

Engineering Dijkstra's Shortest Path Algorithm

VO Algorithm Engineering

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

6. VO

14.11.2005

Literatur für diese VO

- C. Demetrescu und G.F. Italiano: Engineering Shortest Path Algorithms, C.C. Ribeiro und S.L. Martins (Eds.): Workshop on Experimental Algorithmics 2004 (WEA), LNCS 3059, Springer, 2004, 191-198

Überblick

- Dijkstra's SSSP-Algorithmus
- Theoretische Analyse: Korrektheit und Laufzeit

- Variation von Dijkstra für APSP
- Theoretische Analyse: Korrektheit und Laufzeit

- Einführung von LSP's
- Eigenschaften von LSP's

- Experimentelle Resultate

Kürzeste Wege Probleme

Single-Source Shortest Path (SSSP)

- Geg.: Graph $G=(V,E)$ mit Kantenkosten $w \in \mathbb{R}$, keine negativen Kreise, $s \in V$
- Gesucht: Kürzeste (s,v) -Wege in G für alle $v \in V$

All-Pairs Shortest Path (APSP)

- Geg.: Graph $G=(V,E)$ mit Kantenkosten $w \in \mathbb{R}$, keine negativen Kreise
- Gesucht: Kürzeste Wege zwischen allen Knotenpaaren (Distanzmatrix)

Dijkstra's Algorithmus

- Graph G gerichtet mit Kantenkosten $w \geq 0$
- Sei M = „große Zahl“
- Q sei Prioritätswarteschlange mit Operationen
 - $\text{InsertPrioQ}(v, \text{dist}(v))$: fügt v ein mit Priorität $\text{dist}(v)$
 - $\text{ExtractMinQ}()$: Gibt das Minimum zurück und entfernt es aus Q
 - $\text{DecreasePrioQ}(v, \text{dist}(v))$: Aktualisiert die Priorität von v in Q auf $\text{dist}(v)$
- Sei $\text{dist}(v)$ die Distanzmatrix

Dijkstra's Algorithmus

- $Q \leftarrow \emptyset$; $\text{dist}[s]=0$; $\text{InsertPrioQ}(s,0)$ // $S \leftarrow \{s\}$
- Für alle Knoten $x \neq s$:
 - Setze $\text{dist}[x]=M$; $\text{InsertPrioQ}(x,M)$
- Solange $Q \neq \emptyset$:
 - $x \leftarrow \text{ExtractMinQ}()$; // Addiere x zu S ;
 - Für alle Kanten (x,y) , die x verlassen: ★
 - Falls $\text{dist}[y] > \text{dist}[x] + w[x,y]$:
 - $\text{dist}[y] \leftarrow \text{dist}[x] + w(x,y)$;
 - $\text{DecreasePrioQ}(y, \text{dist}[y])$
- Return $\text{dist}[]$

★: edge relaxation

Analyse: Korrektheit

- Induktion: Nach jeder Iteration ist V in 2 Mengen aufgeteilt: S und $T := V \setminus S$
- Ind.-Ann.:
 - (1) Für $x \in S$ ist $\text{dist}(x) = \text{Länge des kürzesten } (s, x)\text{-Weges}$
 - (2) für $x \in T$ ist $\text{dist}(x) = \text{Länge des kürzesten } (s, x)\text{-Weges}$ bei dem jeder Knoten außer x zu S gehört.
- Ind.-Schritt:
 - Knoten x mit kleinstem dist -Wert wird in S aufgenommen. Ist korrekt, denn: Falls ein kürzerer Weg existieren würde, dann würde dieser einen ersten Knoten in T benutzen; aber dieser müßte weiter weg von s sein, denn sein dist -Wert ist größer als der von x . Die „edge relaxation“ sorgt dafür, dass (2) erfüllt ist.

Analyse: Laufzeit

- Sei $n=|V|$ und $m=|E|$
- $T = n * T(\text{InsertPrioQ}()) + n * T(\text{ExtractMinQ}()) + m * T(\text{DecreasePrioQ}())$
- Binärer Heap für Q: $O((n+m) \log n)$
 - $T(\text{InsertPrioQ}()) = O(\log n)$
 - $T(\text{ExtractMinQ}()) = O(\log n)$
 - $T(\text{DecreasePrioQ}()) = O(\log n)$
- Fibonacci Heap für Q: $O(m+n \log n)$
 - $T(\text{InsertPrioQ}()) = O(1)$ amortisiert
 - $T(\text{ExtractMinQ}()) = O(\log n)$
 - $T(\text{DecreasePrioQ}()) = O(1)$ amortisiert

Interleaved Dijkstra für APSP

- Sei $\text{dist}(,)$ die Distanzmatrix
- Initialisierung:
 - Setze Prioritätswarteschlange $Q = \emptyset$

Annahme: Die kürzesten Wege sind eindeutig, d.h. es gibt nur jeweils einen kürzesten (x,y) -Weg zwischen jedem Knotenpaar.

- Für jedes Knotenpaar (x,y) :
 - Falls $(x,y) \in E$: $\text{dist}[x,y] \leftarrow w(x,y)$ Sonst $\text{dist}[x,y] = M$
 - InsertPrioQ((x,y) , $\text{dist}[x,y]$)
- Solange $Q \neq \emptyset$:
 - $(x,y) \leftarrow \text{ExtractMinQ}()$
 - Für jede Kante $(y,z) \in E$, die y verläßt: ★
 - Falls $\text{dist}[x,z] > \text{dist}[x,y] + w[y,z]$:
 - $\text{dist}[x,z] \leftarrow \text{dist}[x,y] + w(y,z)$;
 - DecreasePrioQ((x,z) , $\text{dist}[x,z]$)
 - Für jede Kante $(z,x) \in E$, die nach x zeigt: ★
 - Falls $\text{dist}[z,y] > w(z,x) + \text{dist}(x,y)$:
 - $\text{dist}[z,y] \leftarrow w(z,x) + \text{dist}[x,y]$;
 - DecreasePrioQ((z,y) , $\text{dist}[z,y]$)
- Return $\text{dist}[]$

Analyse: Laufzeit

- Sei $n=|V|$ und $m=|E|$
- $T = n^2 * T(\text{InsertPrioQ}()) + n^2 * T(\text{ExtractMinQ}())$
 $+ n * m * T(\text{DecreasePrioQ}())$
- Binärer Heap für Q : $O((n^2 + mn) \log n)$
 - $T(\text{InsertPrioQ}()) = O(\log n)$
 - $T(\text{ExtractMinQ}()) = O(\log n)$
 - $T(\text{DecreasePrioQ}()) = O(\log n)$
- Fibonacci Heap für Q : $O(mn + n^2 \log n)$
 - $T(\text{InsertPrioQ}()) = O(1)$ amortisiert
 - $T(\text{ExtractMinQ}()) = O(\log n)$
 - $T(\text{DecreasePrioQ}()) = O(1)$ amortisiert

Analyse: Korrektheit

- Falls $w \geq 0$ für alle Kanten, dann enthält die Distanzmatrix nach Ablauf des Algorithmus „Interleaved Dijkstra“ die korrekten Distanzen.
- Denn: In jedem Schritt $(x,y) \leftarrow \text{ExtractMinQ}()$ enthält $\text{dist}(x,y)$ die korrekte Distanz.
- In Q sind alle Knotenpaare enthalten.
- Beweis: Falls nicht, dann sei (a,b) ein solches Paar. Sei $P(u \rightarrow v)$ maximal korrekter Teil-Weg von $P(a \rightarrow b)$. Als dieser berechnet wurde war er mit korrektem dist -Wert in Q . Bei $\text{ExtractMinQ}()$ wurde er versucht zu verlängern. Dann hätte also auch $\text{Succ}(b)$ in $P(u \rightarrow v)$ den korrekten (einen kürzeren) Wert erhalten. Widerspruch.

Experimentelle Resultate

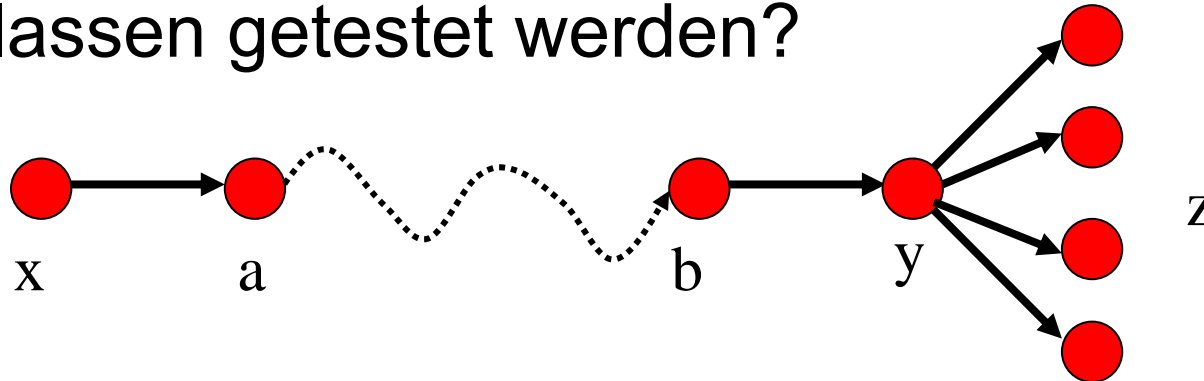
- Experimentelle Resultate zeigen, dass bei
 - dünnen Graphen ausgeklügelte Datenstrukturen eine wichtige Rolle spielen, während bei
 - dichten Graphen die Anzahl der getesteten Kanten in (★) die Laufzeit dominieren.

- ZIEL: Laufzeit für dichte Graphen reduzieren

- Suche nach Heuristiken um die Anzahl der getesteten Kanten zu reduzieren

Idee zur Reduktion

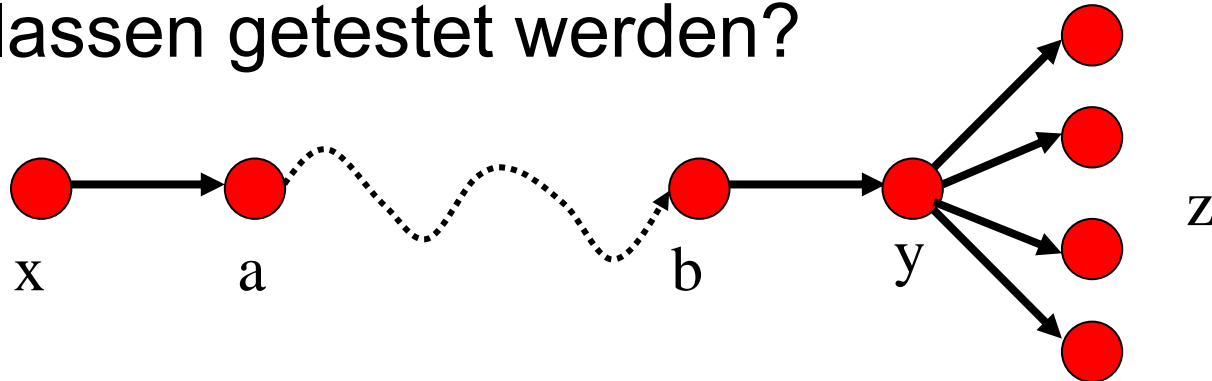
- Müssen tatsächlich alle Kanten (y,z) , die y verlassen getestet werden?



- Nicht, wenn diese nicht zu einem kürzesten Weg von x nach z gehören können.
- Jeder Teil-Weg eines kürzesten Weges ist ein kürzester Weg.
- Nur, wenn $P(x \rightarrow y)$ und $P(a \rightarrow z)$ (beide Teil-Wege von $P(x \rightarrow z)$) kürzeste Wege sind.

Änderung des Algorithmus

- Müssen tatsächlich alle Kanten (y,z) , die y verlassen getestet werden?



Für jedes Knotenpaar (x,y) halte Listen R_{xy} und L_{xy} :

$$R_{xy} := \{ (y,z) \text{ mit } P(x \rightarrow z) = P(x \rightarrow y)(y,z) \text{ ist ein kürzester Weg} \}$$
$$L_{xy} := \{ (z,x) \text{ mit } P(z \rightarrow y) = (z,x)P(x \rightarrow y) \text{ ist ein kürzester Weg} \}$$

- Diese Listen können gleichzeitig mit `ExtractMinQ()` aufgebaut werden
- Teste nur Kanten in R_{ay} und L_{xb} in Schritten ★

Locally Shortest Paths (LSP)

- Ein Weg $P(x \rightarrow y)$ heißt „Lokal Kürzester Weg“ („Locally Shortest Path“, LSP) in G , falls entweder
 - $P(x \rightarrow y)$ aus einem einzigen Knoten besteht, oder
 - jeder echte Teil-Weg von $P(x \rightarrow y)$ ein kürzester Pfad in G ist.
- Laufzeit des neuen Algorithmus:
 $O(|LSP| + n^2 \log n)$
- Wieviele LSP's kann es in G geben?

Locally Shortest Paths (LSP)

- Wenn die kürzesten Wege in G eindeutig sind, dann kann es in G höchstens mn LSP's geben.

Beweis: Fixiere Kante (x,v) und einen Knoten y . Es kann höchstens einen LSP geben, der mit (x,v) beginnt und bei y endet. Denn: jeder echte Teilweg ist kürzester Weg --- also auch $P(v \rightarrow y)$ --- und damit eindeutig.

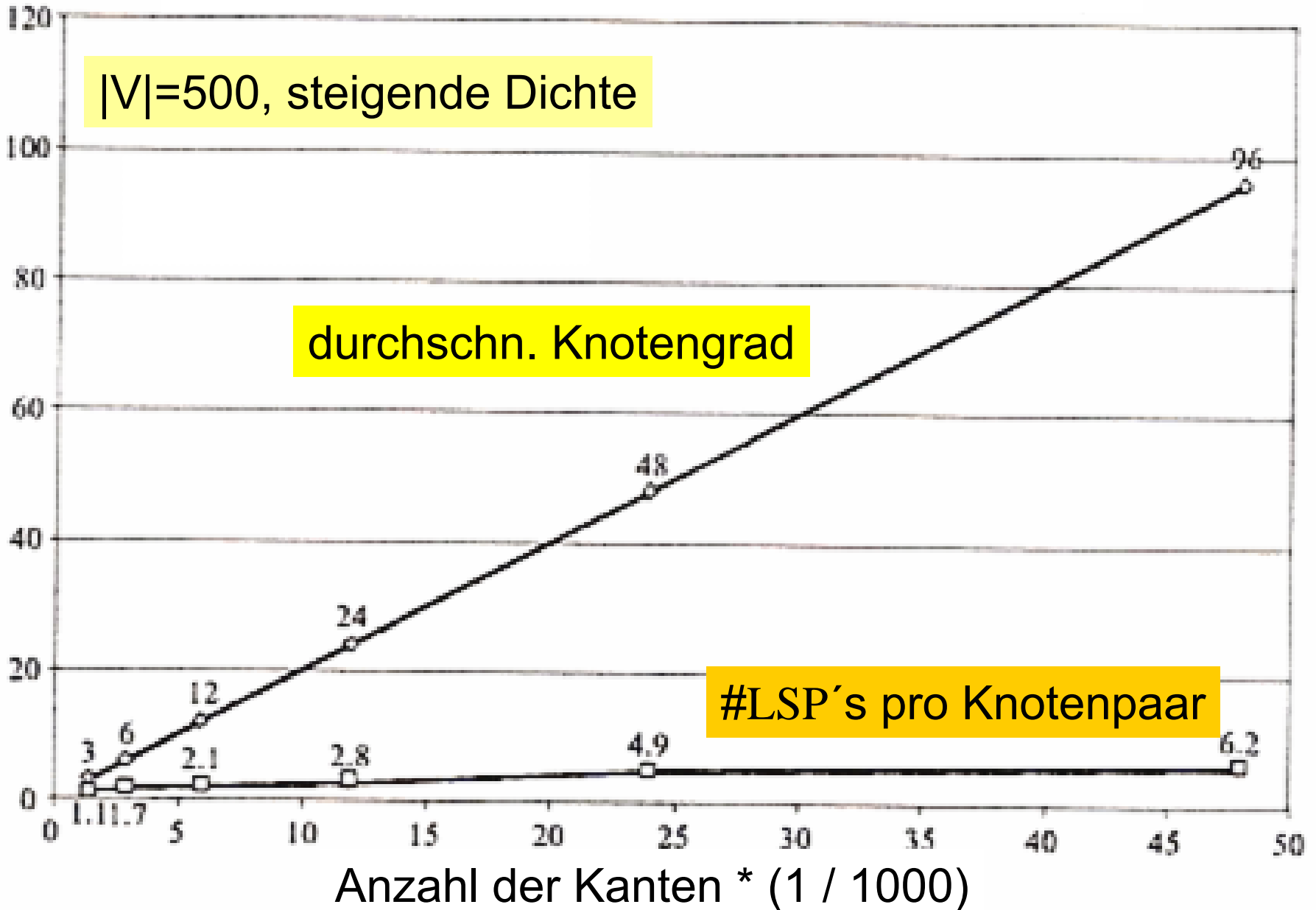
Es gibt m Möglichkeiten zur Wahl der ersten Kante und n Möglichkeiten zur Wahl des letzten Knotens.

→ Asymptotische Laufzeit nicht verbessert

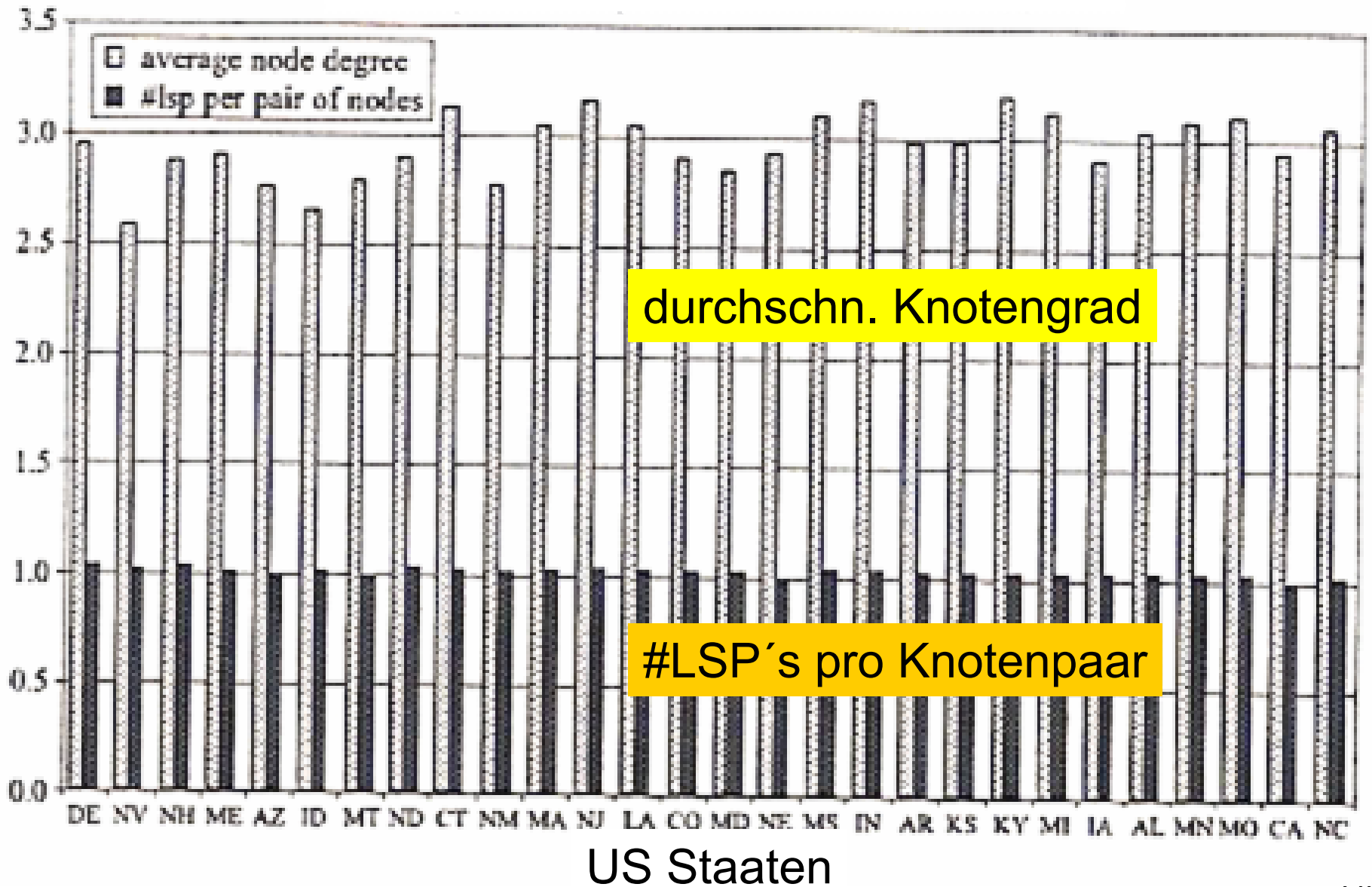
Experimentelle Untersuchungen

- Anzahl der LPS's in zufällig erzeugten Graphen und in „real-world“ Graphen (inkl. US „road networks“ sowie „Internet Autonomous Systems subgraphs“)

Anzahl der LSP's in zufällig erzeugten Graphen

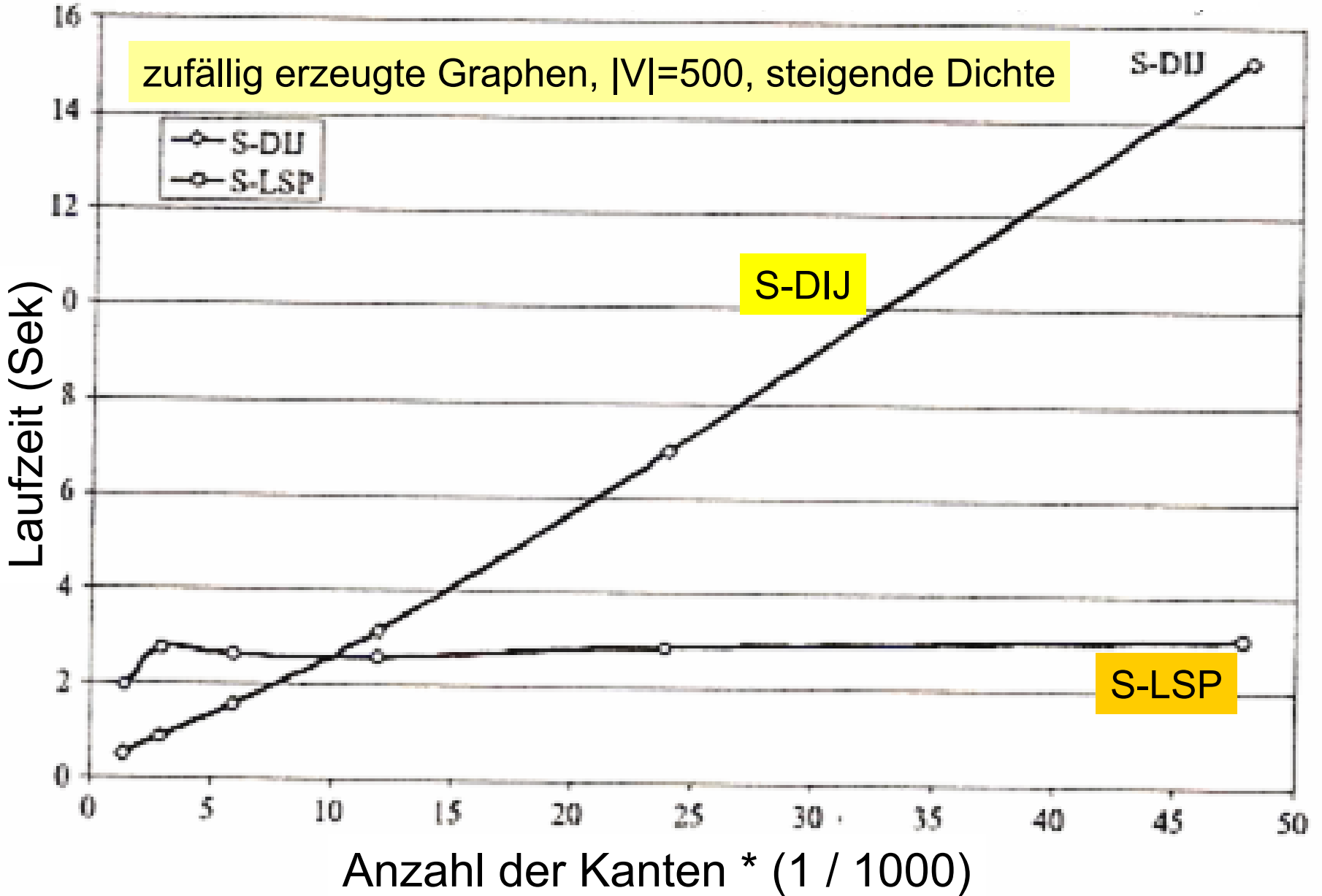


Anzahl der LSP's in US „road network“ Graphen



Laufzeit-Vergleich APSP Dijkstra mit/ohne LSP

zufällig erzeugte Graphen, $|V|=500$, steigende Dichte



Experimentelle Untersuchungen

- Anzahl der LPS's in zufällig erzeugten Graphen und in „real-world“ Graphen (inkl. US „road networks“ sowie „Internet Autonomous Systems subgraphs“)

- Ergebnisse: Anzahl der LSP's ist überraschend nahe bei n^2 , d.h. es gibt typischerweise nur einen LSP pro Knotenpaar.

- In zufällig erzeugten Graphen mit 20% Dichte kann S-LSP 16 Mal schneller sein als S-DIJ.

- Bei dünnen Graphen ist S-LSP langsamer.

Von Engineering zu Theorie

- Entdeckung: LPS's sind bei dynamischen Graphen hilfreich!
- → neuer Algorithmus mit asymptotisch bester Laufzeit: Nächste VO

- Implementierungen:

<http://www.dis.uniroma1.it/~demetres/experim/dsp/>

Vielen Dank!

