Methoden des Algorithmen Entwurfes

The Power of Recourse for Online MST and TSP

Felix Ortmann

Universität Hamburg

18. Juli 2017

Gliederung

Einleitung

„The Power of Recourse for Online MST and TSP“

von Nicole Megow, Martin Skutella, José Verschae und Andreas Wiese

Siam Journal on Computing

[Megow et al., 2016]

Definitionen

2. Definitionen
   - MST und TSP
   - Online Problemstellung
   - Einfacher Greedy Algorithmus – online MST
   - Budget

MST und TSP

Graf $G = (V, E)$; G ist vollständig

Minimal Spanning Tree (MST)

- Verbinde alle Knoten
- Azyklisch

Gesucht: kleinste (metrische) Verbindung aller Knoten

Traveling Salesman Problem (TSP)

- Besuche jeden Knoten exakt einmal
- Zyklisch

Gesucht: kürzeste (metrische) Route

MST und TSP

Minimal Spanning Tree (MST)

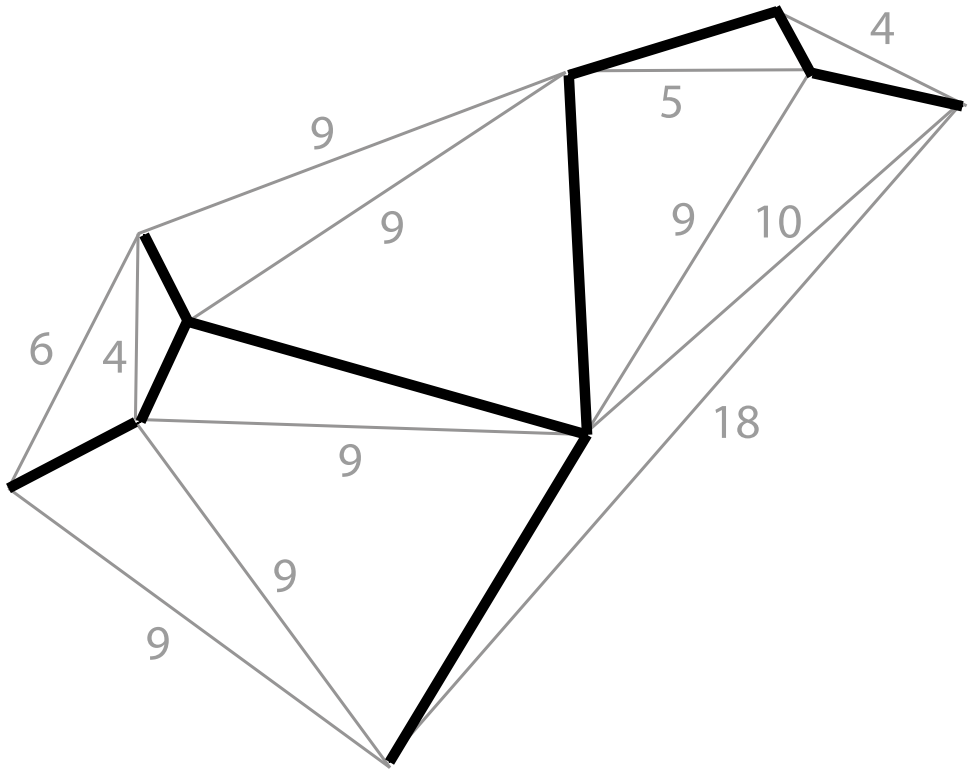

Abbildung: Minimal Spanning Tree[1]

[1] https://commons.wikimedia.org/wiki/File:Minimum_spanning_tree.svg

MST und TSP

Traveling Salesman Problem (TSP)

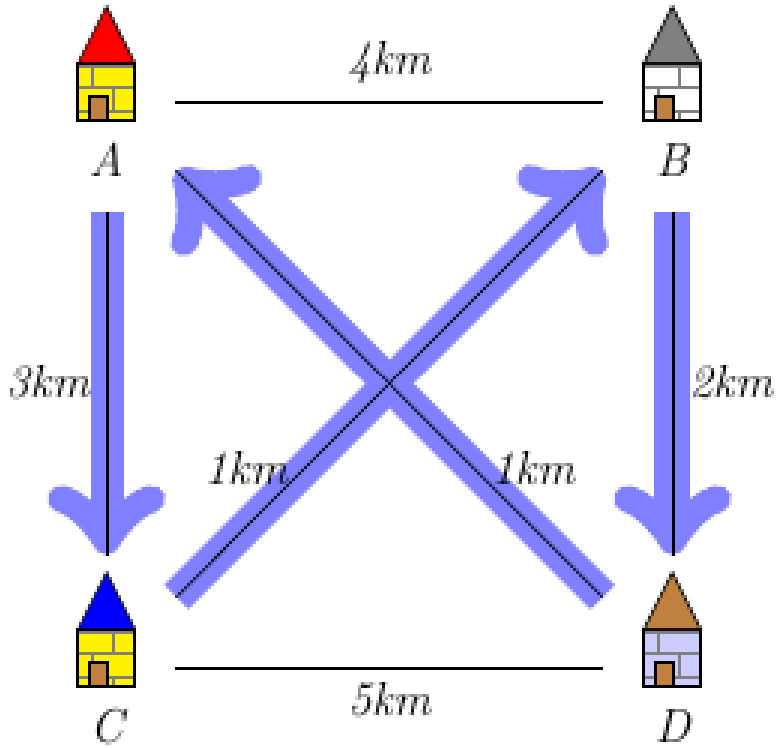

Abbildung: TSP Tour[2]

[2] https://commons.wikimedia.org/wiki/File:Tsp_opt.png

Online Problemstellung

„Nach und nach Knoten und zugehörige Kanten aufdecken. Problemstellung (MST bzw. TSP) in jedem Zug auf Teilgraf möglichst gut bearbeiten.“

- $\sigma = v_0, v_1, \ldots$ wobei v_t: *Vertex*
- $\forall s \leq t-1 : v_t v_s$ Kanten werden zusammen mit Knoten t bekannt
- Kantengewichte erfüllen Dreiecksungleichung ($\rightarrow$ metrisch)
- Iteration t:
  - Teilgraf $G_t = (V_t, E_t)$
  - $V_t = \{v_0, \ldots, v_t\}$
  - $E_t = V_t \times V_t \rightarrow$ Graf ist „complete“

Online Ziele

Online MST

- Kanten Teilmenge T_t
- T_t ist MST zu G_t

Online TSP

- Tour durch G_t: Q_t
- Q_t ist TSP Tour auf G_t ($\approx$ minimiert traveling Kosten)

Einfacher Greedy Algorithmus – online MST

Vorherige Iteration: T_{t-1}

Iteration t:

- Decke neuen Knoten auf: v_t
- Kürzeste Verbindungskante g_t zwischen v_t und T_{t-1}
- $T_t = T_{t-1} \cup g_t$
- Kostenreduktion von T_t – „*Swaps*”:
  - Füge günstige Kante hinzu ($\rightarrow$ Kreis entsteht in Teilbaum)
  - Entferne teuerste Kante aus Kreis
  - Wiederhole ...

Swaps und Recourse

- Swaps $\rightsquigarrow$ optimale Lösung
- Wie viele Swaps...?

Recourse

Recourse bzw. *recourse Budget* – Menge von Kanten, die pro Iteration maximal zur Lösung hinzugefügt werden.

Budget

Iteration t, Budget k

Striktes Budget

$$\forall t \geq 1 : \quad |T_t \setminus T_{t-1}| \leq k$$

Amortisiertes Budget

$$\sum_{s=1}^{t} \quad |T_s \setminus T_{s-1}| \quad \leq k \cdot t$$

Online MST mit amortisiertem Budget

3. Online MST mit amortisiertem Budget
   - Zielsetzung
   - Freezing Rules
   - Algorithmus Sequence Freeze
   - Competitive Analysis
   - Rückblick

Zielsetzung

Ziel

Online Algorithmus für MST mit folgenden Eigenschaften ($\epsilon > 0$): (Theorem 2)

- $(1 + \epsilon)$-competitive
- Amortisiertes Budget $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$

Abschluss

Präsentierter Algorithmus liefert best mögliches Ergebnis, logarithmisch gemessen (Theorem 1)

Online MST mit amortisiertem Budget

Theorem 1 (Lower Bound)

Jeder $(1+\epsilon)$-competitive Algorithmus für das online MST Problem benötigt ein amortisiertes recourse Budget von $\Omega(\frac{1}{\epsilon})$

Beweis siehe [Megow et al., 2016]

Theorem 2 (Upper Bound)

Es existiert ein $(1+\epsilon)$-competitiver Algorithmus für das online MST Problem mit einem amortisierten recourse Budget von $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$

Freezing Rules

Iteration t:

- OPT_t^{max}: bestes (von unserem Algorithmus) erreichbares Ergebnis
- $\ell(t)$: Iteration mit den meisten Swaps (vor dieser Iteration)
- $\ell(t) \leq (t-1)$. $OPT_{\ell(t)}^{max} \leq \epsilon \cdot OPT_t^{max}$
- Greedy Kante g_s^0: In Iteration s greedy hinzugefügt
- Swap: g_s^1 sei Kante, die g_s^0 ersetzt
- Wurde in später ggf. geswapped, momentane Ersetzung: $g_s^{i(s)}$

Freezing Rules

Rule 1

Freeze Sequenz $(g_s^0, ..., g_s^{i(s)})$ gwd. $s \leq \ell(t)$

Rule 2

Freeze Kante wenn der Kostengewinn kleiner als $\epsilon OPT_t^{max}/(t - \ell(t))$ ist.

Freezing Rules In Worten

Finde Balance zwischen Anzahl Swaps ($\leq k$) und Nutzen

Rule 1

Verhindert das Swappen von Kanten, die zu einem Subgrafen gehören, der an den Gesamtkosten des MST keinen großen Einfluss hat.
$\Longrightarrow$ Wurde vor / während der größten Iteration $\ell(t)$ geswapped. „Den besten bisherigen Subgrafen (und alles was schon davor gefroren war) fassen wir nicht mehr an.“

Rule 2

Verhindert das Entfernen von Kanten deren Kosten kleiner als das Mittel seit der größten Iteration sind.
$\Longrightarrow$ „Kein Swap, wenn direkter Nutzen zu gering”.

Algorithmus Sequence Freeze

Wie simple Greedy Algorithmus mit Einschränkungen

$T_0 = \emptyset$

Iteration t: $T_t = T_{t-1} \cup \{\text{„günstigste neue Kante"}\}$

Betrachte Kanten $(f, h) \in (E_t \setminus T_t) \times T_t$
Swap wenn nur, wenn..:

- $(T_t \cup \{f\}) \setminus \{h\}$ bleibt ein Tree

1. $c(h) > (1 + \epsilon) \cdot c(f)$
2. $h = g_s^{i(s)}$ für ein $s \geq \ell(t) + 1$ (nicht gefroren durch *Rule 1*)
3. $c(h) > \epsilon \frac{OPT_t^{max}}{t - \ell(t)}$ (nicht gefroren durch *Rule 2*)

Competitive Analysis (Skizze)

Competitiveness des Algoithmus im Vergleich zu OPT

$(1+\epsilon)$-competitive

- *Cond. 1* & *Cond. 3* – Kosten steigen maximal um $(1+3\epsilon)$ pro Iteration $\rightarrow (1+O(\epsilon))$-competitive
- *Cond. 2* – Durch Weglassen „kostengünstiger" Swaps wird die Gesamtlösung im Vergleich zu OPT_t maximal $O(\epsilon OPT_t)$ schlechter

- Partitioniere Graf $T = T_{new} \cup T_{old}$ – vor / nach längster Iteration
- $T_{old} := \{g_1^{i(1)}, \ldots, g_{\ell(t)}^{i(\ell(t))}\}$
- $T_{new} := \{g_{\ell(t)+1}^{i(\ell(t)+1)}, \ldots, g_t^{i(t)}\}$

Beweise Bounds für Partitionen

Competitive Analysis (Skizze)

Amortisierter Upper Bound für Budget

- $k_q := |T_q \setminus T_{q-1}|$: benutztes Budget in Iteration q
- $D_\epsilon \in O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$

Zeige $\sum_{q=1}^{t} k_q \leq D_\epsilon \cdot t$

1. Upper Bound auf Swap Sequenz Länge
   Cond. 1 – Nur Swap wenn Kostengewinn $> (1+\epsilon)$:
   Maximale Swapanzahl $\implies \log_{1+\epsilon} c(g_s^0) - \log_{1+\epsilon} c(g_s^{i(s)-1}) + 1$
2. Lower Bound auf Swap Kosten
   Cond. 3 – Nur Swap wenn Kosten d. zu entfernenden Kante Threshold übersteigt

Online MST mit amortisiertem Budget

Ergebnis

Online Algorithmus für MST mit folgenden bewiesenen Eigenschaften:

- $(1+\epsilon)$-competitive
- Amortisiertes Budget $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$

Da jeder online Algorithmus für MST mit obigen Eigenschaften mindestens ein Budget von $\Omega(\frac{1}{\epsilon})$ benötigt (vgl. Theorem 1), liefert *Sequence Freeze* das (logarithmisch) bestmögliche online Ergebnis.

Online MST mit striktem Budget

4. Online MST mit striktem Budget

Online MST mit striktem Budget

Kann ein striktes Budget besser sein? Insbesondere besser als 2-competitive?

Gegenbeispiel

- Vollst. Graph, Knoten $v_0, ..., v_n$
- $\forall t < n : c(v_t, v_n) = 1$
- $\forall s, t < n : c(v_s, v_t) = 2$

$\Longrightarrow$ MST ist ein Stern um den letzten Knoten n

Alle Iterationen $t < n$: T_t enthält nur Kanten mit Kosten 2

Bei fixem Budget k: T_n enthält exakt $n - k$ Kanten der Kosten 2

$\Longrightarrow$ Competitive Ratio $2 - k/n$ $\quad (> 2 - \epsilon)$ bei großen n

Anwendung auf TSP

5. Anwendung auf TSP
   - Shortcutting (Beispiel)
   - Motivation: Unvorhersehbare Touren
   - Robuste TSP Touren
   - Algorithmus: Robust Tour Shortcut
   - Algorithmus: Robust Walk Update
   - Competitive Analysis

Shortcutting (Beispiel)

Eulerian Walk

- Besuche jede *Kante* genau einmal
- Sei $(2 \cdot T)$ Multigraf zu T – jede Kante verdoppelt
  $\Longrightarrow$ Jeder Knoten hat geraden Grad
  $\Longrightarrow$ Eulerian Walk möglich

- Bilde Eulerian Walk
- Notiere *Knoten*reihenfolge
- Streiche alle mehrfach Vorkommnisse (*Shortcut*)
  $\Longrightarrow$ Traveling Salesman Tour

Shortcutting (Beispiel)

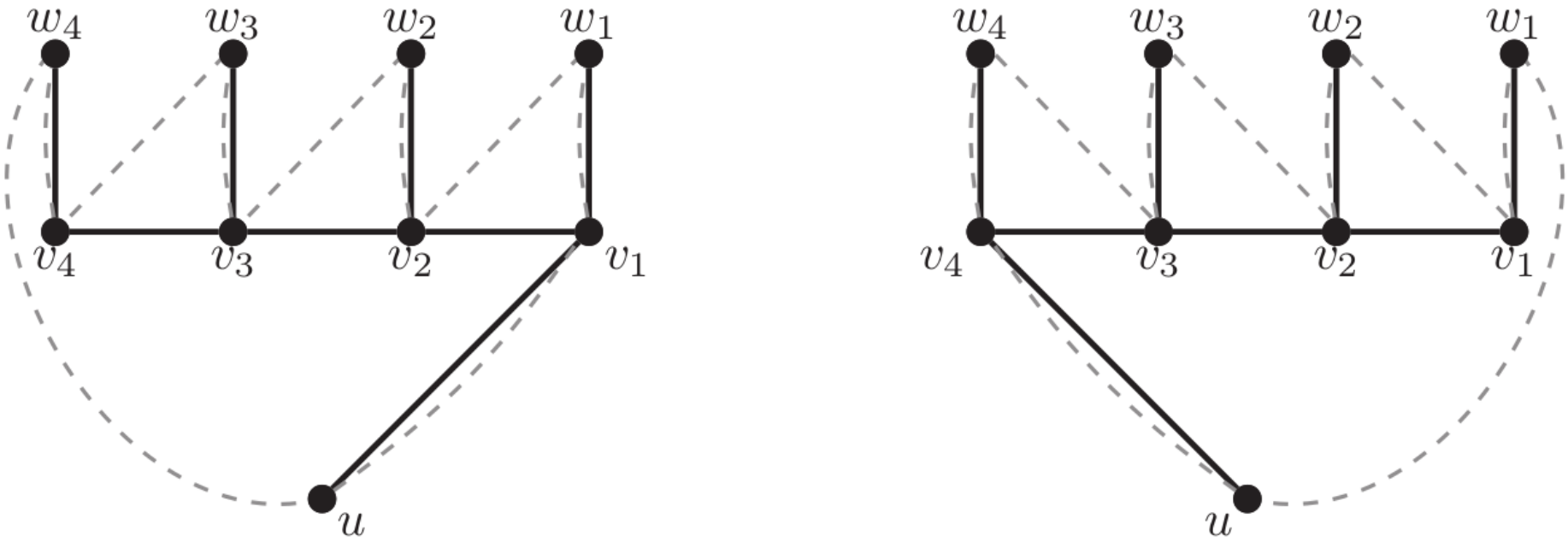

Abbildung: Tree (schwarz), Shortcut Tour (gestrichelt) [Megow et al., 2016]

$W = u, v_1, w_1, \bar{v}_1, v_2, w_2, \bar{v}_2, v_3, w_3, \bar{v}_3, v_4, w_4, \bar{v}_4, \bar{\bar{v}}_3, \bar{\bar{v}}_2, \bar{\bar{v}}_1, \bar{u}$

$\text{Tour} = u, v_1, w_1, v_2, w_2, v_3, w_3, v_4, w_4, (\bar{u})$

Motivation: Unvorhersehbare Touren

- Spanning Trees R, R'
- $R' = (R \cup \{f\}) \setminus \{g\}, f \notin R \wedge g \in R$
  $\Longrightarrow$ „ein Swap Unterschied"

- Sehr ähnliche Bäume führen zu ggf. sehr unterschiedlichen TSP Touren
- Tour zu R' (rechts): hat nie Knotenfolgen der Form (w_i, v_{i+1})
  $\Longrightarrow$ Bei steigender Knotenanzahl zunehmend großer Unterschied in Tour

Robuste TSP Touren

Theorem 3 (MST zu TSP)

Gegeben eine Reihe von metrischen Grafen $G_0, ..., G_t, ..$ mit $G_t = (V_t, E_t) \wedge V_t = \{v_0, ..., v_t\}$. T_t is Spanning Tree zu G_t in Iteration t.

Es existiert ein online Algorithmus, der zu jedem Teilgraf eine TSP Tour ausgibt $(Q_0, ..., Q_t, ...)$, sodass

- $c(Q_t) \leq 2 \cdot c(T_t)$
- $|Q_t \setminus Q_{t-1}| \leq 4 \cdot |T_t \setminus T_{t-1}|$

Beweis benötigt verbessertes Shortcutting

Robuste TSP Touren

Idee: „Unterschiede zwischen Iterationen $(t-1)$ und t messbar machen“

Momentan ist bereits messbar, wie zwei Spanning Trees (R, R') sich nach einem Swap unterscheiden ($\rightarrow$ eine Kante)
(mehrere Swaps deterministisch)

Vorraussetzungen für Beweis von Theorem 3

- Deterministische Prozedur um eine TSP Tour Q von einem Eulerian Walk W und einem Spanning Tree R zu erzeugen $\rightarrow$ Robustes *Shortcutting*
- Deterministische Prozedur um aus zwei Spanning Trees (R, R'), die sich in einer Kante unterscheiden, sowie einem Eulerian Walk W auf $(2 \cdot R)$ einen Eulerian Walk W' auf $(2 \cdot R')$ zu erzeugen

Algorithmus: Robust Tour Shortcut

Input:

Baum R, Eurlerian Walk $W = x_1, ..., x_r$ auf $2 \cdot R$, Funktion $I : \{x_1, ..., x_r\} \to \{0, 1\}$ (Selektionsfunktion um Nodes nur einmal zu besuchen (vgl. $2 \cdot R$))

- Erzeuge Walk der Form: $x_{\ell_1}, x_{\ell_2}, ..., x_{\ell_{|V|}}$, sodass $\forall i < j : \ell_i < \ell_j \wedge \forall i : I(x_{\ell_i}) = 1$
- Rückgabe: $Q := \{x_{\ell_1}, ..., x_{\ell_{|V|}}, x_{\ell_1}\}$

Beobachtung:

Die zurückgegebene Tour Q hat höchstens Kosten von $2 \cdot c(R)$ (unabhängig von I)

Algorithmus: Robust Walk Update (Beispiel)

Ziel:

Update $W \to W'$ bei Update von Baum $R \to R'$

- $R' = (R \cup \{f\}) \setminus \{g\}$ wobei $f = st \wedge g = vw$
- $R \setminus \{g\} \quad \Longrightarrow$ 2 verbundene Komponenten (C_1, C_2)
- o.B.d.A: $x_1, v, s \in C_1 \wedge t, w \in C_2$
- $W = W_1, v, w, W_2, \bar{w}, \bar{v}, W_1'$

Start in C_1 (Teil W_1), via vw zu C_2 (dort vollständig ablaufen, Teil W_2), via $\bar{w}\bar{v}$ zurück zu C_1, dort restliche Knoten ablaufen (Teil W_1'). o.B.d.A $s \in C_1 \wedge t \in C_2$.

Algorithmus: Robust Walk Update (Beispiel)

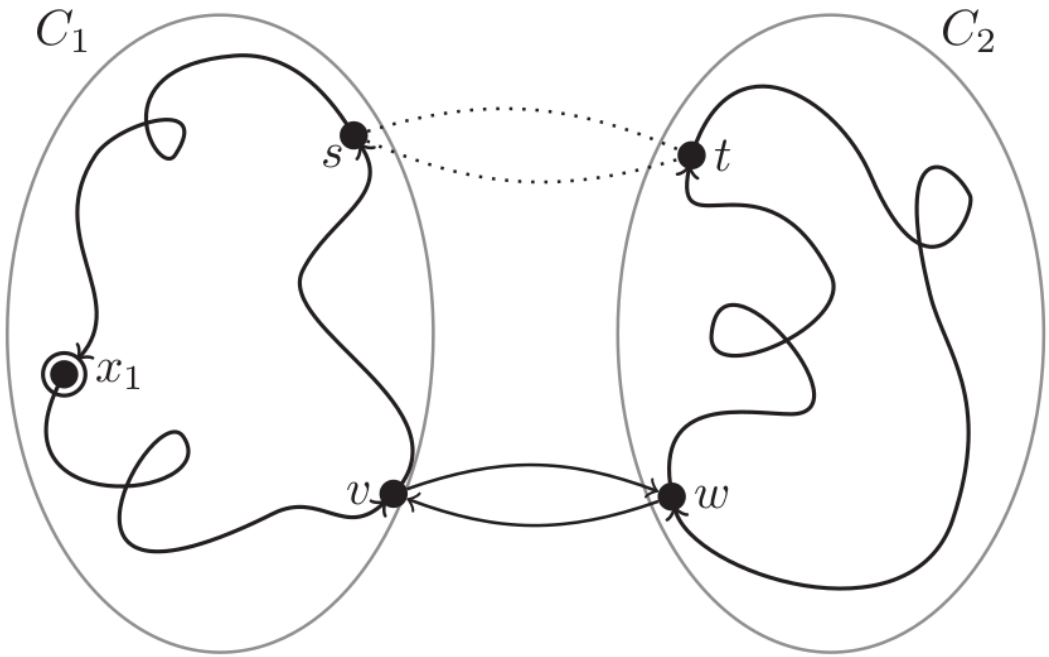

Abbildung: Dekomposition eines Eulerian Walks W [Megow et al., 2016]

$$W = W_1, v, w, W_2, \bar{w}, \bar{v}, W_1'$$

$$W = W(x_1, v), v, W(w, t), W(t, w), \bar{w}, W(v, s), W(s, x_1)$$

$$W' = W(x_1, v), W(v, s), \bar{s}, W(t, w), W(w, t), \bar{t}, W(s, x_1)$$

Algorithmus: Robust Walk Update

Input:

Baum R (auf Graf $G = (V, E)$), Eurlerian Walk $W = x_1, ..., x_r$ auf $2 \cdot R$, Funktion $I : \{x_1, ..., x_r\} \to \{0, 1\}$ (Selektionsfunktion um Nodes nur einmal zu besuchen), ein Baum $R' = (R \cup \{f\}) \setminus \{g\}$ wobei $f = st \notin R \wedge g = vw \in R$

- Zerlege W in Teil-Walks:
  $W = W(x_1, v), v, W(w, t), W(t, w), \bar{w}, W(v, s), W(s, x_1)$
  (Falls nicht möglich, drehe Walk um)
- Rückgabe: $W' = W(x_1, v), W(v, s), \bar{s}, W(t, w), W(w, t), \bar{t}, W(s, x_1)$
- „Patch“ Selektionsfunktion I: $I(\bar{t}) = I(\bar{s}) = 0$. Falls $I(v) = 1$ setze $I(x) = 1$ (x ist erster Knoten in $W(v, s)$). Gleiches für $I(w)$ bzw. $W(w, t)$.

Competitive Analysis (Skizze)

Zusammenreihung der Algorithmen

- Erzeuge Spanning Tree R zu Graf G mit Algorithmus *Sequence Freeze*
- Erzeuge Eulerian Walk W auf Multigraf $2 \cdot R$
- Erzeuge Tour Q durch G mit Algorithmus *Robus Tour Shortcut*
- Modifiziere (M)ST $\rightarrow R'$
- *Update* Eulerian Walk $W \rightarrow W'$ mit Algorithmus *Robust Walk Update*
- Nutze erneut *Robust Tour Shortcut*: $W' \xrightarrow{neue TSP Tour} Q'$

Beobachtung: Die Eulerian Walks sind sehr ähnlich, bis auf wenige Kanten $\rightarrow$ die TSP Touren Q und Q' sind sehr ähnlich.

Competitive Analysis (Skizze)

Zeige: $|Q' \setminus Q| \leq 4$

Fallunterscheidungen

Argumentation über die Dekomposition des Walks W und das Update der Selektionsfunktion I. Betrachte 4 Fälle in Walk W':

- $W(x_1, v) \to W(v, s)$
- $W(v, s) \to W(t, w)$
- $W(t, w) \to W(w, t)$
- $W(w, t) \to W(s, x_1)$

Zusammenfassung

Zusammenfassung

- Standard online MST: Beste bekannte competitive Ratio $\Theta \log n$
- Neue Erkenntnis mit [Megow et al., 2016]:
  Recourse verbessert die competitive Ratio zu $\underline{(1+\epsilon)}$ (amortisiert)
- Verändertes Shortcutting Verfahren
- Online MST $\xrightarrow{\text{det. Verfahren}}$ online TSP
  Bewiesene Upper Bounds:
  - Erhöht competitive Ratio um 2
  - Erhöht Budget um 4
- Online TSP $(2+\epsilon)$-competitive mit amortisiertem Budget $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$

Literatur


Megow, N., Skutella, M., Verschae, J., and Wiese, A. (2016).
The Power of Recourse for Online MST and TSP.
SIAM Journal on Computing, 45(3):859–880.