Proseminar
Perlen der theoretischen Informatik

Zufallsirrfahrten auf Graphen

Überblick

- Das Problem
- LOGSPACE L, NL
- RL
- PATH, UPATH
- Zufallsirrfahrten
- Wahrscheinlichkeit des Erfolges, Irrfahrtslänge

Das Problem

Wir wollen die (Platz)-Komplexität des Problems betrachten, in einem gerichteten bzw. ungerichteten Graphen einen Weg (Pfad), von einem beliebigen Startknoten zu einem beliebigen Endknoten, zu finden. Die Komplexität ist im ungerichteten Graphen einfacher als in einem gerichteten Graphen. Diese erreichen wir durch Zufallsirrfahrten bzw. universelle Durchlaufsequenzen.
Wir betrachten das Problem für ungerichtete Graphen genauer.

Was ist ein Weg?
Eine Folge von Knoten $W = (w_0, \dots, w_l)$, wobei w_0 Start- und w_l Endknoten ist.

Wir kennen bereits eine Methode um in einem gerichteten Graphen den kürzesten Weg zu finden: der „Dijkstra-Algorithmus“.
Allerdings sind wir nun an einem Algorithmus interessiert, wobei wir von anderen Bedingungen ausgehen. Wir haben keine globale Informationen über den Graphen, sondern besitzen nur ein lokales Wissen beim Durchlaufen des Graphen (von dem Knoten wo wir uns gerade befinden und seiner direkten Umgebung). Hinzu kommt, dass wir nur logarithmisch viel Speicherplatz (Arbeitsspeicher) besitzen.

LOGSPACE

Als erstes untersuchen wir die Menge von Problemen, die wir mit logarithmisch Speicherplatz beschränkten Turingmaschinen lösen können.
Allgemein gilt:

- DSPACE(f) Menge aller Sprachen, die eine k-Band TM akzeptiert, wobei sie bei Eingabelänge n nur f(n) Platz benötigt
- NSPACE(f) Menge aller Sprachen, die eine k-Band NTM akzeptiert, wobei sie bei Eingabelänge n nur f(n) Platz benötigt
-

So bezeichnen wir die Menge der algorithmischen Entscheidungsprobleme, die sich durch deterministische Algorithmen mit nur logarithmisch viel Speicherplatz lösen lassen mit L, d.h. $L = DSPACE(O(\log n))$.
Mit n bezeichnen wir die Eingabegröße (z.B. Knotenanzahl im Graphen).
Es zählt nur der Speicherplatz auf dem Arbeitsband, nicht jedoch der auf dem Ein-Ausgabeband.

Analog können wir dies für nichtdeterministische Algorithmen definieren, $NL = NSPACE(O(\log n))$.

Es ist ein offenes Problem, ob die Inklusion $L \subseteq NL \subseteq P$ echt ist oder nicht. Ähnlich wie beim P = NP Problem kann man diese Frage mit Hilfe eines entsprechenden Vollständigkeitsbegriffes behandeln.

In diesem Kontext machen allerdings Polynomialzeit-Reduktionen keinen Sinn, da $NL \subseteq P$ gilt.

Beweis NL $\subseteq$ P:
L, NL Maschinen sind off-line Turingmaschinen, d.h. mit separatem Eingabeband. Eine Konfiguration dieser Maschinen ist gekennzeichnet durch:

- den momentanen Zustand (|Z|)
- die Lesekopf-Position auf dem Eingabeband (n + 2)
- die Kopfposition auf dem Arbeitsband (c*log n)
- die Inschrift des Arbeitsbandes ($|A|^{c*\log n}$)

Sei Z die Zustandsmenge und A das Arbeitsalphabet. Dann gibt es für eine c*log n Speicherplatz-beschränkte Turingmaschine höchstens
$a = |Z| * (n+2) * (c*\log n) * |A|^{c*\log n} = O(n^k)$ viele verschiedene Konfigurationen, für eine geeignete Konstante k.
Das heißt, wenn eine Rechnung mehr als a Schritte macht, muss sich eine Konfiguration wiederholen und somit befindet sie sich in einer Endlosschleife (kann nicht akzeptieren). Somit kann man alle Konfigurationen generieren, welche von der Startkonfiguration aus erreichbar sind, bis evtl. eine akzeptierende Konfiguration auftaucht. Dies nimmt höchstens polynomial viel Zeit $O(n^k)$ in Anspruch.

q.e.d.

Ein Problem A ist auf ein Problem B log-reduzierbar, falls es eine logarithmisch Speicherplatz-beschränkte Turingmaschine M mit separatem Ein- Ausgabeband (dessen Platzbedarf nicht berücksichtigt wird) gibt, so dass für alle x gilt: $x \in A \Leftrightarrow M(x) \in B$.

Diese log-Reduzierbarkeit ist eine transitive Relation.

Ein Problem X heiße NL-vollständig, wenn $X \in NL$ und wenn für alle $A \in NL$ gilt, dass A log-reduzierbar auf X ist.

Motivation: Das (Path-) Problem, einen Pfad in einem gerichteten Graphen zu finden, ist NL-vollständig.

RL

Jetzt wollen wir den Zufall ins Spiel bringen. Wir betrachten welche Probleme Turingmaschinen lösen können, welche Zufallsentscheidungen verwenden dürfen, aber immer noch logarithmisch platzbeschränkt sind.

So bezeichnen wir die Menge der algorithmischen Entscheidungsprobleme, die sich durch Algorithmen lösen lassen, welche nur logarithmisch viel Speicherplatz benötigen, Zufallsentscheidungen in ihrer Rechnung verwenden dürfen und eine polynomiale Laufzeit besitzen mit RL.
Die Ausgabe der Algorithmen wird zu einer zweiwertigen Zufallsvariable. D.h. das Berechnen eines Problems A sieht wie folgt aus.

- $x \in A \Rightarrow P[M \text{ akzeptiert } x] \geq ½$
- $x \notin A \Rightarrow P[M \text{ akzeptiert } x] = 0$

Es ist klar, dass gilt $L \subseteq RL \subseteq NL$.

Motivation: Das (UPATH-) Problem, einen Pfad in einem ungerichteten Graphen zu finden, ist in RL.

Randomisierte log-Speicherplatz beschränkte Maschinen müssen nicht polynomial zeitbeschränkt sein (kann exponentiell sein).

Bsp.1: Algorithmus mit beliebiger Eingabelänge n:

```
Erfolg := false;
While Not Erfolg Do
    Wähle mit gleicher Wahrscheinlichkeit a) oder b) aus:
    a) Bewege Eingabelesekopf ums eins nach rechts. Falls Eingabe
        (der Länge n) überschritten wird, setze Erfolg = True;
    b) Bewege den Eingabe-Lesekopf zurück auf das erste Zeichen der Eingabe
End
```

→ beim ersten Zeichen beginnend tritt Erfolg mit Wahrscheinlichkeit 2^{-n} ein, jeder Versuch kostet mindestens einen Schritt → mittlere Laufzeit bis Erfolg eintritt ist mindestens 2^n

Man kann zeigen, dass NL = Menge der Probleme, welche randomisierte log-Speicherplatz beschränkte (nicht polynomial zeit-beschränkte) Maschinen lösen können.

PATH, UPATH

Nun kommen wir zurück zu unserem eigentlichen Problem. Wir erfassen allgemein die Teilmenge unserer Probleme (einen Pfad in einem Graphen von a nach b zu finden), welche eine positive Lösung besitzen und werden erste Aussagen über ihre Komplexität machen. Zunächst unterscheiden wir zwischen den Problemen in einem gerichteten (PATH) und ungerichteten (UPATH) Graphen.

- Def.: PATH = {(G, a, b) | G ist ein gerichteter Graph und a, b sind gerichtete Knoten in G, so dass ein Pfad von a nach b existiert}
- auch GAP („graph accessibility problem“) genannt
- Path ist NL-vollständig
  - Beweis:
    - Path in NL klar: Angeben einer nichtdeterministischen TM, welche einen Pfad von a nach b systematisch rät und nach n (Knotenanzahl) Schritten stoppt (es muss nur die aktuelle Knotennummer gespeichert werden)
    - es sei ein beliebiges Problem A aus NL mittels O(log n) NTM M gegeben
    - man bilde alle Konfigurationen von M, die Anzahl ist polynomial beschränkt s.o.
    - eine Reduktionsmaschine (von A nach Path) übersetzt alle Konfigurationen zu Knoten
    - alle Nachfolge-Konfigurationen (endliche Anzahl) werden durch gerichtete Kanten verbunden
    - a = Startkonfiguration
    - b = bestimmte akzeptierende Konfiguration (falls vorhanden)
    - alle Konfigurationen haben Größe O(log n) → Reduktionsmaschine ist log-bandbeschränkt q.e.d.

PATH, UPATH

Jetzt betrachten wir die Teilmenge unserer Probleme einen Pfad in einem ungerichteten Graphen von a nach b zu finden, welche eine positive Lösung besitzen.

- Def.: UPATH = {(G, a, b) | G ist ein ungerichteter Graph und a, b sind zwei Knoten in G, so dass ein Pfad zwischen a und b existiert}
- Es ist offen ob UPATH NL-vollständig
  (Beweis für PATH nicht übertragbar, da eine Konfiguration zwei Vorgängerkonfigurationen haben kann siehe Aufgabe 39, 40)
- UPATH ∈ NL (klar siehe PATH)

→ Das UPATH-Problem ist möglicherweise einfacher als das PATH-Problem.
UPATH (und PATH) Problem liegen möglicherweise nicht in L Wir werden zeigen, dass UPATH ∈ RL

Es gibt viele im Sinne der Komplexität äquivalente Aufgabenstellungen für gerichtete bzw. ungerichtete Graphen, z.B. Hamiltonkreis-Problem ist NP-vollständig oder das Graphenisomorphieproblem.

Zufallsirrfahrten

Das Mittel um unser UPATH-Problem unter Beachtung unserer Bedingungen (log-Arbeitsspeicher-Beschränktheit, polynomiale Laufzeit, lokales Wissen) zu lösen sind Zufallsirrfahrten. Wir werden zeigen, dass es einen Algorithmus gibt, der unser Problem löst und in RL enthalten ist, also somit unsere Bedingungen erfüllt.

- Zufallsirrfahrt:
  - Wie muss allgemein unser randomisierter Algorithmus M für „UPATH ∈ RL“ aussehen?
    - Er beginnt in dem Graphen beim Knoten a.
    - Er läuft solange bis evtl. der Knoten b erreicht wird.
    - Er muss nur die aktuelle Knotennummer speichern → benötigt maximal log n Bits
    - Er besitzt eine polynomiale Laufzeit.

    Wir können durch unser lokales Wissen natürlich nicht verhindern, dass Knoten mehrfach durchlaufen werden.

Das ist unser Algorithmus M:

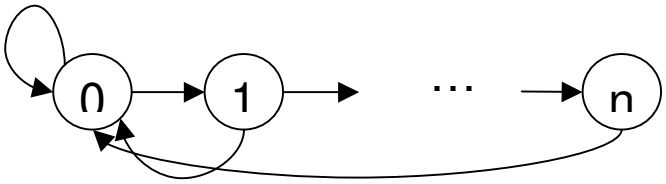
```
Input (G, a, b);
v := a;
For i := 1 To p(n) Do
        Wähle unter den Nachbarknoten von v einen Knoten w unter Gleichverteilung
aus;
        v := w;
        If v = b Then Accept End;
End;
Reject;
```

Dass unser Algorithmus dem Namen Zufallsirrfahrt gerecht wird und unser Problem löst ist jetzt einzusehen. Es muss allerdings noch gezeigt werden, dass unser Algorithmus wirklich in RL liegt.

Formulieren wir nun unsere Behauptung*:

- Die Funktion p(n) kann als Polynom gewählt werden, so dass die Definition von RL erfüllt wird:
  - $(G, a, b) \in \text{UPATH} \Rightarrow P[\text{M akzeptiert } (G, a, b)] \geq ½$
  - $(G, a, b) \notin \text{UPATH} \Rightarrow P[\text{M akzeptiert } (G, a, b)] = 0$

Der Algorithmus funktioniert allerdings nicht für gerichtete Graphen.
Hier ein Bsp.-Graph (siehe Bsp.1), wo die Wahrscheinlichkeit den Knoten b zu erreichen nur 2^{-n} beträgt und somit die mittlere Irrfahrtslänge 2^n ist, also nicht polynomial (a = 0, b = n).



Bevor wir mit unserem Beweis beginnen, fassen wir zusammen, was eine Zufallsirrfahrt ist:

- eine Folge $W = (v_1, v_2, v_3, \ldots)$ wobei,
- v_1 = Startknoten
- v_{i+1} wird unter Gleichverteilung der Nachbarknoten von v_i ausgewählt ($\{v_i, v_{i+1}\}$ ungerichtete Kante in G)

Jetzt definieren wir noch das Anfangsstück eines Pfades: W_i = endliche Teilfolge der ersten i Knoten

Beweis *:

Notationen

- v_i bezeichnet den i-ten Knoten im Weg W
- $P_v = \lim_{n\to\infty} \frac{|\{i \leq n \mid v = v_i\}|}{n}$ ≙ Wahrscheinlichkeit des Auftretens von v in W (*1)
  (Anzahl des Auftretens von v / Anzahl der Schritte in W)
- $1/P_v$ ≙ Erwartungswert der Distanz zwischen zwei aufeinander folgenden Vorkommen von v in W ≙ mittlere Irrfahrtzeit von v nach v

Betrachtung:

- ungerichtete Kanten {u, v} als zwei gerichtete (u, v), (v, u)
- G ist zusammenhängend (klar, in einem nicht zusammenhängenden Graphen gibt es nur eine Lösung, wenn sich a und b im selben zusammenhängendem Teilgraphen befinden)

Zwischen-Behauptung 1.) z.z.:

Die Wahrscheinlichkeit bei einer Zufallsirrfahrt eine beliebige gerichtete Kante zu erreichen = 1/2e

- e = Anzahl der ungerichteten Kanten
- n = Anzahl der Schritte
- $\sum P(u,v) = 1$ (klar, die Wahrscheinlichkeiten verteilen sich irgendwie über alle Kanten)

- für eine Kante folgt aus (*1) → $P_{(u,v)} = \lim_{n\to\infty} \frac{|\{i \le n | (v_i, v_{i+1}) = (u, v)\}|}{n}$

Das heißt, es gibt keine Kante (u,v) die eine größere Wahrscheinlichkeit hat, als die gemittelte Wahrscheinlichkeit ihrer (über den Knoten v) verbundenen Nachbarkanten

Beweis 1.) indirekt,

Annahme: Nicht für alle Kanten gilt: $P_{(u,v)} = 1/2e$

So muss es Kanten mit größerer und kleinerer Wahrscheinlichkeit geben, damit die Summe aller Wahrscheinlichkeiten eins bleibt.
Sei $p_{max} > 1/2e$ die maximal auftretende Wahrscheinlichkeit einer Kante.
Da G zusammenhängend ist, muss mindestens eine Kante (u, v) existieren mit $P(u,v) = p_{max}$, so dass mindestens eine ihrer Nachbarkanten eine kleinere Wahrscheinlichkeit hat.
Andernfalls hätten alle Kanten eine maximale Wahrscheinlichkeit und deren Summe wäre nicht eins ($p_{max} > 1/2e$).

→ $\exists$ Kante (u, v) : $P_{(u,v)} > \frac{1}{d(v)} * \sum_{(v,w) \in G} P_{(v,w)}$

Andererseits muss gelten, dass die (u, v) zugeordnete Wahrscheinlichkeit sich exakt auf die gemittelten Wahrscheinlichkeiten der Nachfolgekanten aufteilen muss:

→ $P_{(u,v)} = \frac{1}{d(v)} * \sum_{(v,w) \in G} P_{(v,w)}$

dies folgt aus der Auswahl des Folgeknotens in v unter Gleichverteilung mit der Wahrscheinlichkeit 1/d(v)

↯ q.e.d. (1)

Somit treten alle gerichteten Kanten mit der Wahrscheinlichkeit 1/2e und alle ungerichteten mit der Wahrscheinlichkeit 1/e auf.
Wir können daraus schließen: Es führen d(v) viele Kanten nach v mit der Wahrscheinlichkeit 1/2e, d.h. $P_v = d(v)/2e$ und die mittlere Irrfahrtszeit beträgt somit $1/P_v = 2e/d(v)$

Jetzt ist es nur noch ein kleiner Schritt zu zeigen, dass die Irrfahrtslänge durch ein Polynom in n beschränkt werden kann und trotzdem mit einer Wahrscheinlichkeit ≥ ½ jeder Knoten (also auch der Zielknoten b) einmal besucht werden kann.

Ein paar weitere Zwischenschritte und Beweise sind allerdings noch nötig:

- Mit E(i, j) wollen wir den Erwartungswert für die Anzahl von Irrfahrtsschritten, bis man vom Knoten i zum Knoten j kommt, bezeichnen.
- aus $1/Pv = 2e/d(v)$ folgt $E(v, v) = 2e/d(v)$

Wir schätzen den Erwartungswert für eine beliebige Kante u, v grob ab:

$E(u, v) \leq d(u) * E(u, u) = d(u) * 2e/d(u) = 2e$

(Im Mittel muss man d(u)-mal über u gehen, um nach v zu gelangen. Schlechte Abschätzung, da man auch über andere Wege von u nach v gehen könnte ohne diese Kante zu benutzen)

E(a, G) bezeichne die mittlere Irrfahrtszeit bis alle Knoten in G von a ausgehend mindestens einmal erreicht wurden (G ist zusammenhängend).
Allgemeine Abschätzung für E(a, b):

G: $E(a, b) \leq E(a, G)$ (klar, man muss über b gegangen sein, um alle Knoten im Graphen G zu erfassen, da $b \in G$)

Sei $(v_0, v_1, \ldots, v_k)$ ein bei $a = v_0$ beginnender Pfad in G, indem alle Knoten aus G vorkommen.

Behauptung:

In einem zusammenhängenden Graphen existiert immer solch ein Pfad mit $k \leq 2n$.

Beweis: Induktion über Knotenzahl n

- IA: Graph mit einem Knoten klar
- G sei zusammenhängend mit n + 1 Knoten
- → Es gibt einen Knoten, dessen Entfernen den Zusammenhang des Restgraphen nicht zerstört (z.B. nimmt man die Blätter bei einem Baum)
- Im Restgraphen existiert nach Induktionsvoraussetzung ein Pfad der Länge $\leq 2n$
- Der entfernte Knoten war mit mindestens einer Kante mit diesem Pfad verbunden.
- Der Pfad wird um zwei Schritte verlängert um diesen Knoten mit einzubeziehen (ein Schritt zum Knoten hin und ein Schritt wieder zurück)
- So erhalten wir einen Pfad der Länge 2(n + 1) q.e.d.

Wir schätzen E(a, G), grob ab, indem wir die mittleren Irrfahrtszeiten zu allen Knoten aufsummieren ($a = v_0$):

- $E(a, G) \leq \sum E(v_i\text{-}1, v_i) \leq 2n * 2e = 4en$

Aus der Markoff-Ungleichung folgern wir:

- X sei Zufallsvariable die einen Erwartungswert besitzt und keine negativen Werte annehmen kann
- es gilt: $P[X \geq c] \leq E(X) / c$

Beweis:

- Sei Y die 0-1 wertige Zufallsvariable $Y = \begin{cases} 1, X \geq c \\ 0, X < c \end{cases}$
- Dann gilt $X \geq a * Y$
- Daraus folgt $E(X) \geq E(a * Y) = a * E(Y) = a * P[X \geq a]$ q.e.d.

Wir legen nun mit X die Laufzeit von a nach b fest und es sei c = 2E(X).
→ $P[X > 8en] \leq P[X \geq 2E(X)] \leq E(X)/(2E(X)) = ½$
→ Die Wahrscheinlichkeit einen festen Knoten b in G nicht zu besuchen, wenn wir bei a starten und 8en Irrfahrtsschritte machen ist höchstens ½ .

q.e.d. (*)

Zusammenfassung:
Bei einen gegebenen Graphen G, mit einem ausgezeichneten Startknoten a, einem ausgezeichnetem Zielknoten b, mit Knotenzahl n und Kantzahl e, können wir sagen dass bei einer Zufallsirrfahrt der Länge 8en (= $O(n^3)$) gilt, wenn:

- es keinen Pfad von a nach b gibt ist die Wahrscheinlichkeit für Erfolg gleich 0
- ein Pfad existiert, also (G, a, b) ∈ UPATH → findet ihn unser Algorithmus mit der Wahrscheinlichkeit ≥ ½.

Verbesserung der Chance auf Erfolg können wir durch Wiederholungen erreichen (müssen nicht bei a anfangen, können Irrfahrt einfach fortsetzen). → Nach m*8en Schritten erreichen wir eine Erfolgswahrscheinlichkeit von mindestens $1 – 2^{-m}$, welche eine praktische Nutzung möglich macht.

Literaturangabe: Perlen der theoretischen Informatik (Uwe Schöning)