Graphalgorithmen

Markus Vieth

12. Dezember 2016

Graphalgorithmen

Markus Vieth

12. Dezember 2016

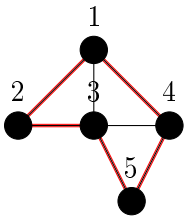
Vorwort

Dieses Skript basiert auf unserer Mitschrift der Vorlesung Computergrafik und VR im WS 2016/17 an der JGU Mainz (Dozent: Prof. Dr. E. Schömer).
Es handelt sich nicht um eine offizielle Veröffentlichung der Universität.
Wir übernehmen keine Gewähr für die Fehlerfreiheit und Vollständigkeit des Skripts.
Fehler können unter Github gemeldet werden. Die aktuelle Version dieses Skriptes ist ebenfalls auf Github zu finden.

Inhaltsverzeichnis

0.1 Exponentielle Algorithmen

Beispiel: Traveling Salesman Problem



- Gegeben Graph $G = (\overset{=\{1,\ldots,n\}}{V}, E)$, Kantenkosten $c : E \to \mathbb{R}$
- Gesucht: Zyklus, der jeden Knoten genau ein Mal besucht und minimale Kosten hat.

0.1.1 Trivialer Algorithmus

Iteriere über alle Permutationen der Knoten, berechne jeweils die Kosten der Tour und speichere die billigste.

Laufzeit

$$O(n! \cdot m)$$

Für $f : \mathbb{N} \to \mathbb{N}$ definieren wir:

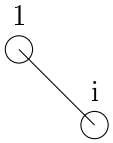
$$O^*(f) : \{g : \mathbb{N} \to \mathbb{N} | g(n) \leq f(n) \cdot \text{poly}(m)\}$$

für ein Polynom p

Also: Algorithmus hat eine Laufzeit von $O^*(n!)$

Ziel: Algorithmus für TSP der bzgl. der O^*-Notation schneller ist. (einen Algorithmus mit Laufzeit $O^*(2^m)$)
Für alle $\{1\} \subseteq S \subseteq V$ und alle $v \in S$ berechnen wir:
$O(S, v) =$ Kürzester Pfad, der jeden Knoten in S genau ein Mal besucht und in v endet.
Die Länge der TSP-Tour ist dann

$$\min_{2 \leq i \leq n} O(V, I) + c(i, 1)$$



$O(S, i)$ können wir wie folgt berechnen:

$$O(1, 1) = 0$$

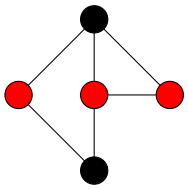
$$\underbrace{O(S, i)}_{2^n \text{Werte}} = O(S \setminus \{1\}, j) + c(j, i) \quad j \in S \setminus \{i\}$$

0.1.2 Laufzeit

$$O^*(2^n)$$

Beispiel

0.2 Vertex-Cover (VC)-Problem



- Gegeben: Graph $G = (V, E)$
- VC: Teilmenge $U \subseteq V$, so dass $e \cap U \neq \emptyset$ für alle $e \in E$
- Gesucht: VC minimaler Kardinalität

0.2.1 Trivialer Algorithmus

Probiere alle Teilmengen $U \subset V$ aus

0.2.2 Laufzeit

$O^*(2^n)$

0.3 Branching-Algorithmus für VC

Wir wählen $v \in V$ und machen die Fallunterscheidung $u \in U$ und $u \notin U$

- $u \in U$: Alle Kanten adjazent zu u sind damit überdeckt und man löst rekursiv das Problem ohne v und dessen adjazenten Kanten
- $u \in V$: Alle zu v inzidenten Knoten müssen zu U hinzugefügt werden und nun löst ma das Problem auf dem Graphen, in dem alle diese Knoten und deren adjazenten Kanten gelöscht werden

0.3.1 Pseudocode

```
1  finde\_VC(G=(V,E))
2      - entferne alle Knoten vom Grad 0
3      - Sei u ∈ V
4      - S_1 = find_VC(G[V \ {u}]) ∪ {u}
5      - S_2 = find_VC(G[V \ {u} \  N[u]  ]) ∪ N[u]
                                  Nachbarn von n
6      - return kleineres von S_1 und S_2
```

Laufzeit

Da wir „nur“ bzgl. der O^*-Notation analysieren, analysieren wir $T(n)$: maximale mögliche Anzahl von Teilproblemen bei einem Graph mit n Knoten
Rekursionsgleichung für $T(n)$: (Annahme T wächst monoton)

$$T(n) \leq \begin{cases} T(n-1) + T(n-2) & \text{für} n \geq 2 \\ 1 & \text{für} n \leq 1 \end{cases}$$

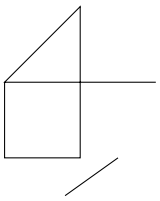
Fibonacci-Zahlen, also $T(n) \leq \left(\frac{1+\sqrt{5}}{2}\right)^n \approx 1,618^n$

Laufzeit

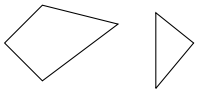
$$O^*\left(\frac{1+\sqrt{5}}{2}\right)^n$$

0.3.2 Verbesserung

Statt $c \in V$ beliebig zu wählen, wähle v mit maximalem Grad
Zuvor entfernen wir nicht nur die Knoten com Grad 0, sondern auch die vom Grad 1 mit folgender Überlegung:



- Hat ein Knoten Grad 1, so nimm den anderen Endpunkt in U auf und vereinfache den Graphen
  $\rightsquigarrow$ nun haben alle Knoten mindestens Grad 2
- Haben alle Knoten Grad 2, so haben wir eine Menge von Zyklen und knönnen das VC-Problem direkt lösen
  $\rightsquigarrow$ Wir können davon ausgehen, dass es mindestens einen Knoten vom Grad ≥ 3 gibt



Verbesserte Abschätzung für die Anzahl T der Teilprobleme

$$T(n) \leq T(n-1) + T(n-4) \in O^*(1,41^n)$$

0.4 Laufzeitanalyse vom Branching-Programm in O^*-Notation

Wir wählen Rekursionsgleichung der Form:

$$T(n) \leq \begin{cases} T(n-d_1) + T(n-d_2) + \ldots + T(n-d_p) & \text{für} n \geq d \\ 1 & \text{für} n \leq d \end{cases}$$

in O^*-Notation abschätzen, wobei $d = \max(d_1, d_2, \ldots, d_p)$
Wir wollen ein möglichste kleines $\lambda \geq 0$ haben, so dass $T(n) \leq \lambda^n$
Wir müssen λ so wählen, dass

$$\lambda^n \geq \lambda^{n-d_1} + \lambda^{n-d_2} + \ldots + \lambda^{n-d_p}$$

(dann können wir mit obiger Rekursionsgleichung leicht per Induktion beweisen)

$$T(n) \leq \delta^n$$

Umschreiben:

$$\lambda^d \geq \lambda^{d-d_1} + \lambda^{d-d_2} + \ldots + \lambda^{d-d_p}$$

$$\lambda^d - \lambda^{d-d_1} - \lambda^{d-d_2} - \ldots - \lambda^{d-d_p} \geq 0$$

P ist ein Polynom vom Grad d und $P(0) < 0$ (solange $p > 1$) und p hat genau eine positive Nullstelle. Diese muss man nun bestimmen und hat dann die Rekursionsgleichung gelöst.
Zahlenbeispiel: $T8N) \leq T(n - d_1) + T(n - d_2)$

	1	2	3	
1	2	1,618	1,46	$\cdots$
2		1,41	1,32	$\cdots$
3			1,25	$\cdots$
			$\vdots$	

0.5 Parametrische Algorithmen aus den Klassen FPT und XP

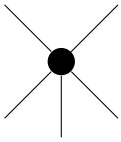
- Wir wollen Algorithmen für NP-schwere Probleme herleiten
- Oft kennt man eine „Eigenschaft“ der zu lösenden Instanzen und hofft, dass diese die Fragestellung „einfacher“ macht
- Lässt sich die Eingenschaft in einer Zahl beschreiben, nenn man das Problem ein Parametrisches Problem

Beispiel Vertex-Cover mit zusätzlicher Annahme „es existiert ein kleines Cover“. Sei k die Größe des minimalen VC. Versuche Algorithmus zu finden, der effizient ist, falls k „klein“. (XP Algorithmus)

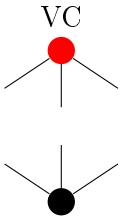
trivialer Algorithmus

Zähle nur Teilmengen bis zur Größe k auf $\rightsquigarrow$ Laufzeit $O(n^k)$
Geht es besser?

- Entferne alle Knoten vom Grad 0 und 1, wie oben
- Füge jeden Knoten mit Grad $\geq k + 1$ zu VC hinzu (da sonst die Nachbarschaft im VC wäre und damit zu groß)
- Iterire



Am Ende hat jeder Knoten einen Grad zwischen 2 und k.



Hat der Graph nun mehr als k^2 Knoten, so gib es kein VC mit der Kardinalität $\leq k$
$\rightsquigarrow$ Also kann der Graph nur k^2 Knoten haben.
Auf diesem Restgraph benutzen wir nun den Branching-Algorithmus und erzielen eine Laufzeit von $O(\underbrace{\text{poly}(m)}_{\text{Vereinfachen des Graphen}} + \left(k^2\right)^{1,4})$ (FPT)

0.5.1 Definition

Parametrisches Problem $L \subseteq \Sigma^* \times \mathbb{N}$ für eine Instanz $(x,t)(x \in \Sigma^*, k \in \mathbb{N})$ heißt x die Eigenschaft, $n - |x|$ die Länge der Eingabe der Parameter

- Ein parametrisches Problem L heißt fixed-parameter-tractable (FPT) wenn es ein en Algorithmus für L gibt mit Laufzeit $f(k) \cdot m^2$ für $c \in \mathbb{N}$ und f berechenbare Funktion
- Ein Problem heißt in XP, wenn es für jedes feste $k \in \mathbb{N}$ einen polynomischen Algorithmus für den alle Instanzen mit Parameter k gibt. (Laufzeit ist immer $O(n^{f(k)})$