Zig-Zag Product
USTCON $\in$ LOGSPACE

Georg Osang

1 Zig-Zag Product

Definition 1. *Ein (n, d, λ)-Graph ist ein d-regulärer Graph mit n Knoten, dessen normalisierte Adjazenzmatrix A einen zweitgrößten Eigenwert $\lambda(G) \leq \lambda$ hat. (Normalisiert heißt, alle Einträge durch d geteilt.)*

Notation: Schreibe kurz $[d]$ für $\{n \in \mathbb{N} | n \leq d\}$ für $d \in \mathbb{N}$.

Wenn aus dem Kontext eindeutig, ist n die Knotenzahl, d der Grad, λ der normalisierte zweite Eigenwert und λ' der zweite Eigenwert eines Graphen.

Eine Knotenmenge der Größe n kann als die Menge $[n]$ angenommen werden.

ZHK steht für Zusammenhangskomponente.

Definition 2. *Sei in einem d-regulären Graphen mit n Knoten bei jedem Knoten für die adjazenten Kanten eine Nummerierung von 1 bis d gegeben. Dann ist die Rotationsabbildung (Rotation Map) wie folgt definiert:*

$$\begin{aligned} Rot_G : [n] \times [d] &\to [n] \times [d] \\ Rot_G(v, i) &\mapsto (w, j) \end{aligned}$$

wenn die i-te Kante von v nach w führt und von w aus gesehen die j-te Kante ist.

Es gilt offensichtlich: $Rot_G^2 = id_{[n]\times[d]}$, also $Rot_G(Rot_G(v, i)) = (v, i)$.

Definition 3. *(Graph Powering) Zu einem (n, d, λ)-Graph G ist G^t der Graph, der die gleiche Knotenmenge, und für jeden Pfad der Länge t eine Kante besitzt. Hat G die Adjazenzmatrix A, so hat G^t die Adjazenzmatrix A^t. G^t ist ein (n, d^t, λ^t)-Graph. (Achtung: Schleifen erhöhen Diagonaleinträge nur um 1.)*

$$Rot_{G^t}(v_0, (a_1, ..., a_t)) = (v_t, (b_t, ..., b_1))$$

wobei iterativ $(v_i, b_i) = Rot_G(v_{i-1}, a_i)$, $v_i \in [n]$, $a_i, b_i \in [d]$.

Beweis. Es gibt für jeden Knoten d^t verschiedene Pfade der Länge t, die in dem Knoten starten.

Da A symmetrisch, ist A diagonalisierbar, also sind die Eigenwerte von A^t die t-ten Potenzen der Eigenwerte von A. □

Bemerkung 4. *Besteht ein Graph aus mehreren, nicht bipartiten ZHK, so bleiben diese unter Graph-Powering bestehen.*

Beweis. Zwei ZHK werden nicht verbunden: klar, denn es gibt keinen Pfad zwischen den ZHK.
Eine ZHK bleibt zusammenhängend, denn wegen Nicht-Bipartitheit ist $-\lambda_n < \lambda_1$ und folglich nach Potenzieren (EW wieder neu nach Größe sortieren) $\lambda_1^{(t)} > \lambda_2^{(t)}$. □

Bemerkung 5. *Ein zusammenhängender Graph mit Schleifen ist nie bipartit.*

Definition 6. *(Replacement Product) Sei G ein d-regulärer Graph mit n Knoten und H ein e-regulärer Graph mit d nummerierten Knoten. An jedem Knoten von G sei eine Nummerierung der Kanten gegeben. Ersetze jeden Knoten v in G durch H_v, eine Instanz von H. Für die i-te Kante aus v heraus nach w in G setze eine Kante von dem i-ten Knoten von H_v ausgehend nach H_w. Das ist das Replacement Product von G und H, notiert mit $G\textcircled{r}H$. Der Graph hat Knotenmenge $[n] \times [d]$ und ist $d+1$-regulär.*

Definition 7. *(Zig-Zag Product) Gleiche Knotenmenge wie im Replacement Product. Für jeden Pfad im Replacement Product, der aus einer Kante in einer Instanz von H, dann einer Kante in G, und dann wieder einer Kante in einer Instanz von H besteht, wird eine Kante eingefügt. Der resultierende Graph wird mit $G\textcircled{z}H$ notiert und ist d^2-regulär.*

$$Rot_{G\textcircled{z}H}((v,a),(i,j)) = ((w,b),(i',j'))$$

wobei

$$\begin{aligned}(a',i') &= Rot_H(a,i)\\ (w,b') &= Rot_G(v,a)\\ (b,j') &= Rot_H(b',j)\end{aligned}$$

Bemerkung 8. *Besteht G aus mehreren ZHK S mit je mindestens einer Schleife, so sind in $G\textcircled{z}H$ die Teilgraphen $S \times H$ wieder ZHK mit je mindestens einer Schleife.*

Beweis. ZHK werden nicht vereinigt und Schleifen bleiben: Wird am Beispiel ersichtlich.

ZHK fallen nicht auseinander: Folgt gleich über zweiten Eigenwert. □

Theorem 9. *Sei G ein (n,m,α)-Graph, H ein (m,d,β)-Graph, dann ist $G\textcircled{z}H$ ein $(nm, d^2, \varphi(\alpha,\beta))$-Graph, wobei φ folgendes erfüllt:*

$$\text{Ist } \alpha,\beta < 1, \text{ dann ist } \varphi(\alpha,\beta) < 1 \tag{1}$$

$$\varphi(\alpha,\beta) \leq \alpha + \beta \tag{2}$$

$$\varphi(\alpha,\beta) \leq 1 - \frac{1}{2}(1-\alpha)(1-\beta^2) \tag{3}$$

Beweis. s. [RVW01] □

Proposition 10. *Mit Hilfe des Zig-Zag-Produkts kann man eine explizite Expanderfamilie konstruieren:*

Sei H ein $(d^4, d, \frac{1}{4})$-Graph für ein konstantes d (explizite Konstruktion dafür gibt es in [RVW01] Proposition 5.3), und definiere rekursiv:

$$\begin{aligned} G_1 &= H^2 \\ G_{n+1} &= (G_n)^2 ⓩ H \end{aligned}$$

Dann ist G_n ein $(d^{4n}, d^2, \frac{1}{2})$-Graph für alle n.

Beweis. Sei G_n ein $(d^{4n}, d^2, \frac{1}{2})$-Graph, so ist $(G_n)^2$ ein $(d^{4n}, d^4, \frac{1}{4})$-Graph und nach Theorem 9 (2) gilt: G_{n+1} ist ein $(d^{4(n+1)}, d^2, \frac{1}{2})$-Graph. □

2 USTCON is in LOGSPACE

Definition 11. *Das Entscheidungsproblem USTCON ist wie folgt definiert: Gegeben ein ungerichteter Graph G und zwei Knoten s und t. Gibt es einen Pfad von s nach t?*

Definition 12. *LOGSPACE enthält Entscheidungsprobleme, die ein deterministischer Algorithmus mit Speicherplatz logarithmisch in der Eingabe lösen kann. (Die Probleminstanz ist dabei auf Read-Only-Speicher kodiert.)*

LOGSPACE $\subseteq$ P: Das es nur logarithmisch viel Speicher gibt, gibt es nur polynomiell viele Speicherzustände. Die Position im Programmcode ist gegen eine Konstante (die Programmlänge) beschränkt. Um zu terminieren, kann das Programm also nur polynomiell lang laufen, weil es sonst in eine Endlosschleife gerät.

Proposition 13. *(Erinnerung) Sei G ein d-regulärer Graph mit n Knoten, $h(G) \geq c > 1$ für konstantes c (also unabhängig von n). Dann hat G logarithmischen Durchmesser in n, also für Knoten s und t existiert ein Pfad mit Länge in $O(\log n)$.*

Beweis. Bezeichne $B_r(S)$ die Menge an Knoten, deren Abstand zur Knotenmenge S bezüglich der Graphenmetrik kleiner gleich r ist. Es ist $|B_r(S)| \geq \min(\lfloor \frac{n}{2} \rfloor + 1, c^r|S|)$, denn per definitionem ist $|B_1(S)| \geq c|S|$ für jede Knotenmenge S mit höchstens $\frac{n}{2}$ Knoten. Für $r = 1 + \log_c \frac{n}{2}$ ist dann wegen $c^r \geq \frac{n}{2}$ sowohl $|B_r(\{s\}) > \frac{n}{2}|$ als auch $|B_r(\{t\}) > \frac{n}{2}|$, also sind diese Mengen nicht disjunkt, und es existiert ein Pfad der Länge $2r = 2 + 2\log_c \frac{n}{2} \in O(\log n)$ von s nach t. □

Proposition 14. *USTCON ist in LOGSPACE für Expandergraphen mit $h(G) \geq c > 1$, c konstant.*

Beweis. Zähle alle Pfade der Länge $r = 2\lceil \log_c \frac{n}{2} \rceil \in O(\log n)$ als Zahlen der Länge r zur Basis d auf und laufe sie ab. Da d konstant und $r \in O(\log n)$, wird $O(\log n)$ Speicherplatz verbraucht. Bei Ablaufen muss immer nur der aktuelle Knoten im Speicher stehen, dieser kann in $O(\log n)$ kodiert werden. □

Theorem 15. *$USTCON \in LOGSPACE$.*

Beweis. Folgender Algorithmus löst das Problem für einen Eingabegraphen $\hat{G}$:

1. Sei H ein $(d^{16}, d, \frac{1}{2})$-Graph für ein $d \geq 5$
2. Transformiere $\hat{G}$ in einen d^{16}-regulären Graphen G mit n Knoten
3. Setze $l \in \mathbb{N}$ als die kleinste Zahl, für die $(1 - \frac{1}{(dn)^2})^{2^l} < \frac{1}{2}$
4. Setze $G_0 := G$ und rekursiv $\mathcal{T}_i(G, H) := (G_{i-1} ⓩ H)^8$
5. Führe auf $\mathcal{T} := \mathcal{T}_l(G, H)$ obigen Algorithmus aus.

Anmerkung: Zusammenhangskomponenten bleiben bestehen wegen der Eigenschaften von Zig-Zag Product und Graph Powering.

Offensichtlich können keine der Graphen außer H als Zwischenergebnisse gespeichert werden.

Man muss für Knoten in $\mathcal{T}$ also Nachbarn in LOGSPACE bestimmen können, ohne explizite Beschreibungen der G_i zu haben. □

Lemma 16. *(s. [Rei05] Proposition 4) Es gibt einen $(d^{16}, d, \frac{1}{2})$-Graph.*

Beweis. Obige Expanderfamilie liefert uns $(d^{4n}, d^2, \frac{1}{2})$-Graphen für alle $n \in \mathbb{N}$.

Für $n = 8$ ist das ein $((d^2)^{16}, d^2, \frac{1}{2})$-Graph. □

Proposition 17. *Ein beliebiger Graph $\hat{G}$ kann zu einem Graphen G mit Grad d^{16}, $d^{16} \geq 3$, regularisiert werden.*

Beweis. Graph 3-regulär machen, in dem man v durch Zyklus der Länge $d(v)$ ersetzt. ($d(v)$ ist der Knotengrad. Achtung: Schleifen erhöhen $d(v)$ nur um 1.)

Dann mit Schleifen auf Grad d^{16} auffüllen.

□

Lemma 18. *Abgeschwächte Form von [Rei05] Lemma 5:*
In einem zusammenhängenden Graphen gilt: $\lambda \leq 1 - \frac{2}{(dn)^2}$.
In nicht zusammenhängenden Graphen gilt dies für die ZHK.

Beweis. Nach Theorem 4.11 [HLW06]: $\frac{d-\lambda'}{2} \leq h(G) \leq \sqrt{2d(d-\lambda')}$.

Da G zusammenhängend ist, ist $|E(S, \overline{S})| \geq 1$ und damit

$$h(G) = \min_{|S| \leq \frac{n}{2}} \frac{|E(S, \overline{S})|}{|S|} \geq \frac{2}{n}$$

$$\begin{aligned}
\frac{2}{n} &\leq \sqrt{2d(d-\lambda')} \\
\frac{4}{n^2} &\leq 2d(d-\lambda') \\
\frac{2}{dn^2} &\leq d-\lambda' \\
d-\frac{2}{dn^2} &\geq \lambda' \\
1-\frac{2}{(dn)^2} &\geq \frac{\lambda'}{d}=\lambda
\end{aligned}$$

□

Lemma 19. *(s. [Rei05] Lemma 11) Jede ZHK von $\mathcal{T}$ ist ein $(nd^{16l}, d^{16}, \frac{1}{2})$-Graph.*

Beweis. Folgende Betrachtungen betreffen jeweils die ZHK des genannten Graphen:

Die ZHK sind zusammenhängend, folglich $\lambda(G_0) \leq 1 - \frac{1}{(dn)^2}$.

Da $l \in \mathbb{N}$ die kleinste Zahl war, für die $(1 - \frac{1}{(dn)^2})^{2^l} < \frac{1}{2}$, reicht zu zeigen:

$$\lambda(G_i) \leq \max\{\lambda(G_{i-1})^2, \frac{1}{2}\}$$

Setze $\lambda := \lambda(G_{i-1})$. Wegen der Eigenschaft des Zig-Zag-Produkts $\varphi(\alpha, \beta) \leq 1 - \frac{1}{2}(1-\alpha)(1-\beta^2)$ mit $\alpha = \lambda$ und $\beta = \frac{1}{2}$ gilt:

$$\lambda(G_{i-1} ⓩ H) \leq 1 - \frac{3}{8}(1-\lambda) < 1 - \frac{1}{3}(1-\lambda)$$

Damit ist wegen $G_i = (G_{i-1} ⓩ H)^8$:

$$\lambda(G_i) < (1 - \frac{1}{3}(1-\lambda))^8$$

Für $\lambda < \frac{1}{2}$ ist dies kleiner als $\frac{1}{2}$.

Ansonsten kann man zeigen, dass $(1-\frac{1}{3}(1-\lambda))^4 < \lambda$ und folglich $\lambda(G) < \lambda^2$.

□

Proposition 20. *Der Algorithmus aus Proposition 14 operiert korrekt auf $\mathcal{T}$, da $h(\mathcal{T}) \geq c > 1$ für ein konstantes c unabhängig vom Eingabegraphen.*

Beweis. G hat zweitgrößten normalisierten Eigenwert $\lambda(G)$ kleiner als $\frac{1}{2}$.

Nach Theorem 4.11 [HLW06]: $\frac{d-\lambda'}{2} \leq h(G) \leq \sqrt{2d(d-\lambda')}$, also insbesondere:

$$\begin{aligned} d\frac{1-\lambda(G)}{2} &\leq h(G) \\ \frac{d}{4} &\leq h(G) \end{aligned}$$

Also ist für $d \geq 5$ $h(G)$ nach unten gegen eine Konstante größer 1 beschränkt, und die Voraussetzung für die Korrektheit des Algorithmus ist erfüllt.

□

Anm.: *Man kann allgemeiner auch zeigen, dass für jedes $\lambda < 1$ ein $\varepsilon > 0$ existiert, sodass für alle (n, d, λ)-Graphen $h(G) > 1 + \varepsilon$.*

Proposition 21. *All das geht in LOGSPACE .*

Beweis. Wir müssen in $\mathcal{T}$ Nachbarn in logarithmisch viel Platz bestimmen können.

1. H ist konstant groß und kann im Algorithmus hardcodiert werden.
2. Kommt gleich in Lemma 22
3. Der Wert l ist in $O(\log n)$ und passt damit in $O(\log \log n)$ Speicher
4. Kommt gleich in Lemma 23
5. Der Grad ist konstant d^{16}, also $O(\log n)$-mal konstanter Platzbedarf.

□

Lemma 22. *Rot_G auswerten geht in LOGSPACE .*

Beweis. Betrachtet man die Konstruktion, um $\hat{\mathrm{G}}$ zu G zu regularisieren, so sieht man: Jedem Knoten (der durch eine Zahl zwischen 1 und $\sum_{v \in V(\hat{\mathrm{G}})} d(v)$, was die Anzahl der Knoten in G ist, gegeben ist) in G kann man einen Paar aus Knoten und Kantennummer (bezüglich dieses Knotens) in $\hat{\mathrm{G}}$ zuordnen. Man suche sich also eine (in LOGSPACE auswertbare) Bijektion dazwischen. Weiterhin muss man jedem Knoten in G seinen drei Kanten Knotennummern zuordnen (die Schleifen bekommen einfach alle Indizes größer als 3, und Rot_G ist in diesem Fall trivial auswertbar). Beispielsweise kann die erste Kante diejenige sein, die aus dem Zyklus herausführt, und die zweite und dritte die vorige bzw. nächste Kante im Zyklus sein. Dann kann man für G zu einem gegebenen Knoten das Ziel der i-ten Kante in LOGSPACE ausrechnen, wenn man nur $\hat{\mathrm{G}}$ im Speicher gegeben hat (unabhängig vom Eingabeformat, solange es ein einigermaßen sinnvolles ist). □

Lemma 23. *([Rei05] Lemma 13) Rot_{G_i} auswerten geht in LOGSPACE .*

Beweis. Auswerten von $Rot_{G_i}((v, a_0, ..., a_{i-1}), a_i)$ nach $((v, a_0, ..., a_{i-1}), a_i)$:

- für $j = 1, ..., 8$
  - $(a_{i-1}, k_{i,2j}) := Rot_H(a_{i-1}, k_{i,2j})$
  - $((v, a_0, ..., a_{i-2}), a_{i-1}) := Rot_{G_{i-1}}((v, a_0, ..., a_{i-2}), a_{i-1})$
  - $(a_{i-1}, k_{i,2j+1}) := Rot_H(a_{i-1}, k_{i,2j+1})$
- $(k_{i,1}, ..., k_{i,16}) := (k_{i,16}, ..., k_{i,1})$

G_i hat Knoten in $[n] \times ([d]^{16})^i$, also von der Form $(v, a_0, ..., a_{i-1})$. Solch ein Knoten ist ein Knoten $(v, a_0, ..., a_{i-2})$ in G_{i-1} zusammen mit einem Knoten a_{i-1} in H, denn $G_i = (G_{i-1} ⓩ H)^8$, i.e. jeder Knoten in G_i wurde durch eine Kopie von H ersetzt.

$a_i = (k_{i,1}, ..., k_{i,16}) \in [d^{16}]$ kodiert einen Pfad in $G_{i-1} ⓡ H$ aus 8 Zig-Zag-Schritten, also einen Pfad der Länge 8 in $G_{i-1} ⓩ H$, was einer Kante in G_i entspricht. Folglich kodiert a_i eine Kantennummer in G_i.

Bei der Nachbarbestimmung in $\mathcal{T}$, also bei der Auswertung von $Rot_{\mathcal{T}}((v, a_0, ..., a_{l-1}), a_l)$, wird nur der Speicherplatz verbraucht, in dem an Anfang der Anfrageknoten und Kantennummer steht. Dieser wird schrittweise überschrieben. Zusätzlich muss bei jedem Rekursionsaufruf die lokale Zählvariable j gespeichert werden, die aber konstante Größe hat. Beides braucht insgesamt $O(\log n)$ Speicherplatz. □

Literatur

[HLW06] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications, 2006.

[Rei05] Omer Reingold. Undirected connectivity in log-space, 2005.

[RVW01] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders, 2001.