Kapitel 6

Ein kombinatorischer Min-Cut-Algorithmus

Effiziente Algorithmen, SS 2018

Professor Dr. Petra Mutzel
Dipl.-Inf. Denis Kurz

VO 10 am 17. Mai 2018

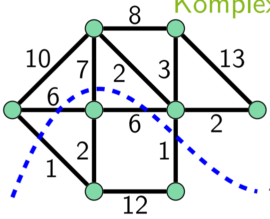
Minimales Schnittproblem (Min Cut)

Eingabe	gewichteter, ungerichteter Graph $G = (V, E)$ Kantengewichte $w(e) \in \mathbb{Q}^+$ (ungewichteter Graph $\widehat{=}$ alle Kantengewichte $= 1$)
Zulässige Lösung	nicht-leerer Schnitt (V_1, V_2): $V_1 \neq \emptyset$, $V_2 \neq \emptyset$, $V_1 \cap V_2 = \emptyset$ und $V_1 \cup V_2 = V$
Notation	Für $W \subseteq V$: $\delta(W) = \{\{u, v\} \in E \mid u \in W, v \notin W\}$
Bewertung	$w(V_1, V_2) = \sum_{e \in \delta(V_1)} w(e)$

Min Cut Ziel: finde nicht-leeren Schnitt mit minimalem Wert

Komplexität: in polynomieller Zeit lösbar

(Algorithmen: gleich)



$w(V_1, V_2) = 1 + 6 + 7 + 6 + 1 = 21$

Maximales Schnittproblem (Max Cut)

Eingabe	gewichteter, ungerichteter Graph $G = (V, E)$ Kantengewichte $w(e) \in \mathbb{Q}$ (ungewichteter Graph $\widehat{=}$ alle Kantengewichte $= 1$)
Zulässige Lösung	Schnitt (V_1, V_2): mit $V_1 \cap V_2 = \emptyset$ und $V_1 \cup V_2 = V$
Bewertung	$w(V_1, V_2) = \sum_{e \in \delta(V_1)} w(e)$

Max Cut

- Ziel: finde Schnitt mit maximalem Wert
- Komplexität: Zugehöriges Entscheidungsproblem ist NP-vollständig (auch für ungewichteten Fall) (Karp 1972)

Erster Lösungsansatz für Min Cut

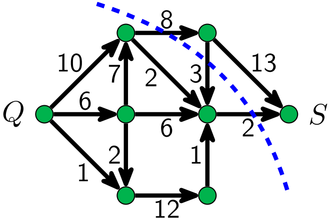
Annahme: G zusammenhängend $\Rightarrow$ sonst trivial: mit BFS

Erinnerung, Kapitel 4:

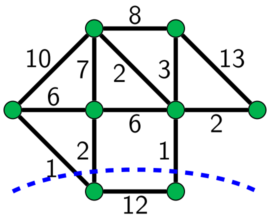
- Definition 4.5: Q-S-Schnitt: Knoten Q und S, $Q \in V_Q, S \in V_S, V_Q \cap V_S = \emptyset$ und $V_Q \cup V_S = V$
- Theorem 4.6: „Max Flow = Min Cut“
  Genauer: Der Wert eines maximalen Flusses in einem Netzwerk mit Quelle Q und Senke S ist gleich dem Wert eines minimalen Q-S-Schnittes

Lösungsidee: Max-Flow-Algorithmus für Minimales Schnittproblem

$w(V_Q, V_S) = 10$



Min Cut C, $w(C) = 4$



Erste Algorithmen für Min Cut

Lösungsidee: verwende Max-Flow-Algorithmus

- Für alle Paare Q und S von Knoten: löse maximales Q-S-Flussproblem und erhalte minimalen Q-S-Schnitt
- Vorher muss G in ein Netzwerk umgewandelt werden: jede Kante $e = \{u, v\}$, $e \cap \{Q, S\} = \emptyset$, mit Gewicht $w(e)$ wird in je drei gerichtete Kanten (u, v), (v, z) und (z, u) mit Kapazität je $w(e)$ umgewandelt
- mit Goldberg-Tarjan-Laufzeit $\mathcal{O}(n^3)$ für $n = |V|$
  $\Rightarrow$ Laufzeit: $\binom{n}{2}\mathcal{O}(n^3) = \mathcal{O}(n^5)$

Einfache Verbesserung:

- Fixiere $Q \in V$ und berechne für alle anderen Knoten $S \in V \setminus \{Q\}$ die minimalen Q-S-Schnitte
- Korrekt, da Q von mindestens einem anderen Knoten abgetrennt wird
- Laufzeit: $(n-1)\mathcal{O}(n^3) = \mathcal{O}(n^4)$

Algorithmus von Stoer und Wagner

Direkter Min-Cut-Algorithmus ohne Max-Flow-Berechnungen

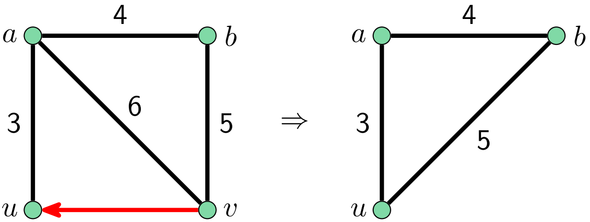
- Grundidee von Nagamochi und Ibaraki, 1989
- verbessert von Nagamochi, Ono und Ibaraki, 1994
- vereinfacht von Stoer und Wagner, 1994

Zentrale Methode: Kontraktion von 2 Knoten $u, v \in V, u \neq v$:

- Knoten v geht in Knoten u auf.
- Eine eventuelle Kante $\{u, v\}$ verschwindet.
- Alle Kanten $\{v, x\}$ werden durch neue Kanten $\{u, x\}$ ersetzt.
- Mehrfachkanten werden zu einer Kante verschmolzen.

Zentrale Methode Kontraktion(G, u, v) mit $u \neq v \in V$

1. $E := E \setminus \{\{u, v\}\}$
2. Für alle $\{v, x\} \in E$:
3.   Falls $\{u, x\} \in E$, dann
4. $w(\{u, x\}) := w(\{u, x\}) + w(\{v, x\})$
5.   sonst
6. $E := E \mathbin{\dot\cup} \{\{u, x\}\}$
7. $w(\{u, x\}) := w(\{v, x\})$
8. $E := E \setminus \{\{v, x\}\}$
9. $V := V \setminus \{v\}$
10. Return G



Algorithmus von Stoer und Wagner – Idee

Direkter Min-Cut-Algorithmus ohne Max-Flow-Berechnungen

Zentrale Methode: Kontraktion von 2 Knoten $u, v \in V, u \neq v$

Notationen:

- MinCut(G): minimaler Schnitt in G bzw. sein Wert
- min(u, v)-Cut: minimaler u-v-Schnitt bzw. sein Wert

Betrachte: $G_{u,v} :=$ Kontraktion(G, u, v),
Schnitt $C = (V_1, V_2)$ in $G_{u,v}$ mit $u \in V_1$ und Wert $w(C)$

Beobachtung: Schnitt $(V_1 \cup \{v\}, V_2)$ in G hat ebenfalls Wert $w(C)$

Folgerung: Alle Schnitte des Graphen $G_{u,v}$ entsprechen denjenigen Schnitten in G, die u und v nicht trennen

Folgerung: MinCut$(G) = \min\{$MinCut$(G_{u,v})$, min(u, v)-Cut$\}$

Algorithmus von Stoer und Wagner – Aufbau

Folgerung: $\text{MinCut}(G) = \min\{\text{MinCut}(G_{u,v}), \text{min}(u,v)\text{-Cut}\}$

Idee:

1. $c_{\min} := \infty$
2. Wähle 2 Knoten u und v und bestimme min(u, v)-Cut C
3. Falls $w(C) < c_{\min}$, dann $c_{\min} := w(C)$
4. Kontrahiere u und v in G
5. Falls $|V| > 1$: Gehe zu (2)

Problem: Ziel ist es, ohne Max-Flow-Berechnung auszukommen. Wie berechnen wir dann aber den min(u, v)-Cut in Schritt (2)?

Lösung: Wähle u und v so, dass min(u, v)-Cut nicht mit Max Flow berechnet werden muss

$\Rightarrow$ Legale Ordnung

Legale Ordnung

Definition 6.1

Eine Permutation $v_1, \ldots, v_n$ von V heißt *Legale Ordnung*, falls für alle $1 \leq i \leq n-1$ gilt: v_{i+1} ist ein Knoten mit größter Summe von Kantengewichten über Kanten zu der Menge $\{v_1, \ldots, v_i\}$.

Notation: $W_i := \{v_1, \ldots, v_i\}$ (also $W_0 = \emptyset$ und $W_n = V$)
$w_i(u) := \sum_{\{u,v\} \in \delta(W_i)} w(\{u, v\})$

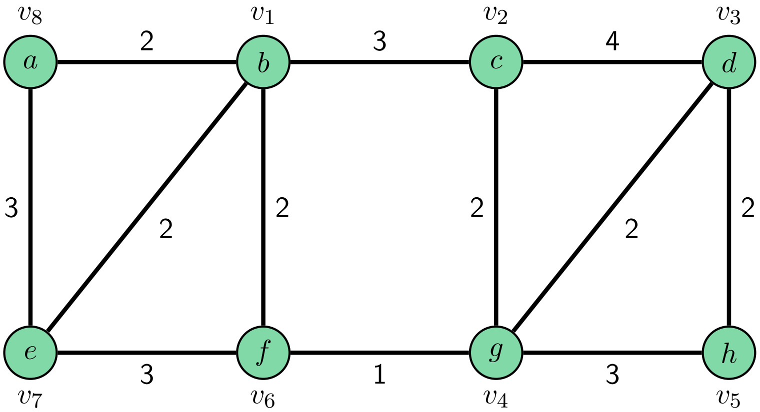
Legale Ordnung: Knoten v_{i+1} hat maximalen Wert $w_i(u)$ unter allen restlichen $u \in \{v_{i+1}, \ldots, v_n\}$.

Theorem 6.2

Sei $v_1, \ldots, v_n$ eine Legale Ordnung von G.
Dann ist $(W_{n-1}, \{v_n\})$ ein min(v_{n-1}, v_n)-Cut.

$\Rightarrow$ Schritt (2): Legale Ordnung bestimmen (Laufzeit später)

Legale Ordnung – Beispiel



Beweis des Theorems 6.2

Notation: Für einen Graph $G = (V, E)$ und $W \subseteq V$ sei $G[W]$ der Teilgraph von G, der durch die Knoten in W induziert wird
$\widehat{=}$ Knotenmenge W, Kantenmenge $\{e \in E \mid e \subseteq W\}$

Zulässige Annahme: G ist vollständig (macht Beweis einfacher)
$\Rightarrow$ nicht vorhandene Kanten Gewicht 0

Lemma 6.3

Sei $v_1, \ldots, v_n$ eine Legale Ordnung, $i \in \{1, \ldots, n-1\}$ und $u \in \{v_{i+1}, \ldots, v_n\}$.
Dann hat der min(v_i, u)-Cut in $G[W_i \cup\!\!\!\cdot\, \{u\}]$ einen Wert von mindestens $w_i(u)$.

Wenn Lemma gezeigt wurde, dann betrachte $i = n-1$, $u = v_n$:
min(v_{n-1}, v_n)-Cut $\geq w_{n-1}(v_n) = w(W_{n-1}, \{v_n\})$
$\Rightarrow$ Theorem 6.2 $\square$

Beweis von Lemma 6.3 durch Induktion

„min(v_i, u)-Cut in $G[W_i \cup \{u\}] \geq w_i(u) = \sum_{v \in W_i} w(\{v, u\})$“

Induktionsanfang: $i = 1$:
In $G[\{v_1, u\}]$ hat min(v_1, u)-Cut den Wert $w_1(u) = w(\{v_1, u\})$.

Induktionsvoraussetzung: Aussage gilt für $i - 1$:
(IV-u) In $G[W_{i-1} \cup \{u\}]$ gilt: min(v_{i-1}, u)-Cut $\geq w_{i-1}(u)$.
Insbesondere für $u = v_i$:
(IV-v_i) In $G[W_i]$ gilt: min(v_{i-1}, v_i)-Cut $\geq w_{i-1}(v_i)$.

Annahme für Widerspruch:
In $G[W_i \cup \{u\}]$ gilt: min(v_i, u)-Cut $< w_i(u)$.

Es gilt $w_{i-1}(u) = w_i(u) - w(\{v_i, u\})$.

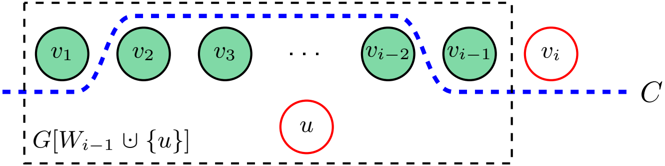
Betrachte $G' := G[W_i \cup \{u\}] \setminus \{\{v_i, u\}\}$; dann:

min(v_i, u)-Cut in G' = min(v_i, u)-Cut in $G[W_i \cup \{u\}] - w(\{v_i, u\})$
$\Rightarrow$ In G' gilt: min(v_i, u)-Cut $< w_i(u) - w(\{v_i, u\})$ (wg. Annahme)
$\Rightarrow$ In G' gilt: min(v_i, u)-Cut $< w_{i-1}(u)$.
Dieser Schnitt sei C.

Beweis von Lemma 6.3 durch Induktion, Fall 1

Induktionsvoraussetzung:
(IV-u) In $G[W_{i-1} \cupdot \{u\}]$ gilt: $\min(v_{i-1}, u)$-Cut $\geq w_{i-1}(u)$.
Betrachte $G' := G[W_i \cupdot \{u\}] \setminus \{\{v_i, u\}\}$
Annahme für Widerspruch:
In G' gilt: $\min(v_i, u)$-Cut $< w_{i-1}(u)$. Dieser Schnitt sei C.

1. Fall: C trennt v_{i-1} von u.
Dann definiert C auch einen Schnitt in $G[W_{i-1} \cupdot \{u\}]$, der v_{i-1} und u trennt.
(IV-u) $\Rightarrow$ Gewicht von $C \geq w_{i-1}(u)$.
Widerspruch!



Beweis von Lemma 6.3 durch Induktion, Fall 2

Induktionsvoraussetzung:

(IV-v_i) In $G[W_i]$ gilt: min(v_{i-1}, v_i)-Cut $\geq w_{i-1}(v_i)$.

Betrachte $G' := G[W_i \cup \{u\}] \setminus \{\{v_i, u\}\}$

Annahme für Widerspruch:

In G' gilt: min(v_i, u)-Cut $< w_{i-1}(u)$. Dieser Schnitt sei C.

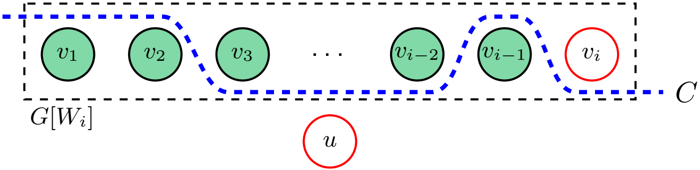
2. Fall: C trennt v_{i-1} nicht von u.

Dann definiert C auch einen Schnitt in $G[W_i]$, der v_{i-1} und v_i trennt.

(IV-v_i) $\Rightarrow$ Gewicht von $C \geq w_{i-1}(v_i)$.

Weil $v_1, \ldots, v_n$ Legale Ordnung ist, gilt $w_{i-1}(v_i) \geq w_{i-1}(u)$.

Widerspruch!



Algorithmus MINCUT$(G = (V, E))$

// Berechne Legale Ordnung:

1. $S := \emptyset$
2. Für $i = 1, \ldots, n$:
3. $\quad v_i := \arg\max\{w_{i-1}(u) \mid u \in V \setminus S\}$
4. $\quad S := S \cup \{v_i\}$

// Rekursionsverankerung:

5. Falls $n = 2$, Return Schnitt $(\{v_1\}, \{v_2\})$

// Rekursion:

6. $G_{v_{n-1},v_n} :=$ Kontraktion(G, v_{n-1}, v_n)
7. $C :=$ MINCUT(G_{v_{n-1},v_n})
8. Return minimalen Schnitt von C und $(W_{n-1}, \{v_n\})$

Laufzeitanalyse

Laufzeit: Rekursionstiefe $\mathcal{O}(|V|)$
pro Iteration dominiert Berechnung der Legalen Ordnung

Hierfür: Priority Queue Q: enthält alle Knoten $u \in V \setminus W_{i-1}$ mit key$(u) = w_{i-1}(u)$

Initialisierung: Setze $\forall u \in V$: key$(u) = 0$
ExtractMax: Erhalte v_i und füge es in S ein
IncreaseKey: Für alle Nachbarn $v \in V \setminus S$ von v_i:
key(v) := key$(v) + w(\{v, v_i\})$

Anzahl Operationen: $\mathcal{O}(|V|)$ ExtractMax, $\mathcal{O}(|E|)$ IncreaseKey

Laufzeit mit Fibonacci-Heap als Priority Queue:
Zeit pro Legale Ordnung: $\mathcal{O}(|V| \log |V| + |E|)$
Gesamtlaufzeit MINCUT: $\mathcal{O}\big(|V|^2 \log |V| + |V||E|\big)$

Zusammenfassung

Theorem 6.4

Der Algorithmus MINCUT berechnet für einen ungerichteten Graphen mit nicht-negativen Gewichten einen Min Cut in Zeit $\mathcal{O}\big(|V|^2 \log |V| + |V||E|\big) = \mathcal{O}\big(|V|^3\big)$.

Beispiel in animierten Folien oder in Originalpaper (moodle)

Historische Anmerkungen

- bis 1986: „naiver“Ansatz: $|V|-1$ Q-S-Min-Cut-Berechnungen (via Max Flow) mit anschließender Kontraktion von Q und S (siehe Folie 5 ff.); Laufzeit: $\mathcal{O}\big(|V|^4\big)$
- 1986: Gusfield stellt Vereinfachung des Gomory-Hu-Baumes (1961) vor: repräsentiert alle Q-S-Min Cuts für alle Knotenpaare Q, S; Laufzeit: $\mathcal{O}\big(|V|^4\big)$
- 1986: Padberg und Rinaldi stellen Preprocessing-Techniken vor, die Instanzgrößen durch *sichere Kontraktionen* zweier Knoten deutlich reduzieren können. Asymptotisch keine Verbesserung
- 1992: Hao und Orlin stellen eine Variante des Max-Flow-Algorithmus von Goldberg/Tarjan vor, die direkt einen Min Cut berechnet: wie naiver Ansatz, nutzt aber Labels der letzten Iteration Laufzeit: $\mathcal{O}\big(|V||E|\log(|V|^2/|E|)\big)$

Historische Anmerkungen

- 1989: Nagamochi und Ibaraki stellen die erste Algorithmen-Idee vor (schwer verständlich und komplex), die ohne Max-Flow-Berechnung auskommt
- 1994: Nagamochi, Ono, Ibaraki stellen eine Version mit zusätzlichen algorithmischen Verbesserungen vor; die letzte Kante in der legalen Ordnung wird kontrahiert und „kontrahierbare Wälder“werden eingeführt; Laufzeit: $\mathcal{O}\big(|V|^2 \log |V| + |V||E|\big)$
- *1994: Stoer und Wagner stellen vereinfachte Version vor, in der die letzten beiden Knoten kontrahiert werden; Laufzeit:* $\mathcal{O}\big(|V||E| + |V|^2 \log |V|\big)$
- 1993/1996: Karger und Stein stellen den ersten randomisierten Algorithmus für Min Cut vor, Laufzeit: $\mathcal{O}\big(|V|^2 \log |V|\big)$, aber randomisiert $\rightarrow$ Kapitel 10

Experimenteller Vergleich

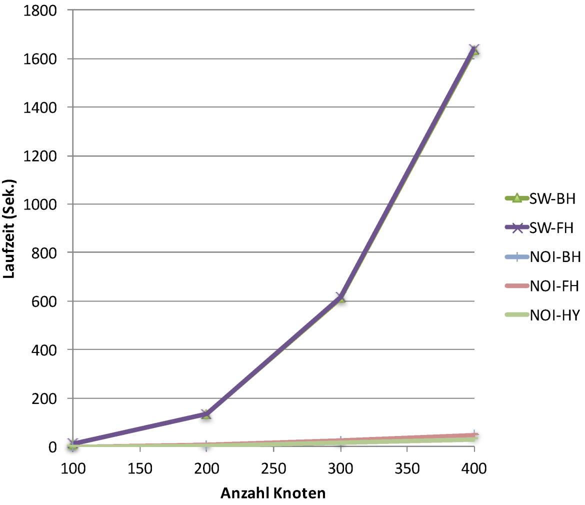
- NA: naiver Algorithmus (siehe Folie 5 ff.)
- GH: Gomory-Hu Baum (Gusfield)
- PR: Padberg-Rinaldi (Preprocessing Regeln + NA)
- NOI: Nagamochi-Ono-Ibaraki (ohne Max Flow)
- SW: Stoer-Wagner
- HO: Hao-Orlin Algorithmus (Variante von Goldberg/Tarjan)
- KS: Karger-Stein Algorithmus (randomisiert)

- Varianten: BH (Binärheap) bzw. FH (Fibonacci Heap)
- HY: Hybride Version mit PR

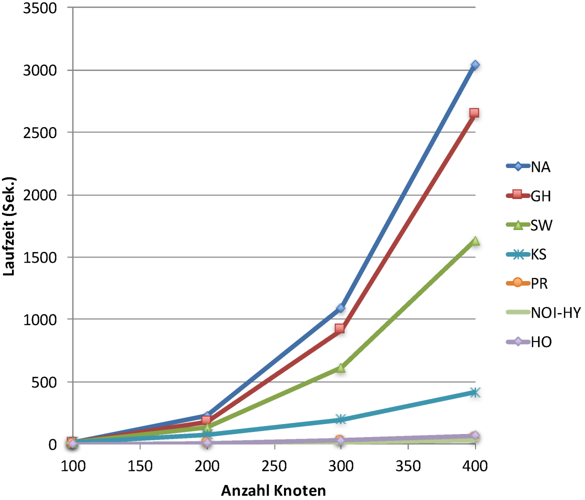
Setup der Experimente

- Experimentelles Paper von Jünger, Rinaldi, Thienel: Algorithmica 26, 2000
- Siemens Nixdorf SCENIC Pro MP6 PC, Linux 2.0.27, Pentium Pro (200 MHz), 256 kB cache memory
- Testinstanzen:
  - Randomisiert, Knotenzahl n, Dichte d, k „Cluster“, Kantengewichte: gleichverteilt in [0..100], zwischen Clustern skaliert mit $p \in [0..1]$
  - Traveling Salesman Problem (TSP) Instanzen (z.B. Städte, Leiterplatten); TSP $\rightarrow$ Kapitel 8
- 10 oder 100 Testläufe pro Datenpunkt
- nur ein Lauf von KS, dieser war auch meist optimal

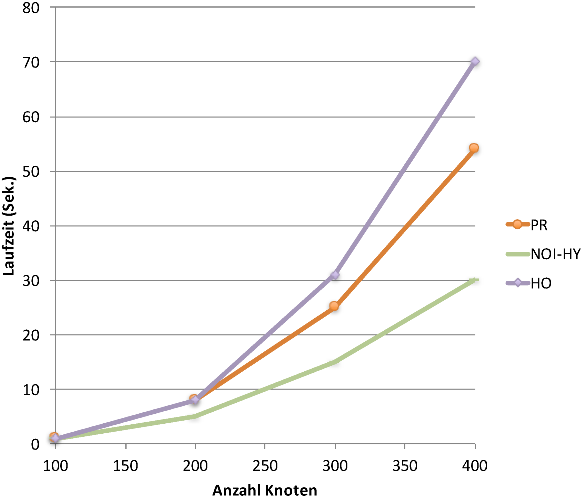
$100 \leq n \leq 400$, Dichte $= 50\%$, $p = 1/n$, BH vs. FH



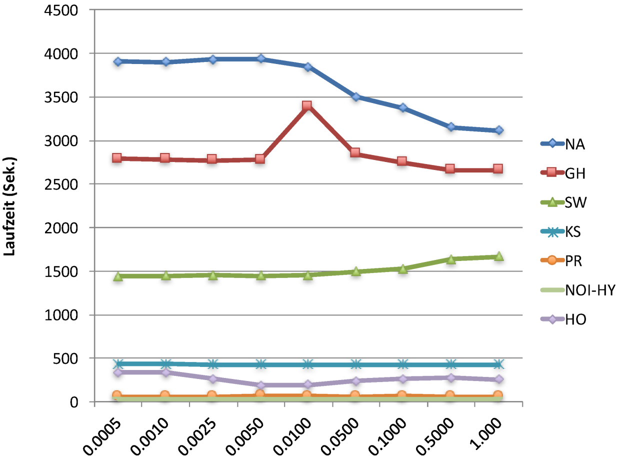
$100 \leq n \leq 400$, Dichte $= 50\%$, $p = 1/n$



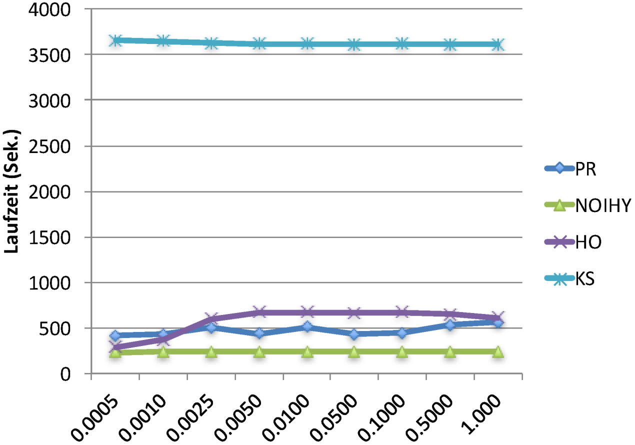
$100 \leq n \leq 400$, Dichte $= 50\%$, $p = 1/n$, Top-3



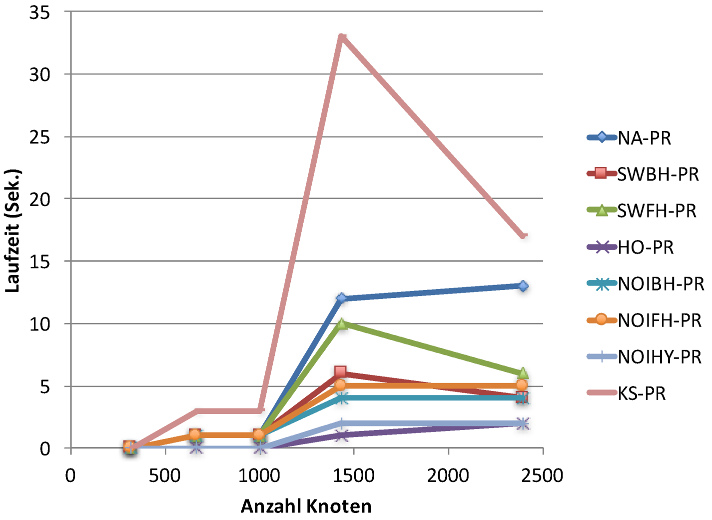
$n = 400$, Dichte $= 50\%$, $p = 0.0005, 0.001, 0.0025, \ldots, 1$



$n = 1000$, Dichte $= 50\%$, $0,0005 \leq p \leq 1$



TSP Instanzen mit PR Preprocessing



Literatur

- Mechthild Stoer und Frank Wagner: *A simple min-cut algorithm*. Journal of the ACM 44(4), pp. 585–591, 1997.
- Hiroshi Nagamochi und Toshihide Ibaraki: *Linear time algorithms for finding a sparse k-connected spanning subgraph of a k-connected graph*. Algorithmica 7, pp. 583–596, 1992.
- Hiroshi Nagamochi, Tadashi Ono und Toshihide Ibaraki: *Implementing an efficient minimum capacity cut algorithm*. Mathematical Programming 67, pp. 325–341, 1994.
- Jianxiu Hao und James B. Orlin: *A Faster Algorithm for Finding the Minimum Cut in a Graph*. Journal of Algorithms 17(3), pp. 424–446, 1994.
- Michael Jünger, Giovanni Rinaldi und Stefan Thienel: *Practical Performance of Efficient Minimum Cut Algorithms*. Algorithmica 26(1). pp. 172–195, 2000.