes / edges (landmark distances, regions, ...)
- source s and target t are usually not core nodes

Further Use of Hierarchies

- Having to select "important" parts of a graph, can be reduced to looking inside the core (i.e. landmark selection)

HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

General Information – Preprocessing

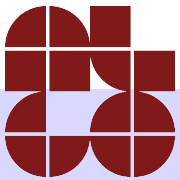
- consists of two separate preprocessing routines
  - preprocessing for Contraction Hierarchies
  - preprocessing for ArcFlags
    (depends on results of CH preprocessing)

Contraction Hierarchies

- unmodified preprocessing, yields
  - level information for all nodes
  - optimized search graph
    (original graph with shortcuts
    and w/o descending edges)

ArcFlags

- uses modified scope for preprocessing
  - identify a k%-core of the grpah
  - partition the core
- compute ArcFlags only for the core



HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

General Information – Preprocessing

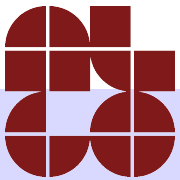
- consists of two separate preprocessing routines
  - preprocessing for Contraction Hierarchies
  - preprocessing for ArcFlags
    (depends on results of CH preprocessing)

Contraction Hierarchies

- unmodified preprocessing, yields
  - level information for all nodes
  - optimized search graph
    (original graph with shortcuts
    and w/o descending edges)

ArcFlags

- uses modified scope for preprocessing
  - identify a $k\%$-core of the grpah
  - partition the core
- compute ArcFlags only for the core



HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

General Information – Preprocessing

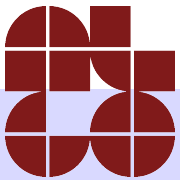
- consists of two separate preprocessing routines
  - preprocessing for Contraction Hierarchies
  - preprocessing for ArcFlags
    (depends on results of CH preprocessing)

Contraction Hierarchies

- unmodified preprocessing, yields
  - level information for all nodes
  - optimized search graph
    (original graph with shortcuts
    and w/o descending edges)

ArcFlags

- uses modified scope for preprocessing
  - identify a k%-core of the grpah
  - partition the core
- compute ArcFlags only for the core

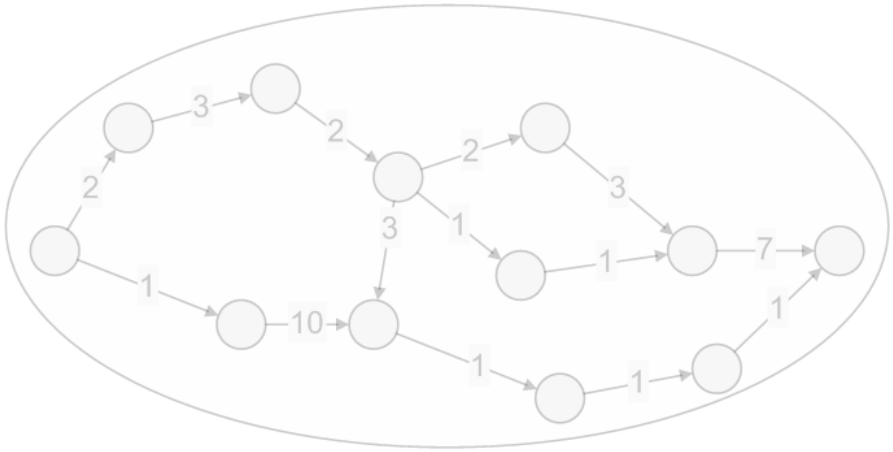


HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

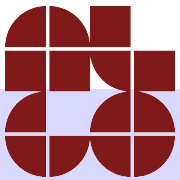
General Information – Query

- bidirectional query
- two phases



Phase 1

- perform modified **Contraction Hierarchies** query
  ↪ prune search at core-nodes (entry points into the core)
- continue search until
  ↪ remaining nodes do not contribute to a shortest path, or
  ↪ priority queues are empty
- proceed with phase 2 if no shortest path was found

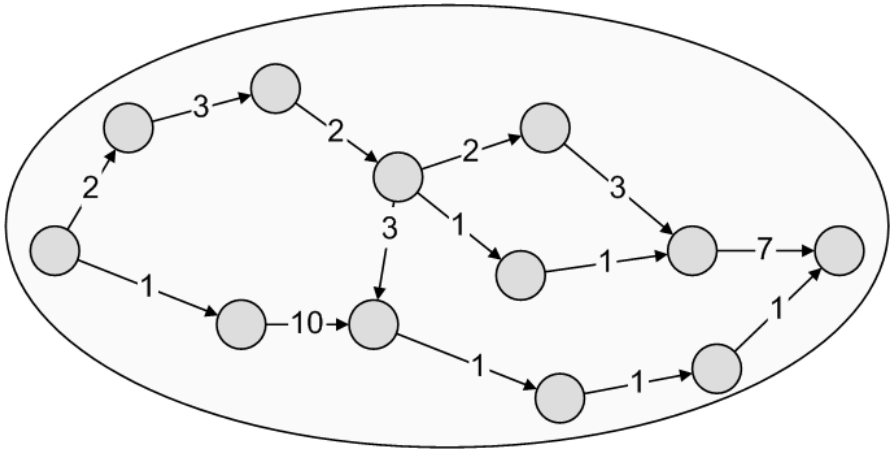


HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

General Information – Query

- bidirectional query
- two phases



Phase 1

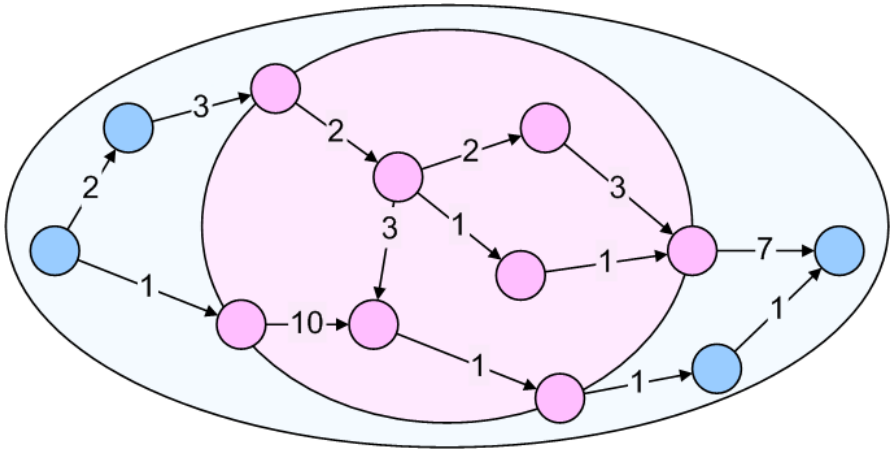
- perform modified **Contraction Hierarchies** query
  ↪ prune search at core-nodes (entry points into the core)
- continue search until
  ↪ remaining nodes do not contribute to a shortest path, or
  ↪ priority queues are empty
- proceed with phase 2 if no shortest path was found

HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

General Information – Query

- bidirectional query
- two phases



Phase 1

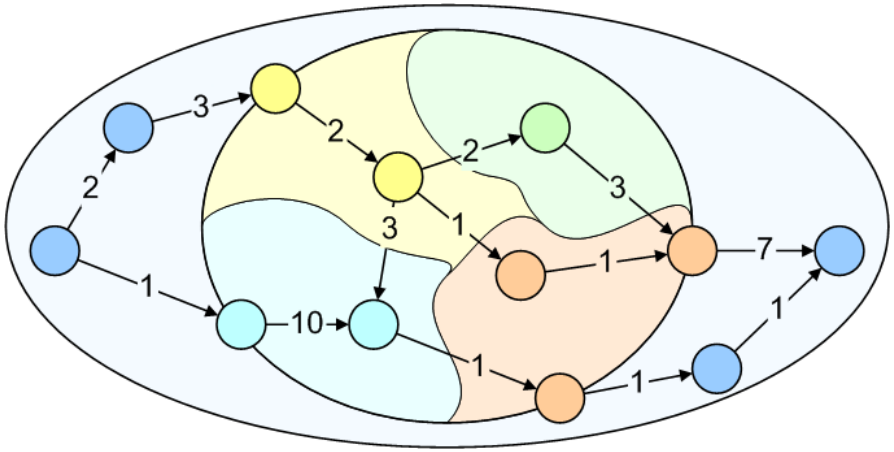
- perform modified **Contraction Hierarchies** query
  ↪ prune search at core-nodes (entry points into the core)
- continue search until
  ↪ remaining nodes do not contribute to a shortest path, or
  ↪ priority queues are empty
- proceed with phase 2 if no shortest path was found

HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

General Information – Query

- bidirectional query
- two phases



Phase 1

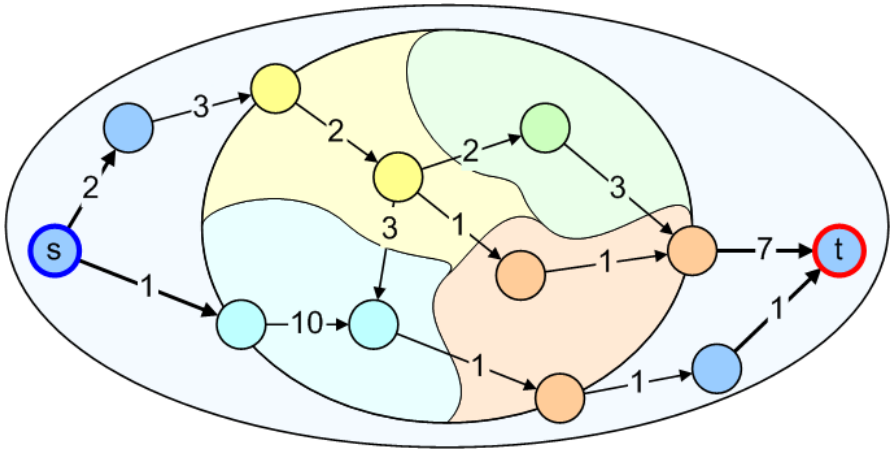
- perform modified **Contraction Hierarchies** query
  ↪ prune search at core-nodes (entry points into the core)
- continue search until
  ↪ remaining nodes do not contribute to a shortest path, or
  ↪ priority queues are empty
- proceed with phase 2 if no shortest path was found

HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

General Information – Query

- bidirectional query
- two phases



Phase 1

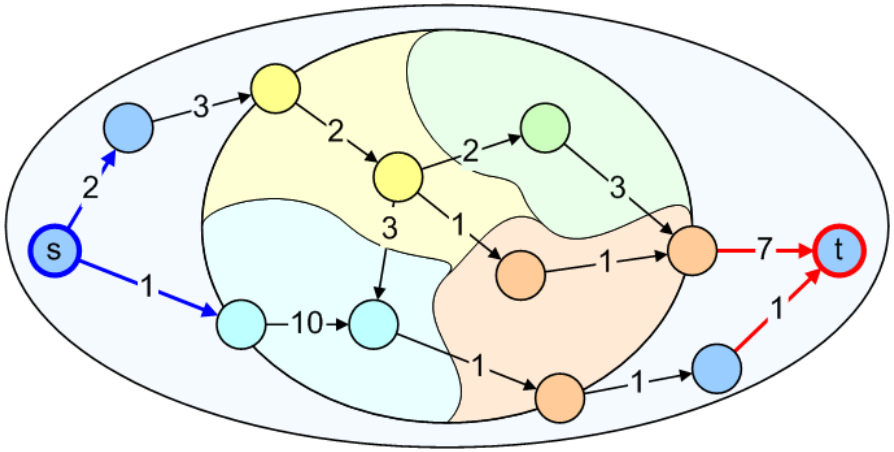
- perform modified **Contraction Hierarchies** query
  ↪ prune search at core-nodes (entry points into the core)
- continue search until
  ↪ remaining nodes do not contribute to a shortest path, or
  ↪ priority queues are empty
- proceed with phase 2 if no shortest path was found

HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

General Information – Query

- bidirectional query
- two phases



Phase 1

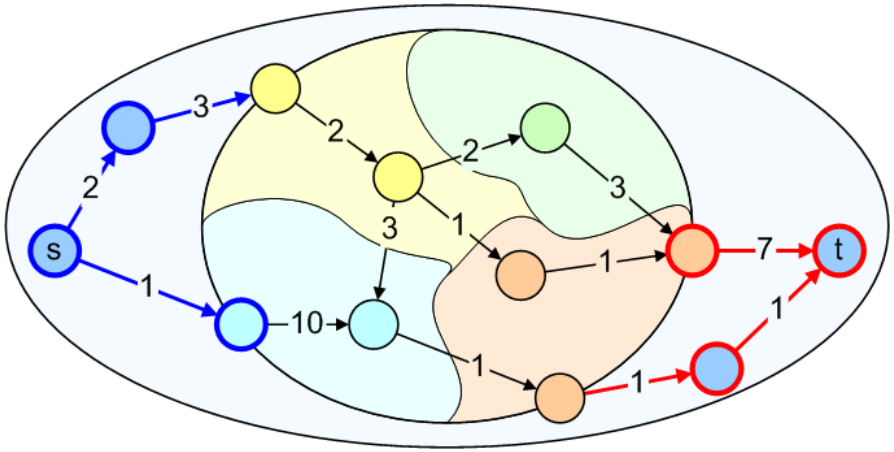
- perform modified **Contraction Hierarchies** query
  ↪ prune search at core-nodes (entry points into the core)
- continue search until
  ↪ remaining nodes do not contribute to a shortest path, or
  ↪ priority queues are empty
- proceed with phase 2 if no shortest path was found

HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

General Information – Query

- bidirectional query
- two phases



Phase 1

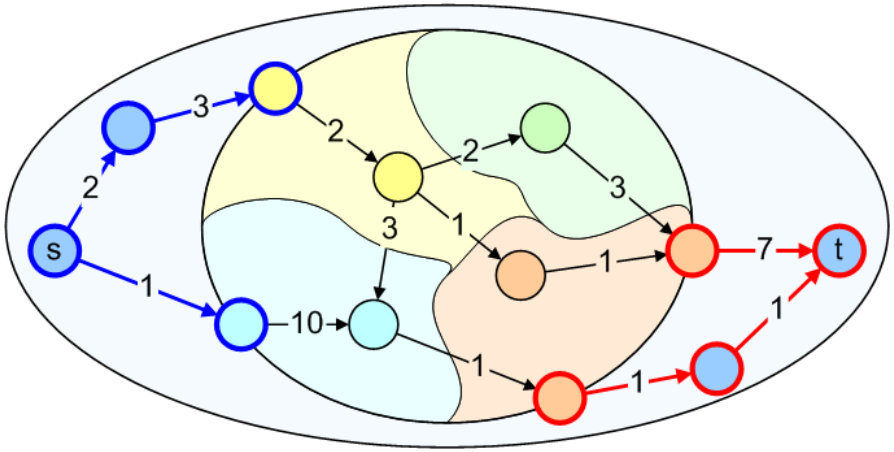
- perform modified **Contraction Hierarchies** query
  ↪ prune search at core-nodes (entry points into the core)
- continue search until
  ↪ remaining nodes do not contribute to a shortest path, or
  ↪ priority queues are empty
- proceed with phase 2 if no shortest path was found

HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

General Information – Query

- bidirectional query
- two phases



Phase 1

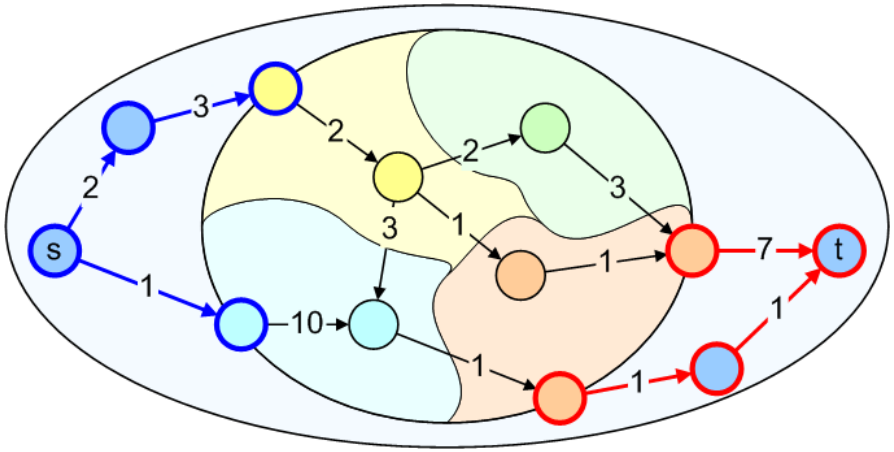
- perform modified **Contraction Hierarchies** query
  ↪ prune search at core-nodes (entry points into the core)
- continue search until
  ↪ remaining nodes do not contribute to a shortest path, or
  ↪ priority queues are empty
- proceed with phase 2 if no shortest path was found

HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

General Information – Query

- bidirectional query
- two phases



Phase 1

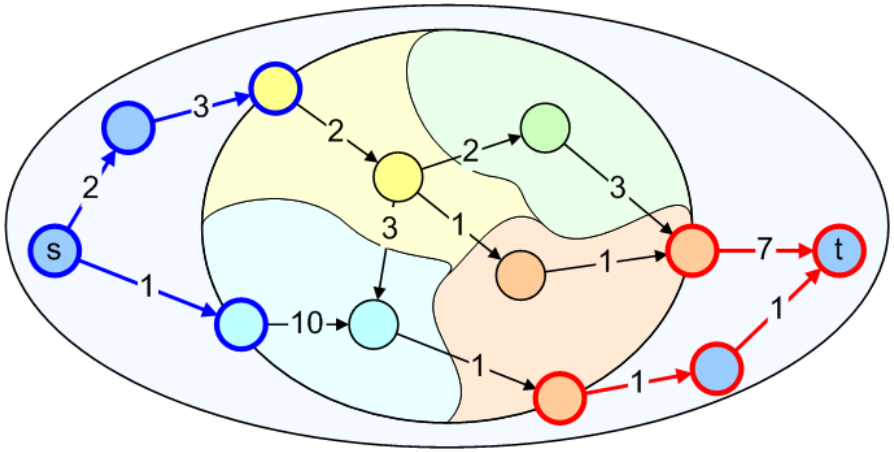
- perform modified **Contraction Hierarchies** query
  ↪ prune search at core-nodes (entry points into the core)
- continue search until
  ↪ remaining nodes do not contribute to a shortest path, or
  ↪ priority queues are empty
- proceed with phase 2 if no shortest path was found

HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

General Information – Query

- bidirectional query
- two phases



Phase 2

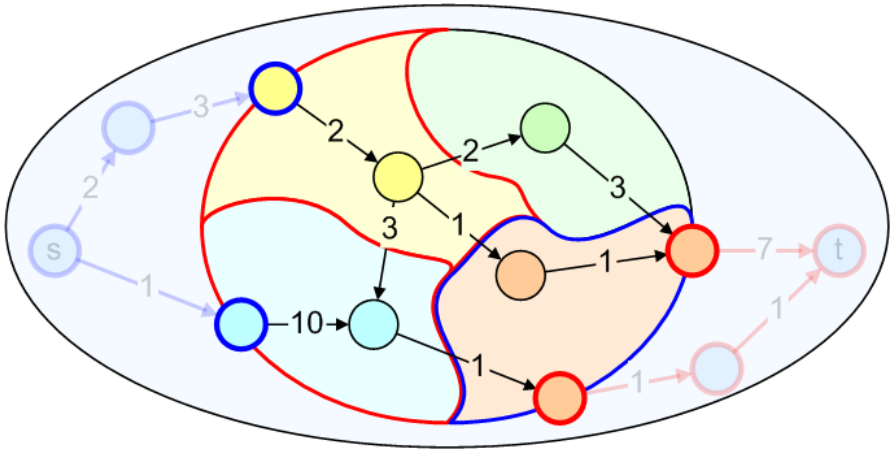
- initialization
  - ↪ flush priority queues
  - ↪ re-add entry points to the queues
  - ↪ remember all of their regions for ArcFlags pruning
- perform modified **Contraction Hierarchies** query (in the core)
  - ↪ prune search according to ArcFlags

HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

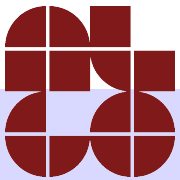
General Information – Query

- bidirectional query
- two phases



Phase 2

- initialization
  - ↪ flush priority queues
  - ↪ re-add entry points to the queues
  - ↪ remember all of their regions for ArcFlags pruning
- perform modified **Contraction Hierarchies** query (in the core)
  - ↪ prune search according to ArcFlags

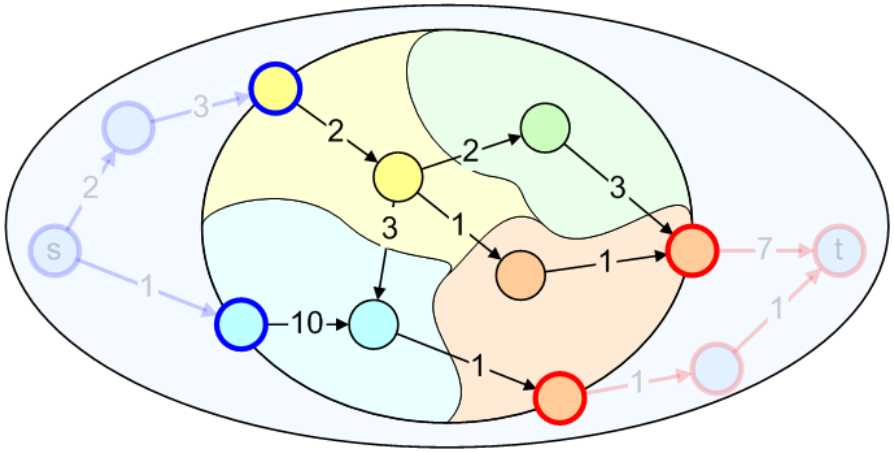


HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

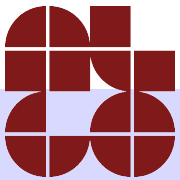
General Information – Query

- bidirectional query
- two phases



Phase 2

- initialization
  - ↪ flush priority queues
  - ↪ re-add entry points to the queues
  - ↪ remember all of their regions for ArcFlags pruning
- perform modified **Contraction Hierarchies** query (in the core)
  - ↪ prune search according to ArcFlags

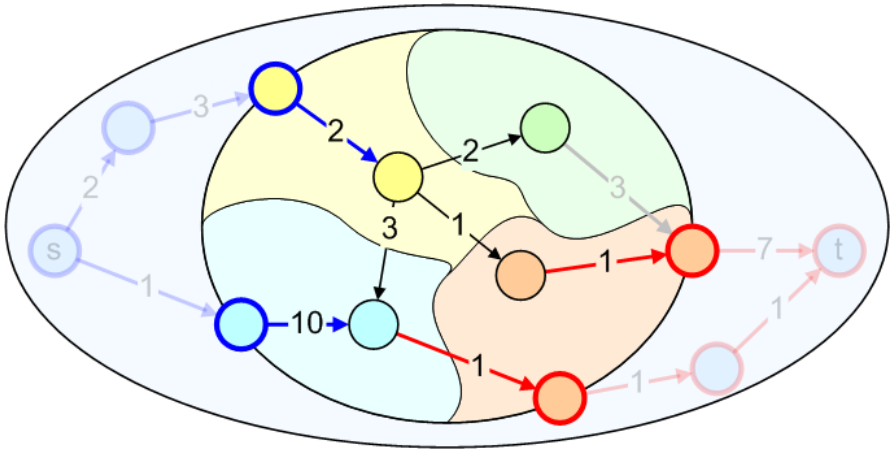


HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

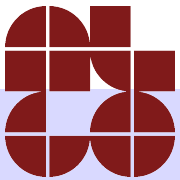
General Information – Query

- bidirectional query
- two phases



Phase 2

- initialization
  - ↪ flush priority queues
  - ↪ re-add entry points to the queues
  - ↪ remember all of their regions for ArcFlags pruning
- perform modified **Contraction Hierarchies** query (in the core)
  - ↪ prune search according to ArcFlags

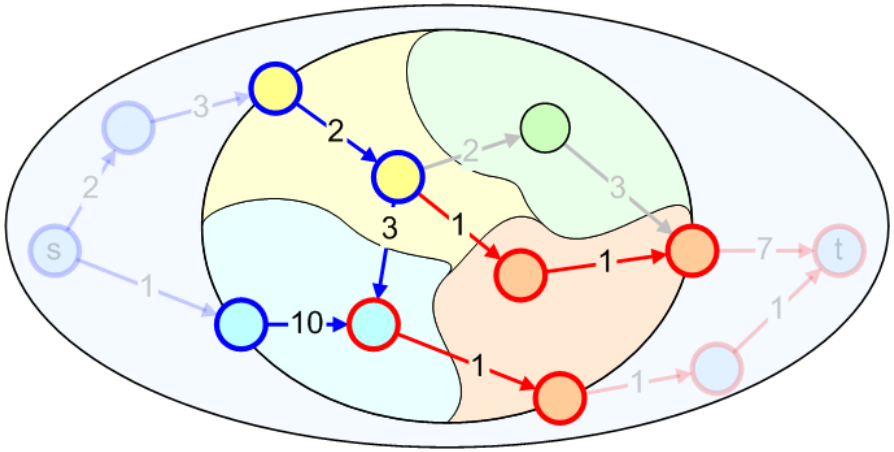


HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

General Information – Query

- bidirectional query
- two phases



Phase 2

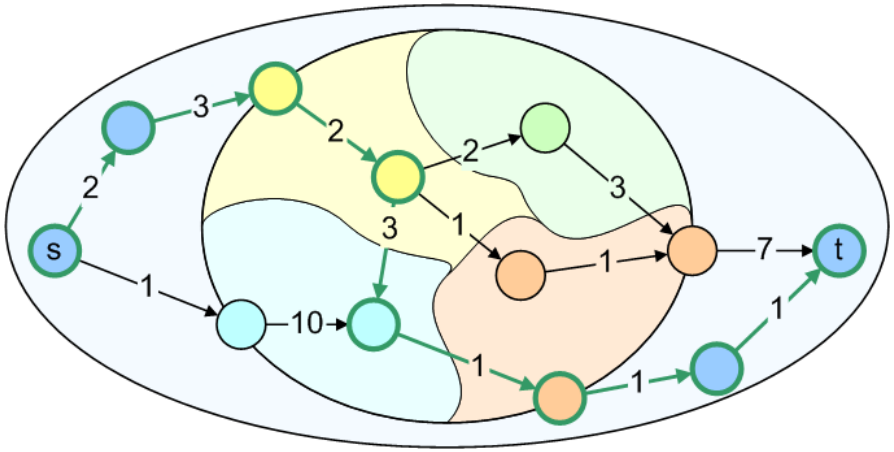
- initialization
  - ↪ flush priority queues
  - ↪ re-add entry points to the queues
  - ↪ remember all of their regions for ArcFlags pruning
- perform modified **Contraction Hierarchies** query (in the core)
  - ↪ prune search according to ArcFlags

HiFlags (Contraction Hierarchies & ArcFlags)

advanced combinations

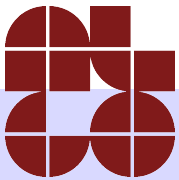
General Information – Query

- bidirectional query
- two phases



Phase 2

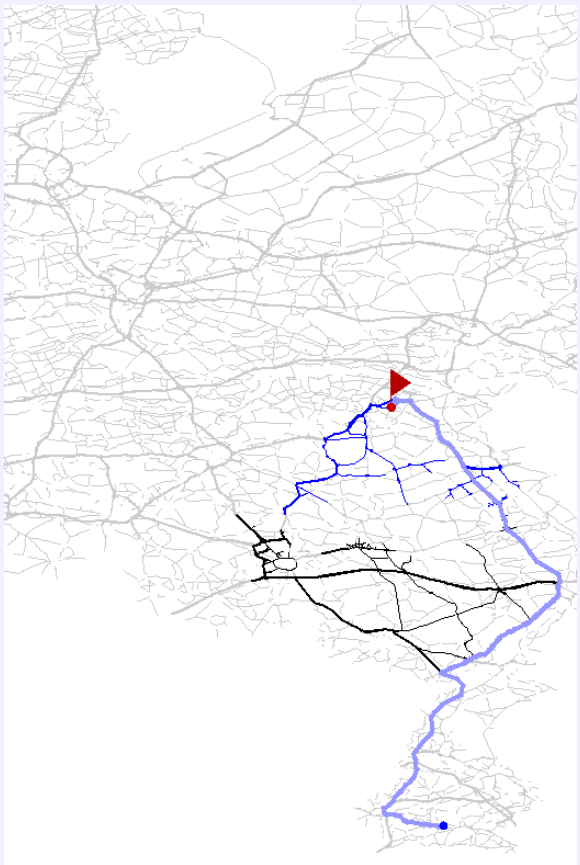
- initialization
  - ↪ flush priority queues
  - ↪ re-add entry points to the queues
  - ↪ remember all of their regions for ArcFlags pruning
- perform modified **Contraction Hierarchies** query (in the core)
  - ↪ prune search according to ArcFlags



HiFlags (Contraction Hierarchies & ArcFlags)

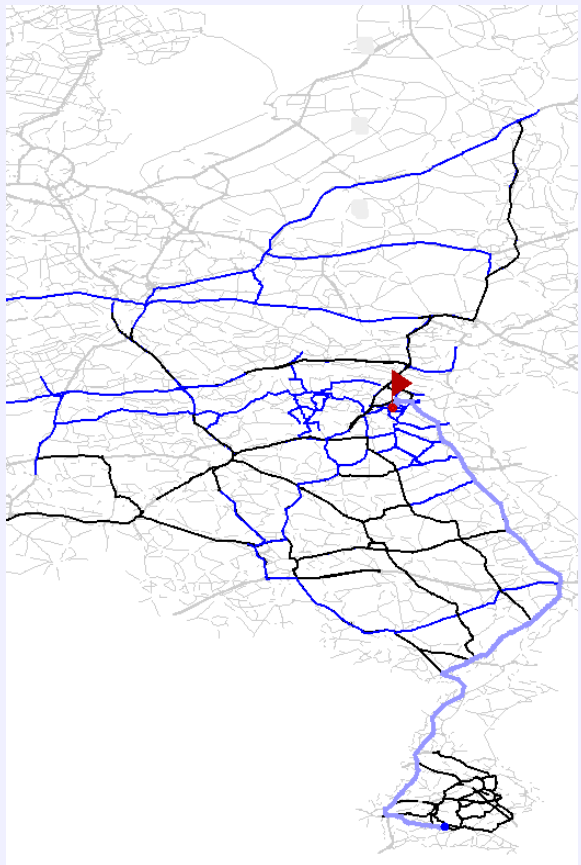
comparing search spaces

ArcFlags (16 regions)



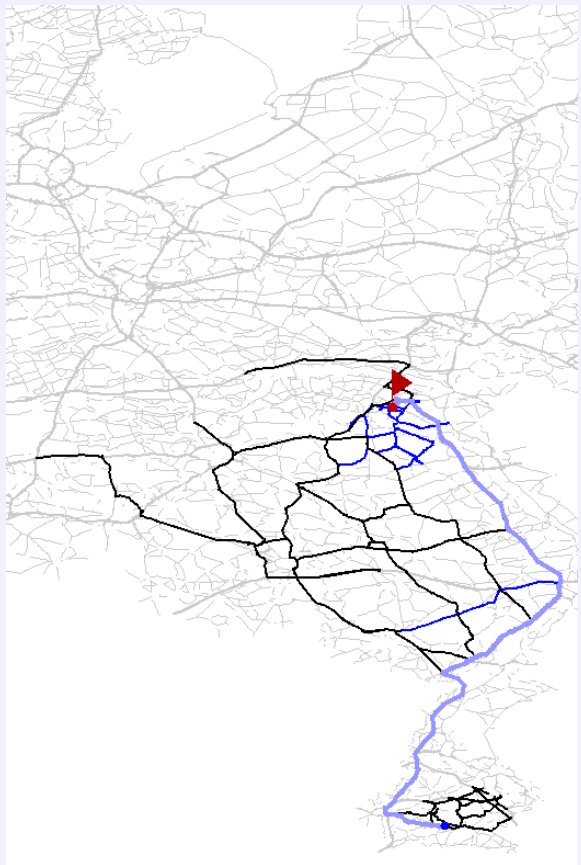
2 384 settled nodes

Contraction Hierarchies



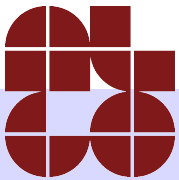
238 settled nodes

HiFlags (0.5% core)



144 settled nodes

- III -
Conclusion
Summary



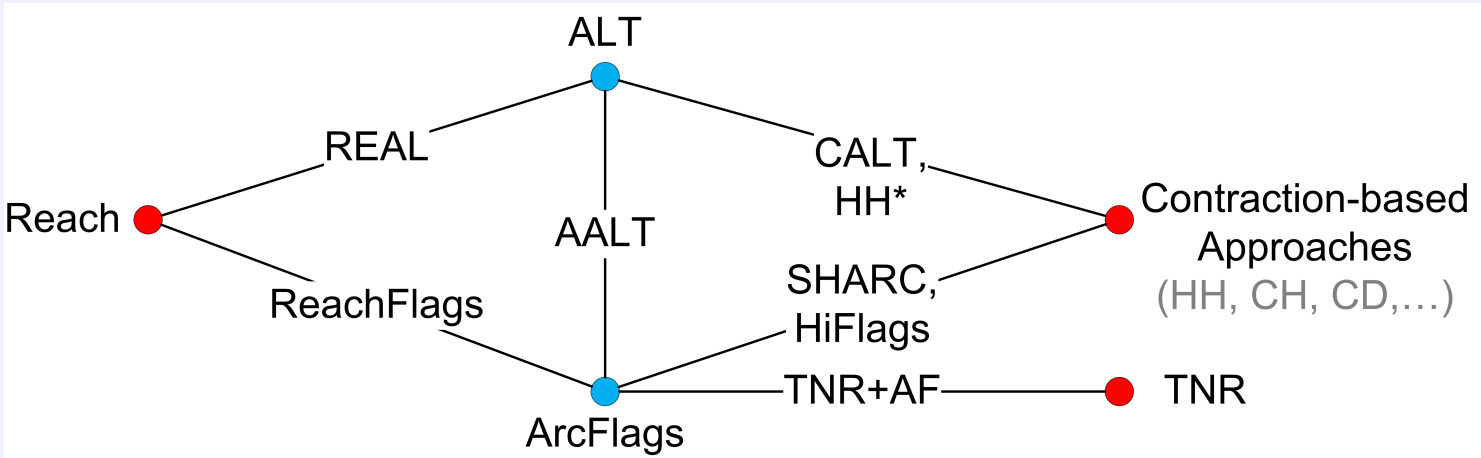
Summary

combination of speed-up techniques

To remember...

- almost all current techniques are simple combinations (bidirectional)
- combine techniques to exploit different properties of the graph
- combining goal-directed and hierarchical techniques is most promising
- applying goal-direction only to higher levels is sufficient

Several (prominent) combinations



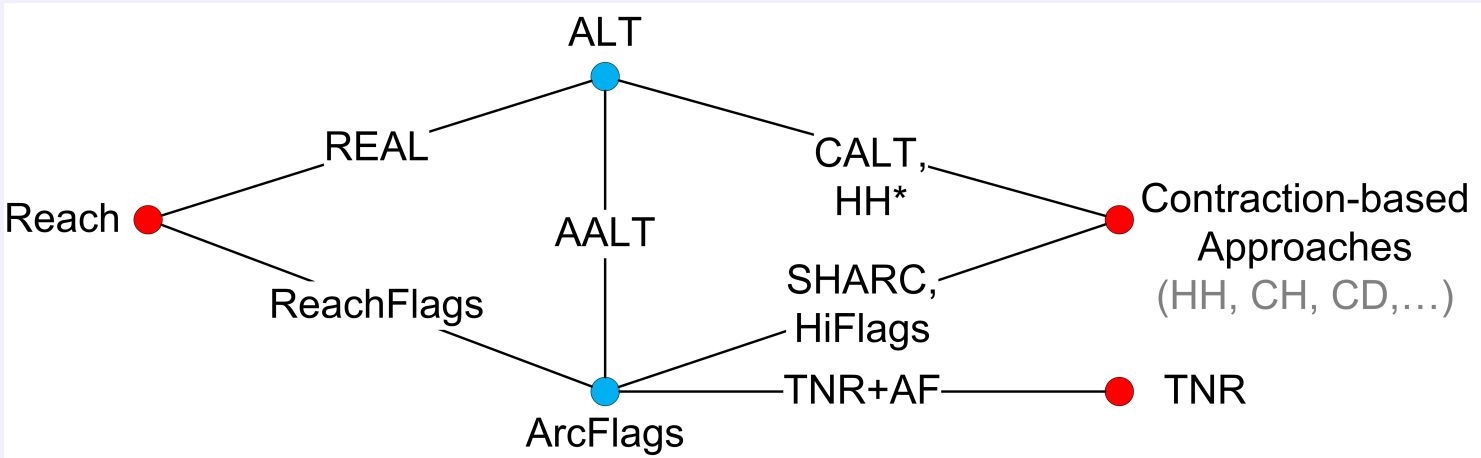
Summary

combination of speed-up techniques

To remember...

- almost all current techniques are simple combinations (bidirectional)
- combine techniques to exploit different properties of the graph
- combining goal-directed and hierarchical techniques is most promising
- applying goal-direction only to higher levels is sufficient

Several (prominent) combinations



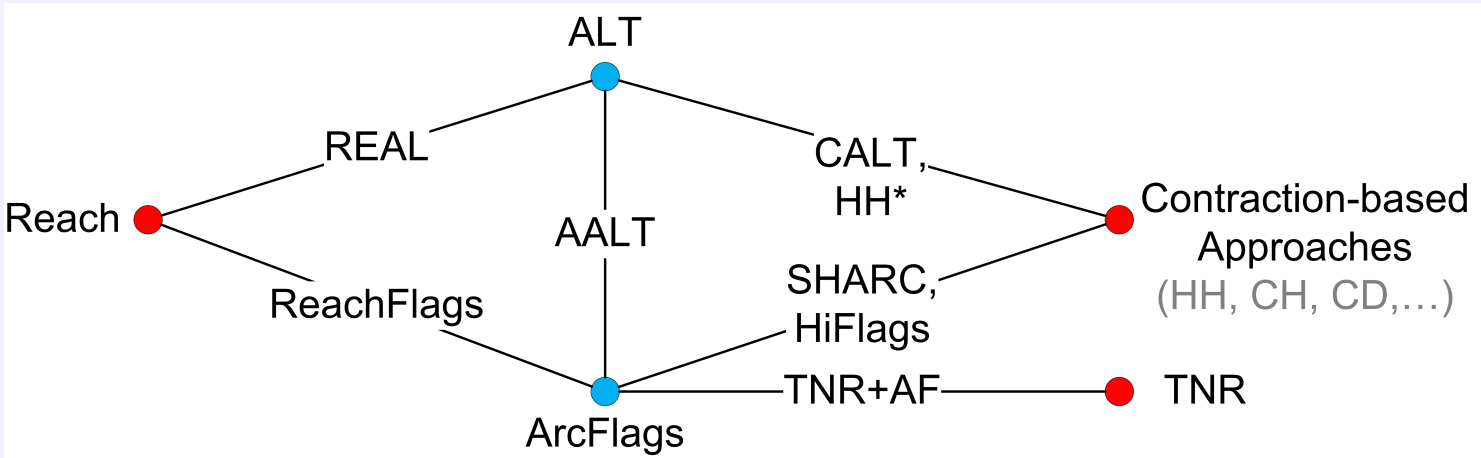
Summary

combination of speed-up techniques

To remember...

- almost all current techniques are simple combinations (bidirectional)
- combine techniques to exploit different properties of the graph
- combining goal-directed and hierarchical techniques is most promising
- applying goal-direction only to higher levels is sufficient

Several (prominent) combinations



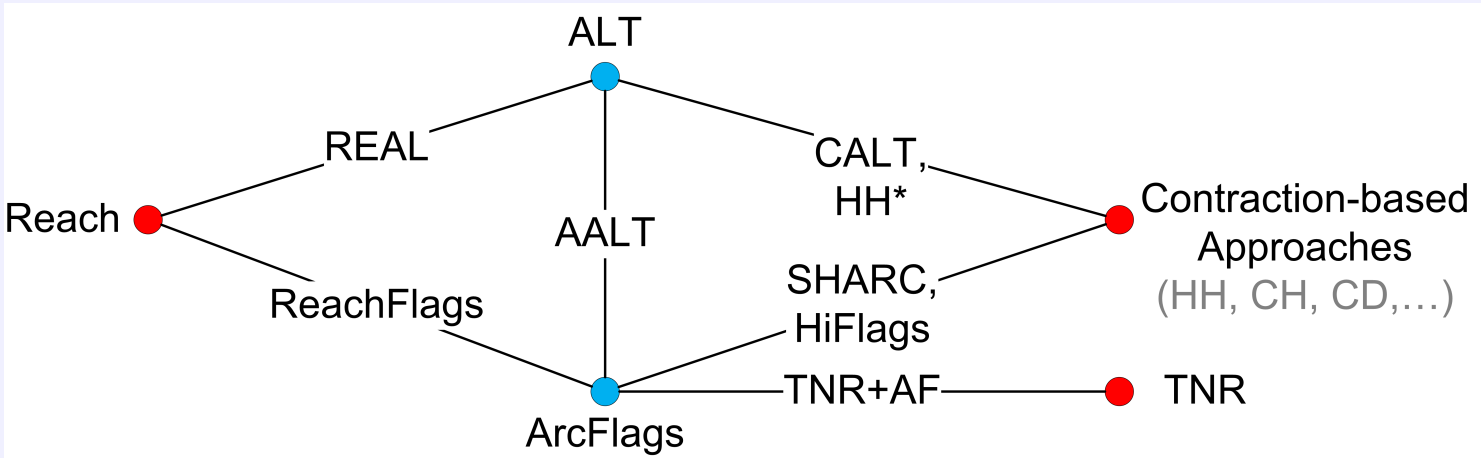
Summary

combination of speed-up techniques

To remember...

- almost all current techniques are simple combinations (bidirectional)
- combine techniques to exploit different properties of the graph
- combining goal-directed and hierarchical techniques is most promising
- applying goal-direction only to higher levels is sufficient

Several (prominent) combinations



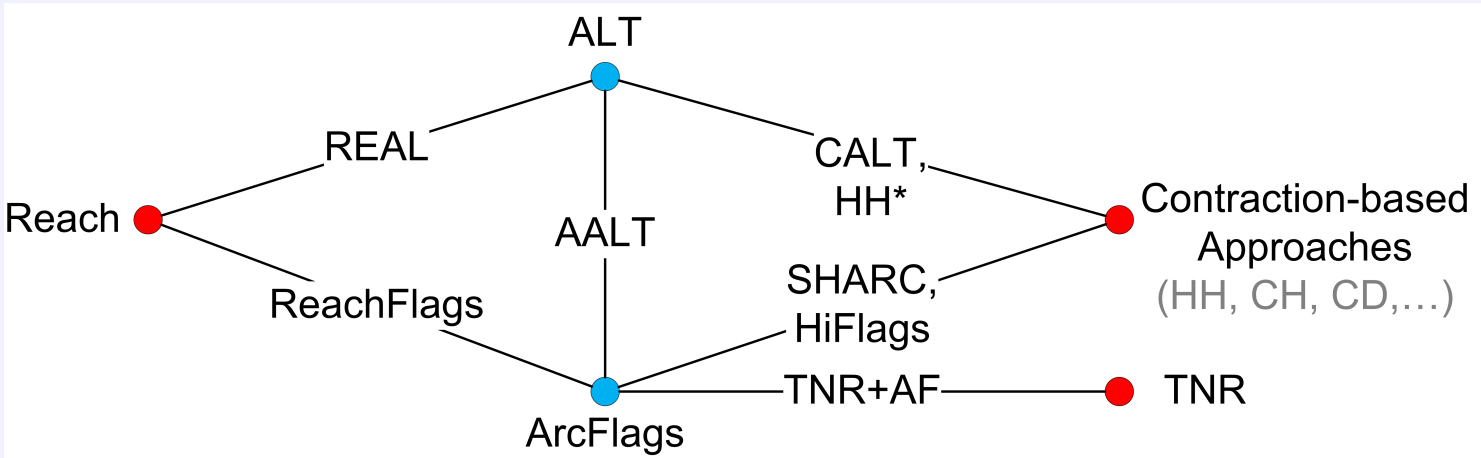
Summary

combination of speed-up techniques

To remember...

- almost all current techniques are simple combinations (bidirectional)
- combine techniques to exploit different properties of the graph
- combining goal-directed and hierarchical techniques is most promising
- applying goal-direction only to higher levels is sufficient

Several (prominent) combinations



Time for questions

Thank you, for your attention!

References

[bdsssw08] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes and D. Wagner, *Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm*, WEA'08, 2008

[pajor08] T. Pajor, "Speed-Up Techniques for Shortest Path Queries in Timetable Networks, Student Research Project, 2008

[schiefer08] D. Schieferdecker, *Systematic Combination of Speed-Up Techniques for Exact Shortest Path Queries*, Diploma Thesis, 2008