Karlsruher Institut für Technologie

Algorithmen für Routenplanung

11. Vorlesung, Sommersemester 2011

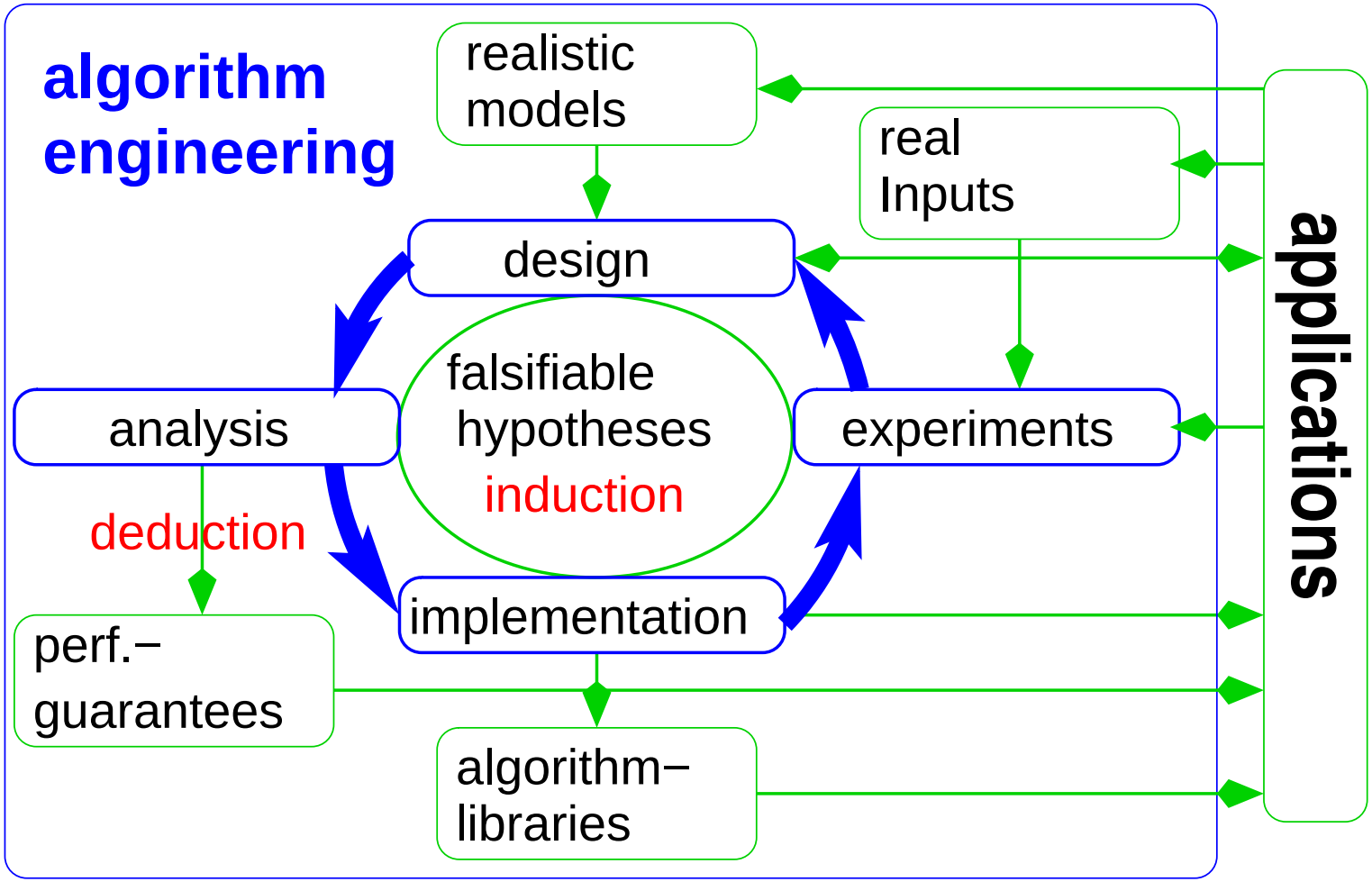
Reinhard Bauer | 30. Mai 2011

Institut für Theoretische Informatik · Algorithmik I · Prof. Dr. Dorothea Wagner

Inhalt

- Vorlesungsbefragung
- Modellierungsaspekte:
  ‘Komplexität der Vorberechnung von Beschleunigungstechniken’
- Modelle für Straßengraphen

Algorithm Engineering



Theoretische Algorithmik

- Theoretische Ergebnisse sollen Rückschlüsse auf praktischen Einsatz von Algorithmen ermöglichen
- **Modellannahme:** Theorie arbeitet notwendigerweise auf einem Modell, das die Wirklichkeit abbilden soll
  - Turing-Maschine
  - Real-RAM
  - Von-Neumann-Rechner
  - Comparison-Based Models

Theoretische Algorithmik

- Problem: Moderne Rechner weitaus komplexer als Modelle:
  - Cache-Effekte
  - Branch-Prediction
  - Parallelität
  - Speicherverwaltung
  - Konstanten können in der Praxis sehr groß sein
- Wichtig, dass
  - das Model zur Aufgabenstellung passt
  - die Ergebnisse richtig interpretiert werden.

Theorie für Algorithm Engineering

- Algorithm Engineering benutzt u.a. folgende Mittel:
  - Feintuning für bestimmte Problemklassen
  - viele kleine heuristische Verbesserung
  - Kombinationen von verschiedenen Techniken
  - machmal rechnernahes Programmieren
- Problem: Verfahren sind häufig zu komplex um sie in ihrer Gesamtheit theoretisch zu betrachten

Wie finde ich ein gutes Modell?

- Ein gutes Modell soll
  - **zielorientiert** sein:
    Eine bestimmte Fragestellung soll durch das Modell beantwortet werden können.
  - **vollständig** sein:
    Alle für die Fragestellung wichtigen Aspekte sollen beinhaltet sein.
  - **knapp und übersichtlich** sein:
    Unnötige Aspekte sollen vernachlässigt werden.

Preprocessing Speed-Up Techniques is Hard

Fragestellung:

- Die Vorberechnungsphase einer Beschleunigungstechnik hat meistens offene Freiheitsgrade
  - Arc-Flags: Wahl der Zellen
  - REAL / SHARC: Welche zusätzlichen Shortcuts sollen eingefügt werden?
- In der Praxis werden Heuristiken eingesetzt um diese Freiheitsgrade zu füllen
- Frage: Wie füllt man den Freiheitsgrad *möglichst* gut?

Beispiel: Contraction Hierarchies

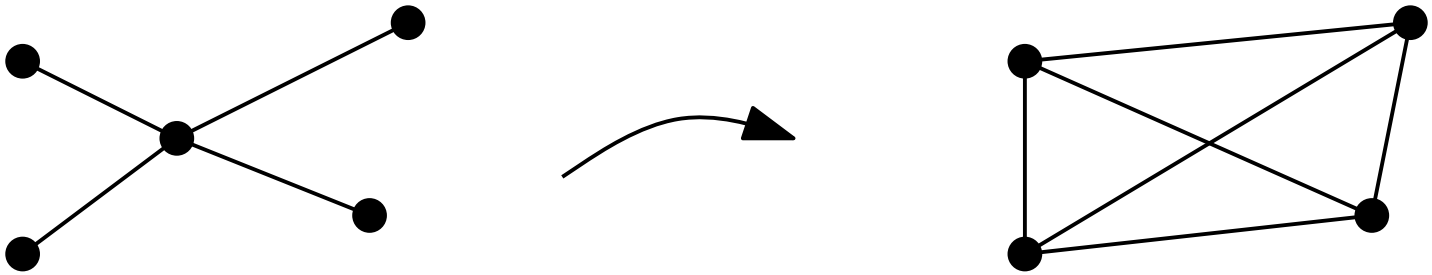
Aufgabe:

Erstelle formales Modell für vorherige Fragestellung.

Wiederholung: CH

How to contract vertex v

delete v
for *each pair* u, w *of neighbours of* v **do**
  use a limited local search heuristic to find a witness that (u, v, w) is *not* the only shortest path
  if (u, v, w) *'might be' the only shortest* u-w-*path* **then**
    insert edge (u, w) with $\text{len}(u, w) = \text{len}(u, v) + \text{len}(v, w)$



Wiederholung: CH

The CH-Preprocessing

- edgeDiff(u) gives the difference of edges in the graph after contraction u
- betweenness(u) measures the centrality of u
- estCostOfQuery(u) estimates the cost of the query when now contracting u
- #delNeighbours(u) count the number of already removed neighbours of u

1 **while** *there are vertices* **do**
2 | choose a vertex v with minimal α edgeDiff(u) + β betweenness(u) + γ estCostOfQuery(u) + δ #delNeighbours(u)
3 | contract v
4 output: set E^+ of edges inserted during contraction
5 output: contraction order

Wiederholung: CH

The CH-Query

- split search graph $G = (V, E)$ into
  - $G_\uparrow = (V, E_\uparrow)$ with $E_\uparrow = \{(u, v) \in E | v \text{ contracted after } u\}$
  - $G_\downarrow = (V, E_\downarrow)$ with $E_\downarrow = \{(u, v) \in E | v \text{ contracted before } u\}$
- Bidirectional Search
  - forward search on $G_\uparrow$
  - backward search on $G_\downarrow$
- Distance Balanced
- use stall on demand to additionally prune the search space
- stop search in one direction, when smallest element in queue is at least the length of the shortest path found so far

Modell für CH

Vorschläge?

Modell für CH

Problemstellung informell:

Fülle den Freiheitsgrad der Vorberechnungsphase von Contraction Hierarchies möglichst gut.

- Was ist der Freiheitsgrad?
- Was ist möglichst gut?

Modell für CH

Problemstellung informell:

Fülle den Freiheitsgrad der Vorberechnungsphase von Contraction Hierarchies möglichst gut.

- Was ist der Freiheitsgrad?
  Die Kontraktionsreihenfolge.
- Was ist möglichst gut? Eine möglichst schnelle Query. Hier gibt es viele Möglichkeiten:
  - durchschnittliche Queryzeit vs. worst-case Queryzeit
  - Maß für Geschwindigkeit der Query: Suchraumgröße.
    - gesehene Knoten, besuchte Knoten
    - gesehene Kanten, besuchte Kanten
  - Wahl von s, t (zufällig, gleichverteilt?)

Modell für CH

Problemstellung informell:

Fülle den Freiheitsgrad der Vorberechnungsphase von Contraction Hierarchies möglichst gut.

Finde eine möglichst gute Kontraktionsfolge für die Contraction Hierarchies Vorberechnung, so dass der erwartete Suchraum (gemessen in besuchten Knoten) von zwei gleichverteilt, zufällig gewählten Knoten minimal ist.

Modell für CH

Verbleibendes Problem: Die Technik ist immer noch sehr kompliziert.

How to contract vertex v

delete v
for *each pair* u, w *of neighbours of* v **do**
  use a limited local search heuristic to find a witness that (u, v, w) is *not* the only shortest path
  if (u, v, w) *'might be' the only shortest* u-w-*path* **then**
    insert edge (u, w) with $\text{len}(u, w) = \text{len}(u, v) + \text{len}(v, w)$

Modell für CH

Verbleibendes Problem: Die Technik ist immer noch sehr kompliziert.

How to contract vertex v

1 delete v
2 **for** *each pair* u, w *of neighbours of* v **do**
3 |
4 | use a limited local search heuristic to find a witness that (u, v, w) is *not* the only shortest path
5 |
6 | **if** (u, v, w) *'might be' the only shortest* u-w-*path* **then**
7 | | insert edge (u, w) with $\text{len}(u, w) = \text{len}(u, v) + \text{len}(v, w)$

Modell für CH

Verbleibendes Problem: Die Technik ist immer noch sehr kompliziert.

How to contract vertex v

1 delete v
2 **for** *each pair* u, w *of neighbours of* v **do**
3 | **if** (u, v, w) *is the only shortest* u-w-*path* **then**
4 | | insert edge (u, w) with $\text{len}(u, w) = \text{len}(u, v) + \text{len}(v, w)$

Modell für CH

The CH-Preprocessing

- edgeDiff(u) gives the difference of edges in the graph after contraction u
- betweenness(u) measures the centrality of u
- estCostOfQuery(u) estimates the cost of the query when now contracting u
- #delNeighbours(u) count the number of already removed neighbours of u

1 **while** *there are vertices* **do**
2 | choose a vertex v with minimal α edgeDiff(u) + β betweenness(u) + γ estCostOfQuery(u) + δ #delNeighbours(u)
3 | contract v
4 output: set E^+ of edges inserted during contraction
5 output: contraction order

Modell für CH

The CH-Preprocessing

1 **while** *there are vertices* **do**
2 | choose a vertex v with minimal α edgeDiff(u) + β betweenness(u) + γ estCostOfQuery(u) + δ #delNeighbours(u)
3 | contract v
4 output: set E^+ of edges inserted during contraction
5 output: contraction order

Modell für CH

The CH-Preprocessing

distuingish a contraction order $<$
while *there are vertices* **do**
    choose next vertex v according to $<$
    contract v
output: set E^+ of edges inserted during contraction
output: contraction order

Modell für CH

The CH-Query

- split search graph $G = (V, E \cup E^+)$ into
  - $G_\uparrow = (V, E_\uparrow)$ with $E_\uparrow = \{(u, v) \in E \cup E^+ | v \text{ contracted after } u\}$
  - $G_\downarrow = (V, E_\downarrow)$ with $E_\downarrow = \{(u, v) \in E \cup E^+ | v \text{ contracted before } u\}$
- Bidirectional Search
  - forward search on $G_\uparrow$
  - backward search on $G_\downarrow$
- Distance Balanced
- use stall on demand to additionally prune the search space
- stop search in one direction, when smallest element in queue is at least the length of the shortest path found so far

Modell für CH

The CH-Query

- split search graph $G = (V, E \cup E^+)$ into
  - $G_\uparrow = (V, E_\uparrow)$ with $E_\uparrow = \{(u, v) \in E \cup E^+ | v \text{ contracted after } u\}$
  - $G_\downarrow = (V, E_\downarrow)$ with $E_\downarrow = \{(u, v) \in E \cup E^+ | v \text{ contracted before } u\}$
- Bidirectional Search
  - forward search on $G_\uparrow$
  - backward search on $G_\downarrow$
- Distance Balanced
- use stall on demand to additionally prune the search space
- stop search in one direction, when smallest element in queue is at least the length of the shortest path found so far

Modell für CH

The CH-Query

- split search graph $G = (V, E \cup E^+)$ into
  - $G_\uparrow = (V, E_\uparrow)$ with $E_\uparrow = \{(u, v) \in E \cup E^+ | v \text{ contracted after } u\}$
  - $G_\downarrow = (V, E_\downarrow)$ with $E_\downarrow = \{(u, v) \in E \cup E^+ | v \text{ contracted before } u\}$
- Bidirectional Search
  - forward search on $G_\uparrow$
  - backward search on $G_\downarrow$
- Distance Balanced

Modell für CH

The CH-Query

- split search graph $G = (V, E \cup E^+)$ into
  - $G_\uparrow = (V, E_\uparrow)$ with $E_\uparrow = \{(u, v) \in E \cup E^+ | v \text{ contracted after } u\}$
  - $G_\downarrow = (V, E_\downarrow)$ with $E_\downarrow = \{(u, v) \in E \cup E^+ | v \text{ contracted before } u\}$
- Bidirectional Search
  - forward search on $G_\uparrow$
  - backward search on $G_\downarrow$

Modell für CH

The CH-Query

- split search graph $G = (V, E \cup E^+)$ into
  - $G_\uparrow = (V, E_\uparrow)$ with $E_\uparrow = \{(u, v) \in E \cup E^+ | v \text{ contracted after } u\}$
  - $G_\downarrow = (V, E_\downarrow)$ with $E_\downarrow = \{(u, v) \in E \cup E^+ | v \text{ contracted before } u\}$
- Bidirectional Search
  - forward search on $G_\uparrow$
  - backward search on $G_\downarrow$

Modell für CH

The CH-Query

- perform two Dijkstra's searches
  - starting from s on $G_\uparrow = (V, E_\uparrow)$ with $E_\uparrow = \{(u, v) \in E | v \text{ contracted after } u\}$
  - starting from t on $G_\downarrow = (V, E_\downarrow)$ with $E_\downarrow = \{(u, v) \in E | v \text{ contracted before } u\}$

Modell für CH

The CH-Query

- perform two Dijkstra's searches
  - starting from s on $G_\uparrow = (V, E_\uparrow)$ with $E_\uparrow = \{(u, v) \in E | v \text{ contracted after } u\}$
  - starting from t on $G_\downarrow = (V, E_\downarrow)$ with $E_\downarrow = \{(u, v) \in E | v \text{ contracted before } u\}$

Suchraum einer s-t-query ist also

- Knoten erreichbar von s in $G_\uparrow$
- Knoten erreichbar von t in $G_\downarrow$

Modell für CH - Zusammenfassung

Given a graph $G = (V, E)$, find a total order $<$ on V such that

$$\sum_{s \in V} \nu^+(s) + \nu^-(s)$$

is minimal, where

$$\begin{aligned} \nu^+(s) &:= \text{number of nodes reachable from } s \text{ in} \\ &\quad (V, \{(u,v) \in E \cup E^+ | v > u\}) \\ \nu^+(s) &:= \text{number of nodes reachable from } s \text{ in} \\ &\quad (V, \{(u,v) \in E \cup E^+ | v < u\}) \end{aligned}$$

and E^+ is the set of edges inserted into G by the simplified CH-preprocessing algorithm.

Literatur

Bauer, Columbus, Katz, Krug, Wagner:

Preprocessing Speed-Up Techniques is Hard

CIAC 2010

- Überträgt obiges Szenario auf andere Techniken
- Betrachtet die Komplexität der resultierenden Probleme
  Ergebnis: NP-schwer für alle Techniken
- Beweise sehr technisch deswegen hier nur eine kleine Skizze

Techniques Considered

- **ALT**
  (Goldberg, Harrelson)
  by reduction from 3-Minimum Cover
- **Reach-Based Pruning**
  (Gutman, Goldberg, Kaplan, Werneck)
  by reduction from Exact Cover by 3-Sets
- **Highway-Node Routing, Min-Overlay Graph**
  (Holzer, Schulz, Wagner, Schultes, Sanders)
  by reduction from Exact Cover by 3-Sets
- **Contraction Hierarchies**
  (Geisberger, Sanders, Schultes, Delling)
  by reduction from Vertex Cover
- **Arc-Flags / Sharc**
  (Lauther, Hilger, Köhler, Möhring, Schilling, Bauer, Delling)
  by reduction from 3-Partition and Exact Cover by 3-Sets

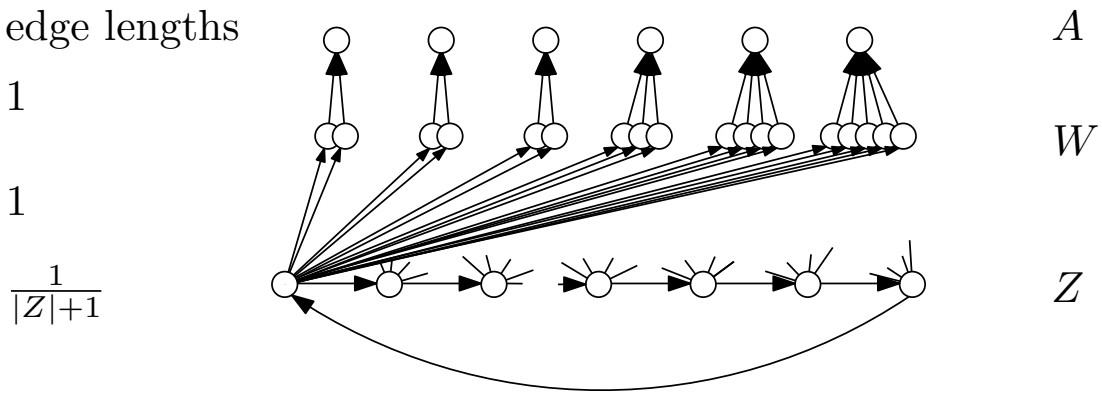
Techniques Not Considered

- **Bidirectional Search**
  (no degree of freedom)
- **Highway Hierarchies**
  (Sanders, Schultes)
  only tuning-paramenters, no degree of freedom
- **Transit Node Routing**
  (Bast, Funke, Matijevic, Sanders, Schultes)
  also the query is a degree of freedom
- **Geometric Container**
  (Schulz, Wagner, Weihe, Willhalm, Zaroliagis)
  no degree of freedom that does not affect the query

Was war nochmal NP-schwer?

- *NP*: Klasse der Probleme die sich von einer nichtdeterministischen Turingmaschine in polynomieller Zeit lösen lassen
- Problem *L* ist *NP*-schwer: alle Probleme in *NP* lassen sich polynomiell auf *L* reduzieren
- Problem *L* ist *NP*-vollständig: Problem *L* ist in *NP* und Problem *L* ist *NP*-schwer

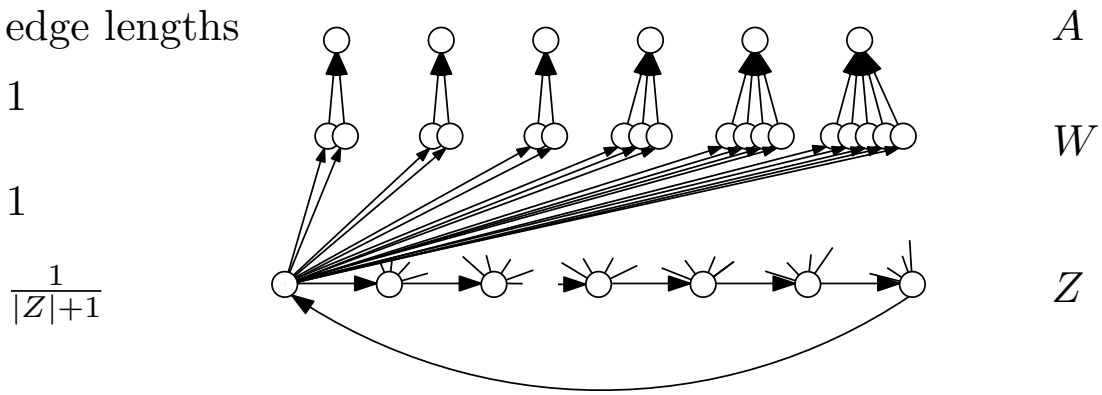
Arc-Flags: NP-hardness



reduction from strongly NP-hard problem 3-PARTITION

- given:
  - a set A of $3m$ elements
  - a bound $B \in \mathbb{N}$
  - a size w_a for each $a \in A$
  - such that $B/4 < w_a < B/2$ and such that $\sum w_a = mB$
- question:
  - can A be partitioned into m disjoint sets of size B each

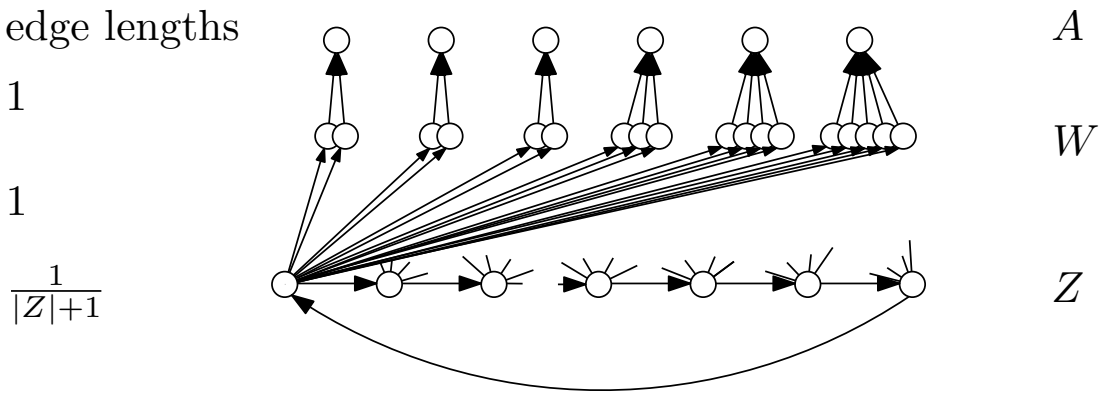
Arc-Flags: NP-hardness



transformation

- given the 3-Partition instance $A = \{a_1, \ldots, a_{3m}\}$, b, sizes w_a for each $a \in A$
- for each element a, insert a node a and w_a nodes pointing to a
- insert a directed cycle Z
- set the number of cells to be $m + 1$

Arc-Flags: NP-hardness

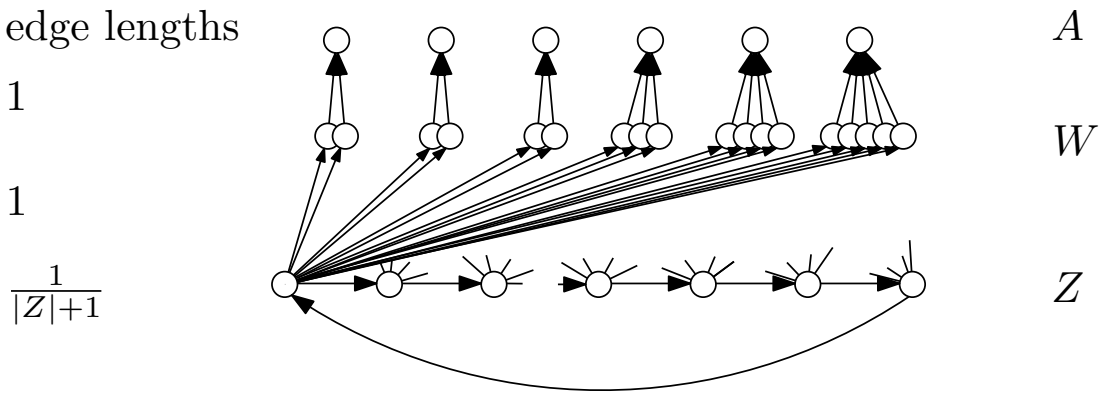


steps of the proof

- Let $Z > 6|A \cup W|^3$.
  Then there is an optimal $(m+1)$-partition $\mathcal{V} = (V_1, \ldots, V_{m+1})$ with $V_{m+1} = Z$.
- Decompose the objective function into

$$\underbrace{\sum_{s,t \notin Z} V_{\mathcal{V}^*}(s,t)}_{\leq |W \cup A|^3} + \underbrace{\sum_{\substack{s,t \in Z \\ s \notin Z, t \in Z}} V_{\mathcal{V}^*}(s,t)}_{=:\alpha} + \underbrace{\sum_{s \in Z, t \in A} V_{\mathcal{V}^*}(s,t)}_{=:\beta} + \underbrace{\sum_{s \in Z, t \in W} V_{\mathcal{V}^*}(s,t)}_{=:\gamma}$$

Arc-Flags: NP-hardness



$$\underbrace{\sum_{s,t\notin Z} V_{\mathcal{V}^*}(s,t)}_{\leq |W\cup A|^3} + \underbrace{\sum_{\substack{s,t\in Z\\ s\notin Z,t\in Z}} V_{\mathcal{V}^*}(s,t)}_{=:\alpha} + \underbrace{\sum_{s\in Z,t\in A} V_{\mathcal{V}^*}(s,t)}_{=:\beta} + \underbrace{\sum_{s\in Z,t\in W} V_{\mathcal{V}^*}(s,t)}_{=:\gamma}$$

- value of α independent of the partition
- Let $\beta^* := |Z| \sum_{i=1}^{m} (3(3+1)/2 + 3mB + 1)$ and $\gamma^* := |Z| \sum_{i=1}^{m} (B(B+1)/2 + 1)$.
- Then $\sum_{s,t\in V} V_{\mathcal{V}^*}(s,t) \leq |W \cup A|^3 + \alpha + \beta^* + \gamma^*$ if, and only if the 3-Partition instance $(A, \{w_a : a \in A\})$ is a yes-instance.

Modelle für Straßengraphen

- Beschleunigungstechniken sind häufig für Straßengraphen maßgeschneidert
- Intuition: Straßengraphen beinhalten eine Art Hierarchie
- Für tieferes Verständnis der Algorithmen ist ein tieferes Verständnis der Instanzen hilfreich
- Literatur
  Bauer, Krug, Meinert, Wagner:
  Synthetic Road Networks,
  AAIM 2010

Was sind typische Eigenschaften von Straßengraphen?

Was sind typische Eigenschaften von Straßengraphen?

dataset	origin	represents	#nodes	#edges	density	dir'ness
TIGER	U.S administration	USA	24,412,259	58,596,814	2.40	.0 %
PTV	commercial data	Europe	31,127,235	69,200,809	2.22	4.9 %
OSM	collaborative project	Europe	48,767,450	99,755,206	2.05	3.5 %
ABR	Abraham et al	synthetic	15,000,000	43,573,536	2.90	.0 %
VOR	Voronoi generator	synthetic	42,183,476	93,242,474	2.21	.0 %

Was sind typische Eigenschaften von Straßengraphen?

Tabelle: Relative sizes (measured in number of nodes) of the k biggest SCCs.

dataset	k = 1	2	5	20	100	total # of SCCs
OSM	.80	.91	.94	.96	.97	541264
PTV	.97	.97	.97	.97	.97	924561
TIGER	.98	.98	.99	.99	.99	89796

Was sind typische Eigenschaften von Straßengraphen?

Tabelle: Degree distribution of the datasets: Relative frequency according to degree.

	degree						
dataset	0	1	2	3	4	5	≥ 6
OSM	.001	.097	.773	.114	.015	0	0
PTV	.030	.247	.245	.429	.049	.001	0
TIGER	0	.205	.301	.386	.106	.001	0
ABR	0	.324	.254	.160	.093	.056	.077
VOR	0	.100	.601	.292	.005	.002	0

Qualitätssicherung!

- Wie kann man Graphen generieren, die ähnlich zu Realwelt-Straßengraphen sind?
- Wie überprüft man diese Ähnlichkeit?

Skizze - Ansatz von Krug et al.

- Basiert auf Voronoi-Partitionierung, Hierarchie und Graph-Spannern

Algorithmus

- Starte mit beliebigem Polygon
- Für jede Hierarchiestufe i
  - wähle die γ_i kleinsten Facetten
  - unterteile diese Facetten

  berechne t-Spanner des Resultats

Wie unterteile ich ein Polygon?

- für $i = 1, \ldots,$ Anzahl neue Zentren
  - wähle zufällig neues Zentrum x
  - wähle zufällig Radius r um x
  - wähle zufällig Anzahl m von Punkten in Kreis $K(x, r)$
  - wähle zufällig die m Punkte in $K(x, r)$
- berechne Voronoi-Diagramm des Resultats

Ansatz von Krug et al. - Hilfsroutine

Input: $P, n, \mathcal{D}, \mathcal{R}$

1 $C \leftarrow \emptyset$;
2 **for** $i = 1$ **to** n **do**
3 $\quad x \leftarrow$ choose uniform point inside P;
4 $\quad \alpha \leftarrow$ random value chosen according to $\mathcal{D}$;
5 $\quad r \leftarrow$ random value chosen according to $\mathcal{R}$;
6 $\quad m \leftarrow \lceil r^{\alpha} \rceil$;
7 $\quad C \leftarrow C \cup \{x\}$;
8 $\quad$ **for** $j = 1$ **to** m **do**
9 $\quad\quad p \leftarrow$ choose random point in $R(x, r)$;
10 $\quad\quad C \leftarrow C \cup \{p\}$;
11 $\quad$ compute Voronoi diagram of C in P;

Algorithm 1: SUBDIVIDE-POLYGON

Ansatz von Krug et al. - Phase 1

Input: ℓ, $\gamma_i, \mathcal{C}_i, \mathcal{R}_i, \mathcal{D}_i$, $1 \leq i \leq \ell$

1 $\ell_0 \leftarrow 1$;
2 $S \leftarrow P$;
3 **for** $i = 1$ **to** ℓ **do**
4 $\quad m \leftarrow \gamma_{i-1}|S|$;
5 $\quad S \leftarrow$ smallest m faces in S;
6 $\quad S' \leftarrow \emptyset$;
7 $\quad$ **for** $f \in S$ **do**
8 $\quad\quad n \leftarrow$ choose according to distribution $\mathcal{C}_i$;
9 $\quad\quad S' \leftarrow S' \cup$ SUBDIVIDE-POLYGON $(P(f), n, \mathcal{D}_i, \mathcal{R}_i)$;
10 $\quad S \leftarrow S'$

Algorithm 2: Voronoi-Roadnetwork

Ansatz von Krug et al - Phase 2

- Given a graph G, a graph spanner H of G with stretch t is a subgraph of G such that for each pair of nodes u, v in G we have

$$\text{dist}_H(u, v) \leq t \cdot \text{dist}_G(u, v)$$

.

- Gegeben den Graph aus Phase 1, wähle (sortiert in nicht-absteigender Länge) greedy Kanten so aus, dass der resultierende Graph ein t-Spanner ist.

Skizze - Generator von Abraham et al

Starte mit leerem Polygon P
solange nicht 'genügend' Punkte

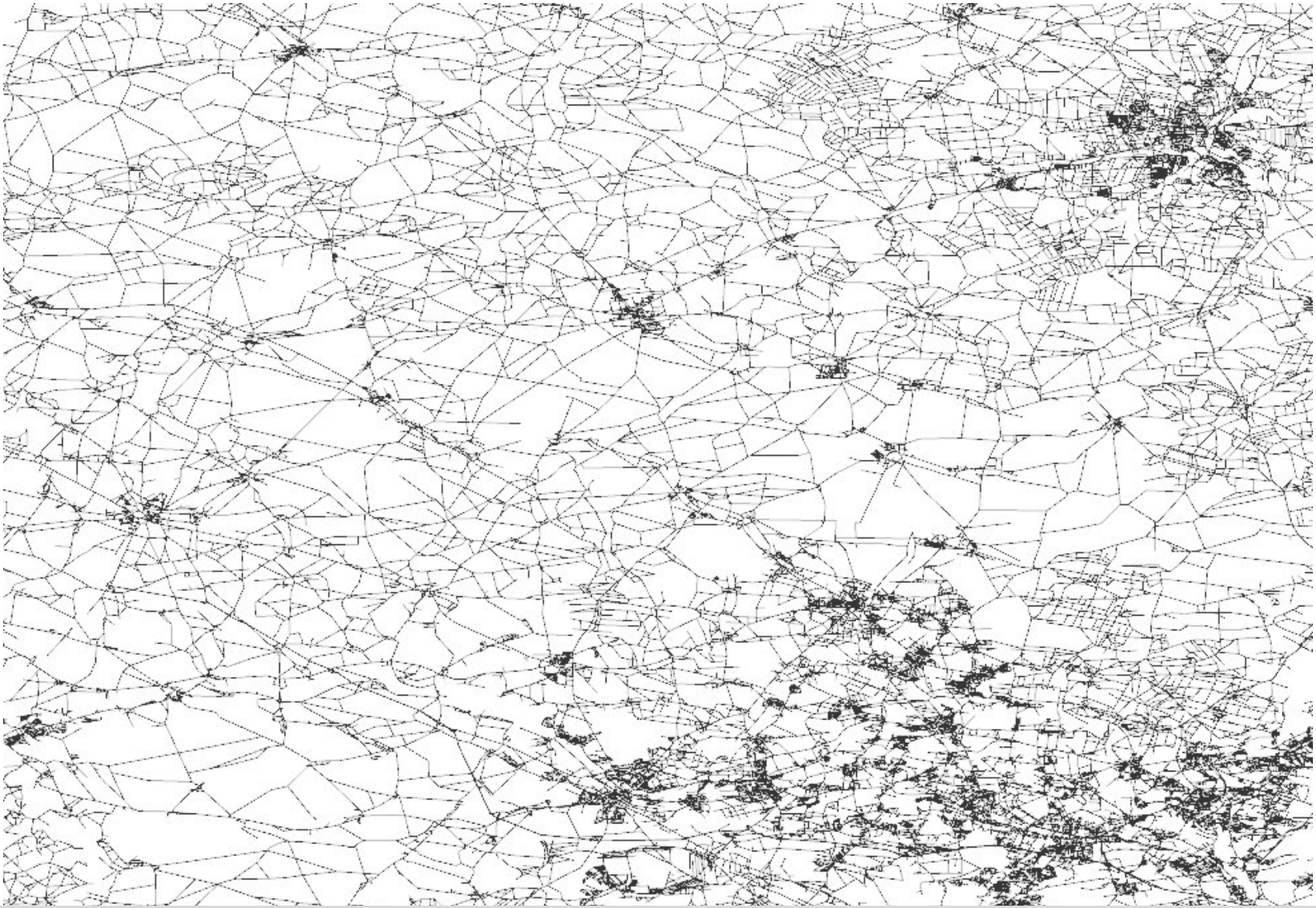
- wähle zufällig neuen Punkt p in P
- für jede Hierarchiestufe i aufsteigend
  - wenn p in Stufe i nicht gecovert
    - füge p in Stufe i ein
    - verbinde p mit 'Nachbarn'
  - verbinde P mit nächsten Knoten in Stufe i und $i + 1$

Generator of Abraham et al

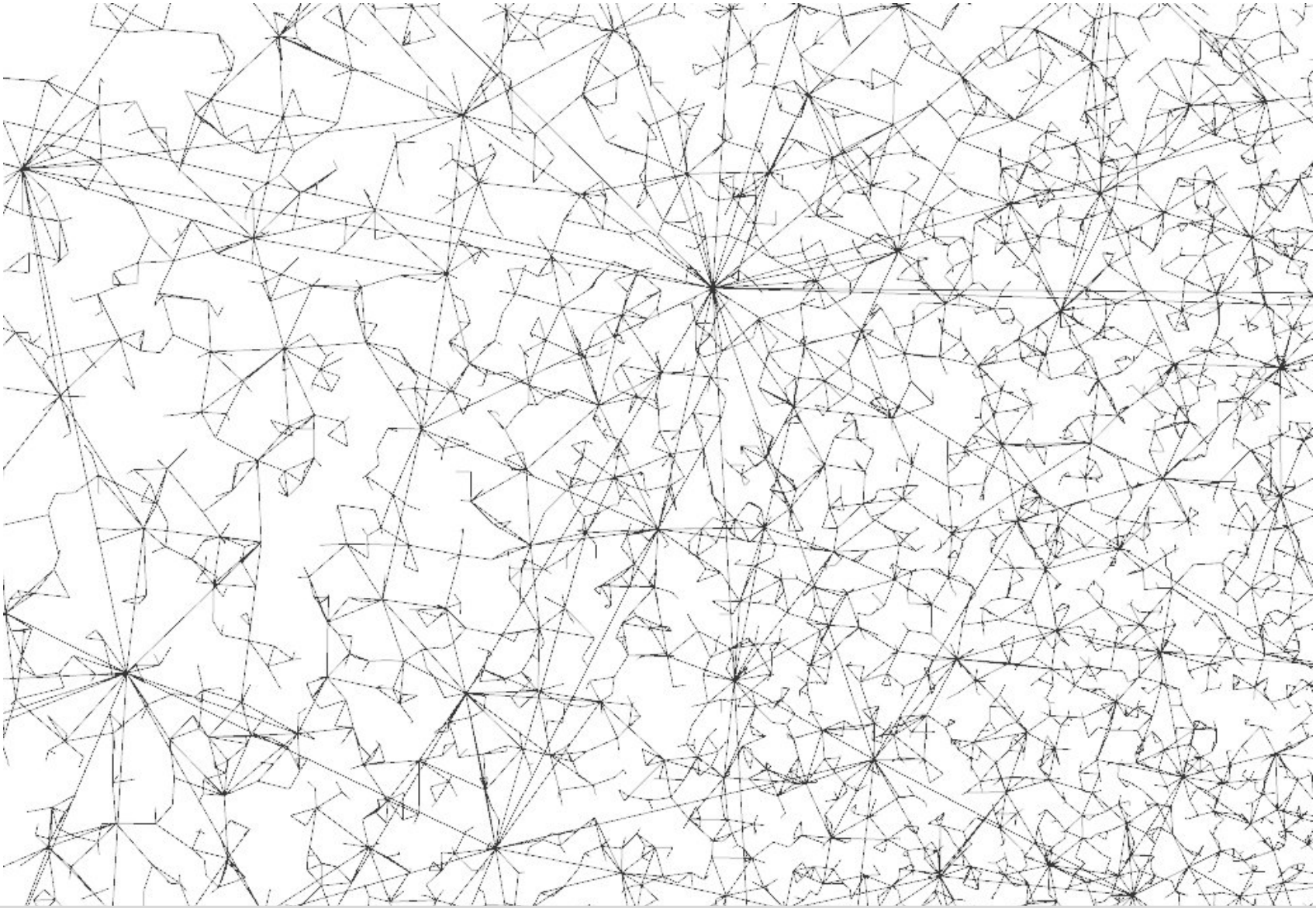
input : number of vertices n
output: Graph (V, E)

1 initialize $V = C_0, \ldots, C_{\log D}, E$ to be $\emptyset$;
2 **for** $t = 1$ *to* n **do**
3 $\quad v_t \leftarrow \text{rand}()$;
4 $\quad V \leftarrow V \cup \{v_t\}$;
5 $\quad$ **for** $i = \log D$ *to* 1 **do**
6 $\quad\quad$ **if** $d(v_t) > 2^i$ *for each* $w \in C_i$ **then**
7 $\quad\quad\quad C_i \leftarrow C_i \cup \{v_t\}$;
8 $\quad\quad\quad$ **for** $w \in C_i$ **do**
9 $\quad\quad\quad\quad$ **if** $d(v_t, w) \leq k \cdot 2^i$ **then** $E \leftarrow E \cup \{v_t, w\}$
10 $\quad\quad$ $w \leftarrow$ closest point of v_t in C_{i+1} ;
11 $\quad\quad$ **if** $v_t \neq w$ **then** $E \leftarrow E \cup \{v_t, w\}$
12 set edge weights such that $\text{len}(u, v) = d(u, v)^{1-\delta}$ for $\delta = 1/8$

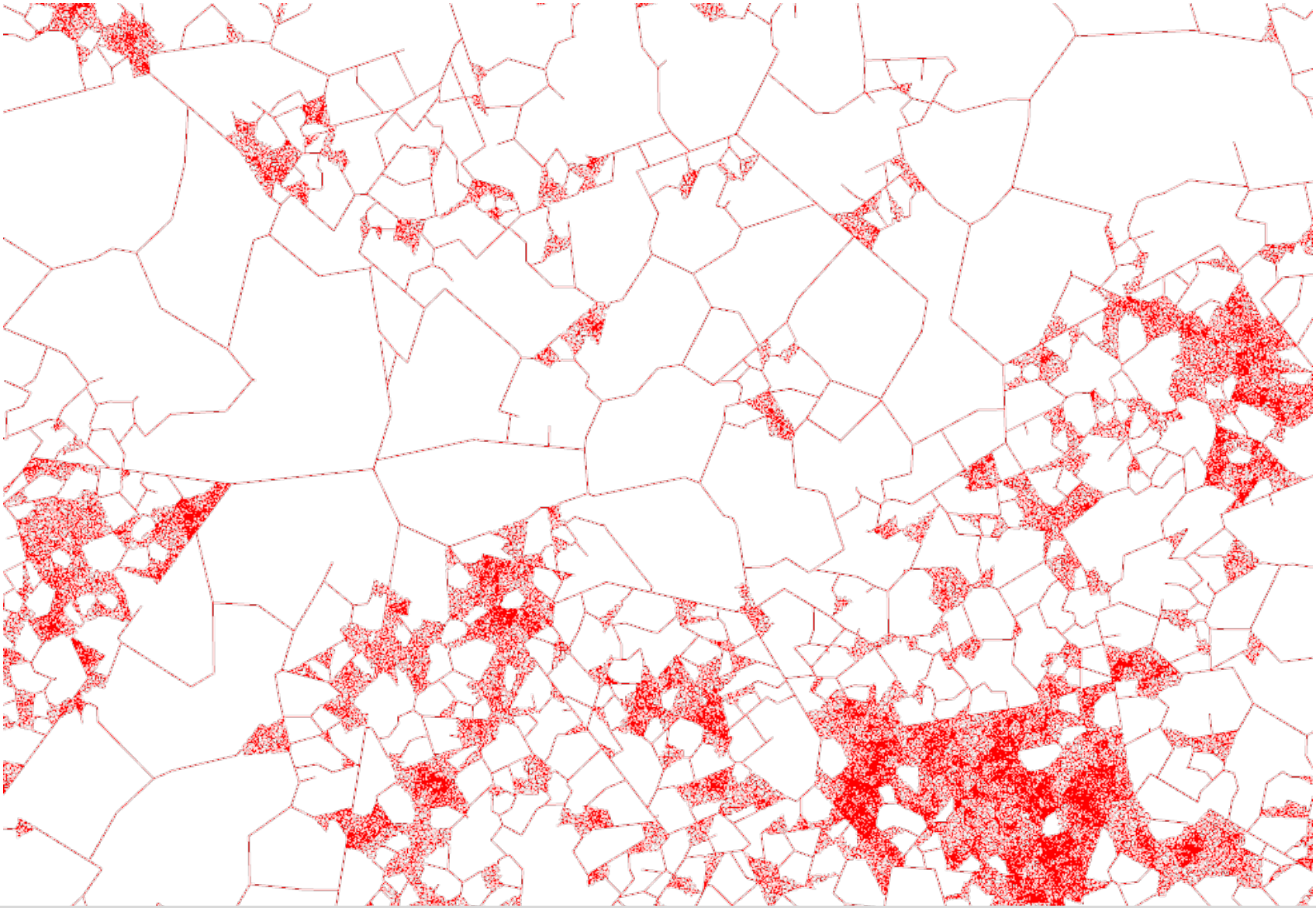
Beispielgraph: Realwelt



Beispielgraph: Abrahams et al



Beispielgraph: Krug et al



Algorithmic Behaviour

Tabelle: Speedups of the tested Point-To-Point Shortest Path Algorithms.

technique	#nodes	OSM	TIGER	PTV	VOR	ABR	UD	GRID
ALT	2000	1.9	3.9	5.1	4.9	9.3	5.9	8.1
ALT	32000	5.5	9.6	10.7	11.3	16.4	12.3	12.6
ALT	128000	10.8	10.8	18.2	13.8	31.4	21.7	22.2
ALT	256000	15.3	13.4	26.9	16	35.8	26.1	25.6
ALT	512000	16	17.7	30.6	21.6	37.3	28.8	24.2
Arc-Flags	2000	4.4	14.5	20.7	20.5	48.6	29.1	35.7
Arc-Flags	32000	21.8	55.9	68.5	89.7	131.5	44.1	32.7
Arc-Flags	128000	48.7	65.7	142.2	111.7	219.9	64.4	52.3
Arc-Flags	256000	69.5	76.2	219.9	143.2	245.6	74	72
CH	2000	24.3	16.5	20.7	13.9	13.7	9	7.5
CH	32000	100.4	80.5	102.5	114.9	42.2	38.3	29.1
CH	128000	229	209.2	390.2	346.4	184.6	134.2	74.5
CH	256000	435.2	357.3	626.1	684.7	327.8	195.7	128.2
CH	512000	459.5	497.9	831.4	1565.5	474.2	271.8	175.6
Reach	2000	3.9	3.3	3.8	3.3	4.2	1.8	1.2
Reach	32000	10.1	7.5	9.1	9.9	5.9	2.9	2.1
Reach	128000	28.2	15.7	24.2	22.3	10.1	4.2	2.9
Reach	256000	37.8	22.8	31.8	32.9	10	4.4	4
Reach	512000	34.3	16.3	27.1	49.2	10.7	5	4.7