K elérhetőség: Ki van a kis világodban

James Cheng
Nanyang Technological University, Singapore

Zechao Shang
The Chinese University of Hong Kong

Hong Cheng
The Chinese University of Hong Kong

Haixun Wang
Microsoft research Asia Beijing, China

Jeffrey Xu Yu
The Chinese University of Hong Kong

Összefoglaló

Ebben a cikkben a **k-lépésbeni elérhetőségi problémát** vizsgáljuk irányított gráfokban, tehát azt, hogy létezik-e egy k hosszú irányított út egy kezdőcsúcsból egy célcsúcsba egy adott gráfban. A k-lépésbeni elérhetőség problémája az általánosítása a klasszikus elérhetőségi problémának, ahol $k=\infty$. A klasszikus problémákra létező megoldások nem alkalmazhatóak, vagy nem hatékonyak ahhoz, hogy a k-lépésbeni elérhetőség problémáját megoldják. Ebben a cikkben egy olyan indexelési eljárást készítünk, amely mindkét a feladatra képes, egyszerű átlátni és hatékony elkészíteni. A teszteredmények, amelyeket sok valós adatbázison teszteltünk le, megmutatják, hogy ez az eljárás a klasszikus elérhetőségi probléma megválaszolásában hatékonyabb, mint a jelenleg legjobbként számontartott és használt indexelési eljárások. Azt is megmutatjuk, hogy ez az indexelési eljárás hatékonyan válaszolja meg a k-lépésbeni elérhetőségi kérdést is.

Bevezetés

Az elérhetőségi probléma - ami azt kérdezi meg, hogy el lehet-e érni egy csúcsból egy másikat - egy gyakori feladat az adatbázisok egyes fajtáinál (pl.: XML,RDF) és hálózati alkalmazásoknál (például szociális hálózatok). Elég sok kiterjesztése is akad a feladatnak; például ugyanezt a feladatot bizonytalansági gráfokon elvégezni, ahol az élek létezésének csak egy valószínűsége van.

Ebben a cikkben egy újféle megközelítéssel próbáljuk megoldani a problémát. Ahelyett, hogy azt kérdeznénk meg, elérhető-e egy t csúcs s-ből, azt kérdezzük meg, hogy t elérhető-e k lépés alatt s-ből. Vagyis azt kérdezzük meg, hogy *létezik-e út s-ből t-be úgy, hogy az út hossza nem hosszabb mint k.* Ezt a problémáat k-lépésbeni elérhetőségi problémának hívjuk. A fő motiváció a kidolgozáskor az volt, hogy nagyon sok valós hálózatban (pl.: vezetéknélküli hálózat, internet, kommunikácós hálózatok, szociális hálózatok stb.) az ugrások száma ami alatt s eléri t-t jelzi , hogy s-nek mekkora hatása van t-re. Az alkalmazások ezeken a hálózatokon sokkal nagyobb hasznát látják a k-elérési, mint a klasszikus elérési problémának($k=\infty$). Ezt néhány példával illusztráljuk.

A vezetéknélküli hálózatokban, ahol a szórt üzenet elveszhet bármelyik két közvetítő pont között, annak esélye hogy megkapjuk az üzenetet, minden köztes lépéssel csökken. Ezeknél az alkalmazásoknál a klasszikus elérhetőség nem lenne túl hasznos információ, de a k-lépésbeni elérhetőség, mivel tudja modellezni az üzenetszórási terület méretét, hasznos lehet sok feladatban.

Sok valós hálózatban a k-lépésbeni elérhetőség két pont között sokkal érdekesebb kis k értékekre. Szociális hálózatokban, bár létezik a jól ismert 6-ismerős-távolságra-vagyok-mindenkitől (bármelyik két ember hat vagy kevesebb éltávolságnyia van egymástól), a tényleges ismeretség két ember között drasztikusan csökken (pl.: két ember nagy valószínűséggel nem ismeri egymást, ha már három éltávolságnyira vannak egymástól).

Elég világos, hogy sokkal jobban érdekel minket az, hogy két ember vajon egy-két éltávolságnyira van-e egymástól, mint 6 élnyire, ahol már majdnem mindig elérik egymást. Persze, a kis k választása nem feltétlenül teszi a feladatot könnyebbé.

Vegyük azt a feladatot, hogy meg akarjuk tudni, két ember között van-e legfeljebb 6 él hosszú kapcsolat. A naív implementációja ennek a problémának egy szélességi bejárás lenne a gráfban. De valós adatoknál a legtöbbjük legalább egy híres emberrel kapcsolatban lesz valamelyik szintjén a bejárásnak, (például Lady Gagával, akinek 40,000,000 rajongója van Facebookon). Egy ilyen híresség a legnagyobb részét eléri három éltávolság alatt az egész szociális hálózatnak, így annak megválaszolására, hogy két ember milyen távol van egymástól, elég nyilvánvalóan alkalmatlan csomópont.

A k-lépésbeli elérhetőség problémája nem a klasszikus elérhetőségi problémából adódik. Éppen fordítva; a klasszikus egy speciális esete a k-lépésbelinek, ahol $k=\infty$. És valóban, a k-lépésbeli probléma nagyobb feladat, mivel több információ szükséges a megválaszolásához. Ennek belátásához vegyük egy irányított gráf szomszédsági mátrixának tranzitív lezártját. Ha van egy ilyen mátrixunk, azonnal meg tudjuk mondani hogy s eléri-e t-t, úgy, hogy megnézzük, hogy a hozzájuk tartozó érték a mátrixban 1 vagy 0. A gond az, hogy nagy gráfoknál nagyon költséges kiszámítani és tárolni a tranzitív lezártat, mivel $O(n^2)$ helyet igényel, ahol n a gráf csúcsainak a száma. Ezért aztán az elérhetőségi probléma igazából az a feladat, hogy hogyan kódoljuk el hatékonyan ezt a 0-1 tranzitív mátrixot egy kicsi index struktúrába amiben még mindig hatékonyan tudunk keresni bármelyik két csúcs között. A k-lépésbeni elérhetőséghez viszont ez a mátrix már nem 0-1-esekből fog állni. Ehelyett minden csúcspárhoz tartozó érték a legrövidebb út hosszát fogja tárolni az adott két csúcs között. Nyilvánvaló, hogy ez a mátrix jóval több információt tárol, mint a 0-1 mátrix.

Ebben a cikkben egy ilyen hatékony indexet mutatunk be, amivel k-lépésbeni elérhérhetőséget kérdezhetünk le. Ezt az indexet **k-elérhetőségi index**nek hívjuk. A k-elérhetőségi index a lefedő csúcshalmaz elméletén alapszik. *Az index kialakításához azt a tényt használtuk fel, hogy egy gráfban minden csúcs közvetlenül elérhető a gráfot lefedő csúcshalmaz valamely pontjából.* A lefedő csúcshalmaz a gárfhoz képest kicsi a legtöbb valóságban használt gráfnál. Így csak az eredeti csúcsok egy kis halmazán kell előállítanunk. Azt is megmutatjuk, hogy legfeljebb 2 bitre van szükségünk az egyes lépésekhez tartozó információk eltárolásához.

Egy másik előnye a k-elérhetőségi indexnek, hogy lehetővé teszi a magas-(ki és be)fokú csúcsokat a lefedő csúcshalmazban. Ez nem csak a lefedő csúcshalmaz méretét csökkenti (és ezáltal az index méretét), hanem lehetővé teszi azoknak a lekérdezéseknek a hatékonyabb megválaszolását is amiben magasfokú csúcsok szerepelnek. (mint a fentebb említett Lady Gaga példa). A k-elérhetőségi index egyszerű és könnyű implementálni. Segítségével lehet keresni klasszikus és k-lépésbeni elérhetőséget is. A kísérletek 15 valós adatbázison lettek elvégezve, és az eredmények azt mutatják, hogy a k-elérhetőségi index jelentősen gyorsabb azoknál jelenlegi legjobbnak elismert algoritmusoknál, amiket kifejezetten a klasszikus elérési problémára terveztek. Azt is megmutatjuk, hogy nagyságrendekkel gyorsabb a szélességi bejárásnál vagy a legrövidebb út keresésénél, amelyeket a k-lépésbeni probléma megválaszolására alkalmazhatunk. Ezenkívül, az algoritmus teljesítménye stabil kis és nagyméretű k-ra egyaránt. Az eredmények azt mutatják, hogy a k-elérhetőségi indexet hatékonyan elő lehet állítani, és kis kis helyen tárolni.

Alapfogalmak

Table 1: Frequently used notations

Notation	Description
$G = (V, E)$	A directed, unweighted graph
n or m	The number of vertices or edges in G
$s \to t$	t is reachable from s
$s \to_k t$	t is reachable from s within k hops
$inNei(v, G)$	the set of in-neighbors of v in G
$inDeg(v, G)$	the in-degree of v in G, i.e., $\|inNei(v, G)\|$
$outNei(v, G)$	the set of out-neighbors of v in G
$outDeg(v, G)$	the out-degree of v in G, i.e., $\|outNei(v, G)\|$
$Nei(v, G)$	the set of neighbors of v in G
$Deg(v, G)$	the out-degree of v in G, i.e., $\|Nei(v, G)\|$

A fenti tábla a gyakran használt jelöléseket tartalmazza. Ezeket a jelöléseket használjuk a cikkben.

Legyen $G=(V,E)$ egy súlyozatlan irányított gráf, ahol V a csúcsok, E pedig az élek halmaza G-ben. Egy $(u,v)\in E$ egy u-ból v-be menő irányított él, $(v,u)\in E$ pedig egy v-ből u-ba menő irányított él.

Legyen egy s,t csúcspárunk G-ben. Azt mondjuk, hogy t elérhető s-ből, azaz $s\to t$, ha létezik egy egyszerű irányított út $P=\langle s,\dots,t\rangle$ G-ben. Ha $|P|\le k$, ahol $|P|$ az út hossza (azaz az élek száma az útban), akkor t k-lépésben elérhető s-ből, azaz $s\to_k t$. Egy elérhetőségi probléma annak eldöntése, hogy $s\to t$, és egy k lépésbeni elérhetőségi probléma annak eldöntése, hogy $s\to_k t$ .Vegyük észre, hogy itt $s\to t$ és $s\to_\infty t$ ugyanazt jelenti.

A k-lépésbeni indexelés:

Legyen G egy súlyozatlan, irányított gráf. A cikk megmutat egy index struktúrát G-re, ami válaszol a k-lépésbeni elérhetőségi kérdésre.

Legyen $G=(V,E)$, $n=|V|, m=|E|$.
Hívjuk egy $v\in G$ csúcs **be-szomszéd**jainak azt a halmazt, ami $inNei(v,G)=\{u|(u,v)\in E\}$, és hívjuk egy $v\in G$ csúcsnak a **be-fok**át $inDeg(v,G)=|inNei(v,G)|$.
Hasonlóképpen, hívjuk egy $v\in G$ csúcs **ki-szomszéd**jainak azt a halmazt, ami $outNei(v,G)=\{u|(v,u)\in E\}$, és egy $v\in G$ csúcs **ki-fok**át $outDeg(v,G)=|outNei(v,G)|$. A ki és be-szomszédok együttesét hívjuk **szomszéd**oknak, valamint a ki és be-fokok együttesét **fok**oknak.

Elérhetőség vs. k-lépésbeni elérhetőség

Ebben részben azt vizsgáljuk meg, hogy a különböző jelenleg létező elérhetőségi indexek hogyan alkalmazhatóak k-lépésbeni elérhetőségei lekérdezések feldolgozására. Kategorizáljuk a létező elérhetőségi indexeket hat különböző kategóriába, aztán megmutatjuk, hogy ezeket miért nem lehet alkalmazni, vagy hatékonyságuk miért nem megfelelő a k-lépésbeni kérdések megválaszolásához. Megjegyezzük, hogy a már létező megoldások közül néhány több mint egy kategóriába is besorolható, mivel több különböző megoldást is kombinálnak a probléma megoldására.

Kapcsolódó munkák

Irányított Körmentes Gráf alapú megközelítés

Az első kategóriája az indexeknek az irányított körmentes gráf (directed acyclic graph - DAG) elméletén alapszik. Sok indexkészítő eljárás él azzal a feltételezéssel, hogy a bejövő gráf DAG, mivel ha nem az, át lehet alakítani DAG-ra a következőképp. Először is ki kell számolni az összes szorosan kapcsolódó komponensét a bejövő gráfnak. Ezután átalakítunk minden ilyen komponenst egy szuper-csúccsá, ahol mindegyik szupercsúcs egy csúcs a DAG-ban. Végül, egy irányított élt (c1,c2) hozzáadunk a DAG-hoz akkor és csak akkor ha létezik egy ugyanilyen (u,v) él az eredeti gráfban úgy, hogy u a c1-ben és v a c2-ben van.

Összetömöríteni egy általános gráfot egy DAG gráffá helyet takaríthat meg és alkalmas az elérhetőségi problémák megválaszolására, mivel minden csúcs ami a gráf egy erősen összefüggő komponensében van, kölcsönösen elérhetőek egymásból. Viszont a k-lépésbeni keresésekben ez a DAG módszer elbukik, mivel két k-lépésben-elérhető csúcs a DAG-ban lehetséges hogy nem k-lépésben-elérhető az eredeti gráfban, mivel az őket összekötő legrövidebb út átalakulhatott egy rövidebb úttá a DAG-ban. Ahhoz, hogy ezzel a módszerrel válaszolhassuk meg a kérdést, szét kell szedni a DAG érintett csúcsait és megvizsgálni bennük az eredeti gráfot, hogy a pontos lépésszámot megkaphassuk, ami végeredményben nem olcsóbb művelet mintha az eredeti gráfban keresnénk meg ugyanezt.

Bejárás-alapú csúcs kódolás megközelítés

A második kategóriába azok az algoritmusok tartoznak, amik valamilyen gráfbejárási módszerrel alakítanak ki csúcs elkódolásokat. Egy bejárás a bejárás szerinti sorrendben kódokat rendel a gráf egyes csúcsaihoz (például mélységi bejárásnál a felfedezési és a feldolgozás befejezésének idejét). A kódpárok így egy intervallumot alakítanak ki, amit aztán tovább lehet módosítani ahhoz, hogy több információt szerezzünk az egyes csúcsok környezetéről. Az elérhetőségi kérdés eldönthető azzal, hogy a csúcsok kapcsolatban vannak-e egymással az intervallumon. Többféle bejárás is alkalmazható, ezzel több információt adva az egyes csúcsokhoz.

A k-lépésbeni kérések megválaszolására azonban a csúcsok kódjai nem tartalmaznak információt. Ahhoz, hogy ezt meg tudjuk válaszolni, be kell járnunk a gráfot a forrástól a célcsúcsokig. Ezt a bejárást a szerzett információk segíthetik, de a bejárás olyan lassú eredményeket is produkálhat mint egy egyszerű mélységi bejárás.

Más Megközelítések

Más megközelítések is léteznek ezeken kívül, például a tranzitív szomszédsági mátrix kibővíthető a k-lépésbeni elérhetőség információival. Ez a megoldás sajnos túl nagy adathalmazt generálna, ami miatt felhasználása nem praktikus. Egy nemrég publikált cikk bemutatott egy megoldást ami egy ilyen mátrixot alkalmazott bitvektoros tömörítési technikával[28], ami hatékonynak bizonyult elérhetőségi lekérdezésekre. A baj az, ez a megoldást nem lehet kibővíteni hatékonyan k-lépésbeni elérhetőségekre. Ehhez a mátrixban 1 és 0 bitek helyett távolságokat kellene eltárolni, ami elrontja a bitvektoros tömörítési technikát, ami csak 0-1 sorozatokra alkalmazható hatékonyan. Valamint mind a tranzitív lezárt mátrix, mind a tömörítése a bejövő gráf jóval kisebb DAG-ján dolgzik, ami nem alkalmas a k-lépésbeni elérhetőség megválaszolására, ahogy azt a 3.1-es leírásban taglaltuk.

Csúcs-lefedés alapú index

Most, hogy megbeszéltük a problémáit és határait a jelenlegi k-lépésbeni elérhetőséget megválaszoló indexeknek, ebben a részben megmutatunk egy hatékony eljárást, a **k-elérhetőségi index**et, mint megoldást.

K-elérhetőségi index : Index építés

A k-elérhetőségi index a csúcs lefedés elméletén alapszik [18]. Először megbeszéljük, hogy hogyan számoljuk ki egy kis csúcslefedési halmazát a G gráfnak. Aztután definiáljuk az index struktúrát és leírjuk az algoritmust, ami elkészíti ezt az indexet.

Legkisebb lefedő csúcshalmaz közelítés

Csúcsok egy halmaza, S egy lefedő csúcshalmaza $G=(V,E)$ gráfnak, ha minden $(u,v)\in E$ élre létezik $(\{u,v\}\cap S)\neq 0$. Értelemszerűen, V magában is egy lefedő csúcshalmaza G gráfnak, de túl nagy ahhoz hogy a nekünk kellő indexeket előállítsuk. Ezért meg kell keresnünk a legkisebb ilyen csúcshalmazt.

Egy S akkor minimális lefedő csúcshalmaza G-nek, ha S a legkisebb méretű lefedő csúcshalmaz G összes ilyen halmaza közül. Ennek kiszámítása egy ismert NP-nehéz probléma[18]. Létezik viszont egy polinomiális időben végrehajtható közelítő algoritmus, ami a következőképpen működik.

Véletlenszerűen kiválasztunk egy $(u,v)\in E$ élet, u és v csúcsot is beletesszük S-be, majd kitöröljük őket G-ből az összes szomszédos élükkel együtt. Mind a ki- mind a bemenő éleket eltávolíthatjuk, mivel minden él le van fedve vagy u vagy v-vel S-ből. Ezt addig ismételjük, amíg minden él el nem tűnik G-ből.

Ez az algoritmus $O(m+n)$ időigényű, mivel minden élet csak egyszer kell érintenünk. Legyen C a minimum lefedő csúcshalmaza G-nek. Ekkor, minden csúcspárra amit kiválasztottunk a fentebbi algoritmus során, vagy v-nek vagy u-nak benne kell lennie C-ben, máskülönben az (u,v) élt nem fedné le semmilyen csúcs C-ben.Így aztán a kapott S lefedő csúcshalmazra igaz, hogy $|S|\leq 2*|C|$.

Az analízisből látszik, hogy ennél a közelítő algoritmusnál egyszerűen mellőzhetjük az élek irányát.

Ezt *2-közelítő minimum csúcslefedő halmaz*nak hívjuk.

Algorithm 1 *Construction of k-reach*

Input: a directed graph $G = (V, E)$ and an integer k
Output: a k-reach index of G

1. Compute a 2-approximate minimum vertex cover, S, of G;
2. Initialize a weighted, directed graph $I = (V_I, E_I, \omega_I)$;
3. $V_I \leftarrow S$;
4. **for each** $u \in S$ **do**
5. Compute $S_k(u) = \{v : v \in S, u \rightarrow_k v\}$ by a k-hop BFS from u;
6. **for each** $v \in S_k(u)$ **do**
7. $E_I \leftarrow (E_I \cup \{(u, v)\})$;
8. **if**$(u \rightarrow_{k-2} v)$
9. $\omega_I((u, v)) \leftarrow (k - 2)$;
10. **else if**$(u \rightarrow_{k-1} v)$
11. $\omega_I((u, v)) \leftarrow (k - 1)$;
12. **else** $/* (u \rightarrow_k v) */$
13. $\omega_I((u, v)) \leftarrow k$;
14. **return** $I = (V_I, E_I, \omega_I)$;

A k-elérhetőség definiciója és felépítése

Most már minden eszközünk megvan ahhoz, hogy felépítsük a k-elérhetőségi indexet a következőképpen.

Definició 1(k-elérhetőség) :

Legyen egy G = (V,E) irányított gráf, egy S lefedő csúcshalmaza G-nek, és egy k szám. Ekkor a k-elérhetőségi indexe G-nek az a súlyozott, irányított gráf $I=(V_I, E_I, w_I)$ *a következő:*

- $V_I=S$
- $E_I=\{(u,v)|u,v\in S, u\rightarrow_k v\}$
- w_I egy súlyfüggvény ami minden élhez egy $e=(u,v,)\in E_I$ súlyt rendel a következőképpen:

- ha $u\rightarrow_{k-2} v$, akkor $w_I(e)=k-2;$

- különben ha $u\rightarrow_{k-1} v$, akkor $w_I(e)=(k-1);$

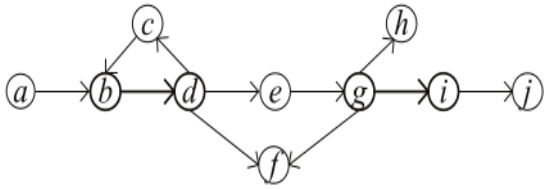
- különben ha $u\rightarrow_k v$, akkor $w_I(e)=k$.

Vegyük észre hogy az $u\rightarrow_{k-2} v$ -ból következik $u\rightarrow_{k-1} v$, és mindkettőből következik $u\rightarrow_k v$.

Most pedig leírjuk az index készítési eljárást..

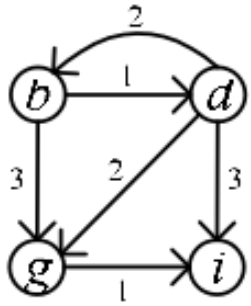
Az eljárás először kiszámítja G 2-közelítő lefedő csúcshalmazát, S-et a fentebb leírt algoritmussal. Ezután készít egy $I=(V_I, E_I, w_I)$ gráfot úgy, hogy szélességi bejárásban bejárja G gráfot k mélységig minden $u\in S$ csúcsban. Ez az eljárás kiszámítja minden S halmazbeli csúcs azon halmazát, amit el lehet érni u-ból k lépésben a G gráfban.

Példa 1. Legyen G gráfunk a következő:



Tegyük fel, hogy a lefedő csúcshalmaz algoritmusunk véletlenszerűen a (b,d) és (g,i) éleket választja. Ekkor a közelített minimum lefedő csúcshalmazunk {d,b,g,i}. Látszik, hogy ez valóban egy lefedő csúcshalmaza G-nek, mivel minden él G-ben szomszédos legalább egy csúccsal {b,d,g,i}-ből.

Legyen k = 3. A k-elérhetőségi gráf $I=(V_I, E_I, w_I)$ ekkor a következő:



Mivel k = 3, a lehetséges élsúlyok k-2 = 1, k-1 = 2 és k = 3. Például, $b \rightarrow_3 g$ G-ben, és így megvan egy irányított élünk (b,g) ahol $w_i((b,g))=3$.

A k-elérhetőségi index építési költsége

Kiszámolni a 2-közelítő lefedőcsúcshalmazt $O(n+m)$ időt vesz igénybe. Felépíteni egy súlyozott, irányított $I=(V_I, E_I, w_I)$ gráfot $O(\sum_{u \in S} |G_k(u)|)$ idő, ahol $G_k(u)$ az a részgráfja G-nek, amit be lehet járni u-ból k lépésben. Vegyük észre, hogy emiatt ezt az eljárást könnyen lehet párhuzamosítani, ha több számítógép vagy CPU mag áll rendelkezésünkre.

Az index mérete, azaz az I gráf mérete függ az S és az $S_k(u)$ minden $u \in S$ -től. Valós gráfokban nehéz elméleti határértéket mondani az S vagy az $S_k(u)$ értékeire, mert adatbázisonként nagyon eltérőek lehetnek. Néhány ilyen gráf hasonló karakterisztikájú és hasonló tulajdonságokkal rendelkezik, úgy mint ritkaság és az élszámok eloszlása a csúcsok között. Nem ismerünk olyan jelenleg létező munkát, ami valamilyen elméleti becslést adna a minimális csúcs lefedés halmaz méretére valós gráfoknál. Így az index méretét sokfajta valós gráf feldolgozásával próbáljuk kikísérletezni.

Végül, a k-elérhetőségi index O(m + n) helyet használ, amit aztán el is tárolunk a lemezen.*k*-elérés: lekérdezés feldolgozása

Az alábbi algoritmus leírja, hogyan használható a *k*-elérés index egy *k* ugrással elérhetőségre vonatkozó lekérdezés feldolgozására.

Algorithm 2 *Query processing using k-reach*

Input: a directed graph, $G = (V, E)$,
the k-reach index, $I = (V_I, E_I, \omega_I)$, of G,
and two query vertices, s and t
Output: a boolean indicator whether $s \rightarrow_k t$

```
/* Case 1: both s and t are in the vertex cover */
1. if(s ∈ V_I and t ∈ V_I)
2.     if((s,t) ∈ E_I)
3.         return true;
4.     else
5.         return false;

/* Case 2: only s is in the vertex cover */
6. else if(s ∈ V_I and t ∉ V_I)
7.     if(∃v ∈ inNei(t,G) such that
         (s,v) ∈ E_I and ω_I((s,v)) ≤ (k − 1))
8.         return true;
9.     else
10.        return false;

/* Case 3: only t is in the vertex cover */
11. else if(s ∉ V_I and t ∈ V_I)
12.    if(∃v ∈ outNei(s,G) such that
         (v,t) ∈ E_I and ω_I((v,t)) ≤ (k − 1))
13.        return true;
14.    else
15.        return false;

/* Case 4: both s and t are not in the vertex cover */
16. else if(s ∉ V_I and t ∉ V_I)
17.    if(∃u ∈ outNei(s,G) and ∃v ∈ inNei(t,G) such that
         (u,v) ∈ E_I and ω_I((u,v)) ≤ (k − 2))
18.        return true;
19.    else
20.        return false;
```

Adott két csúcs a gráfban: *s* és *t*. Az algoritmus eldönti, hogy létezik-e legfeljebb *k* hosszú út *s* és *t* között az eredeti *G* gráfban, és visszaad egy logikai értéket. A kérés feldolgozása során négy esetet különböztetünk meg:

1. Ha mind *s*, mind *t* benne van az *S* csúcslefedésben, mely egyben az *I* indexgráf V_I csúcshalmaza, akkor csupán azt kell vizsgálnunk, létezik-e él *I*-ben *s* és *t* között. Definíció szerint pontosan akkor szerepel *I*-ben az (*s*, *t*) él, ha *G*-ben található *k*-nál nem

hosszabb út a két csúcs között. Tehát az algoritmus akkor és csak akkor ad *igaz* választ, ha az (s, t) benne van E_I-ben.

2. Ha csak s eleme V_I-nek, t nem, akkor keressük az úton a t-t megelőző élt. A csúcslefedés definíciója alapján tudjuk, hogy t minden szomszédja benne van S-ben, elvégre minden él legalább egyik végpontja S-beli, t pedig esetünkben nem az. Vizsgáljuk t azon szomszédjait, amelyeknek t rákövetkezője. Olyan v csúcsot keresünk, amely s-ből (k–1) lépésben elérhető. Ez pontosan akkor teljesül, ha létezik az (s, v) él I-ben és annak súlya legfeljebb (k–1). Ha találtunk ilyen v csúcsot, akkor az algoritmus *igaz* válasszal tér vissza.

3. Ha csak t eleme V_I-nek, s nem, az a 2. ponthoz hasonló eset, szükségtelen külön tárgyalni.

4. Ha sem s, sem t nem eleme V_I-nek, akkor mindkettőnek keressük egy alkalmas szomszédját (s-nek rákövetkező u, t-nek megelőző v csúcsát), és ezek között vizsgáljuk a távolságot. Minden választott u, v S-beli. Ha van olyan u, v pár, melyre (u, v) egy él I-ben, a hozzá tartozó súly pedig (k–2)-nél nem nagyobb, akkor s és t között létezik legfeljebb k hosszú út, tehát az algoritmus *igaz* válasszal tér vissza; különben *hamis*-sal. (Ellenpélda: ha az (u, v) I-beli él súlya (k–1), akkor az egy (k–1)+2 = (k+1) hosszú utat garantál s és t között, amely nem válaszolja meg a k ugrással elérhetőségre vonatkozó lekérdezést.)

A k-elérés alapú lekérdezésfeldolgozás műveletigénye

Annak ellenőrzése, hogy s és t eleme-e V_I-nek, $O(1)$ műveletigényű. Hogy egy (u, v) él szerepel-e E_I-ben illetve milyen súllyal, $O(\log outDeg(u, I))$ vagy $O(\log inDeg(v, I))$ időben meghatározható, amennyiben I-t éllistásan ábrázoljuk. Ezért az algoritmus 1. esete $O(\log outDeg(s, I))$ műveletigényű, a 2. eset $O(outDeg(s, I) + inDeg(t, G))$ műveletigényű, a 3. eset $O(outDeg(s, G) + inDeg(t, I))$ műveletigényű, a 4. eset pedig $O(\Sigma_{u \in outNei(s, G)}(outDeg(u, I) + inDeg(t, G)))$ műveletigényű.
Megjegyzés: egy L éllista lemezről történő beolvasása igényel továbbá $O(|L|/B)$ I/O műveletet, ahol B egy lemezblokk mérete. A gyakorlatban ritka gráf lévén $|L|<B$, ezért az I/O költség jellemzően alacsony.

A magasfokú csúcsok átka

A fentebb leírt műveletigény elemzés alapján a lekérdezés sebessége nagyban függ a G és I beli csúcsok fokától. Sok valós gráfban megfigyelhető a hatványos eloszlás a fokoknál – azaz a csúcsok egy kis része nagyon magas fokszámmal rendelkezik. Például Lady Gaga énekesnőnek 40,000,000 rajongója van Facebookon. Ezért aztán kritikus probléma, hogy ezeket a csúcsokat feldolgozás közben elkerüljük mint lekérdezési csúcsokat, amik a 2, 3 vagy 4 esetbe tartoznak az előzőleg leírt algoritmusban. Statisztikailag, ezek a magas fokszámú csúcsok sokkal nagyobb eséllyel kerülnek kiválasztásra, mivel általában olyan objektumokat tartalmaznak, amik több figyelmet vonnak magukra. Ahhoz, hogy ezeket a csúcsokat hatékonyan tudjuk feldolgozni, módosítjuk a 2-közelítő lefedő csúcshalmaz algoritmusát a következőképpen. Amikor kiválasztunk egy élt, magasabb prioritást adunk neki attól függően, hogy az él két csúcsa milyen magas fokszámmal rendelkezik. Minél nagyobb fokszáma van

a csúcsoknak, annál magasabb prioritást kap az él. Mivel a legtöbb valós gráfnak csak egy kis százaléka magas-fokú csúcs [25], nyugodtan bevehetjük őket a lefedő csúcshalmazba anélkül, hogy elrontanánk vele az lefedő csúcshalmazt készítő algoritmust. Sőt, ez a megoldás egy mohó stratégiának felel meg, ami a gyakroltaban csökkenti a helyigényt, mivel a magas fokszámú csúcsokat részesíti előnyben. A gyakorlati vizsgálat azt mutatta ki, hogy egy tipikus valós gráfnál ahol ez a hatványozott fok eloszlás érvényesül, akkor ha a gráf 1 millió csúcsból áll, akkor a "h-indexe" a gráfnak mindeössze 300 [10,11] csúcs; tehát az 1 millió csúcsú gráf mindössze h = 300 csúcsot tartalmaz, aminek a foka legalább 300.

A csúcsok amiknek magas foka van G-ben, általában magas foka van I-ben is.Ez nem csak a sebességbeli csökkenést mutatja, hanem megnöveli az index méretét is. De ez a probléma eltüntethető a következőképpen. Mivel I-ben csak 3 féle élsúlyunk lehetséges, k , (k-1) és (k-2), csak 2 bitre van szükségünk, hogy ábrázoljuk az élek súlyát. Így a magasfokú csúcsok szomszédjait tartozó csúcshalmaz egy sokkal tömörebb formában is felírható, mint egy intervallum lista, vagy particionált szavak hibrid tömörítéssel rendezve[28], ami rendkívül hatékonynak bizonyult hogy lecsökkentsük a szükségelt helyigényét az indexnek. Ezekkel a tömörített reprezentációkkal csak az idetartozó biteket kell meghatároznunk a lekérdezéshez[28], ahelyett, hogy a szomszédok listájában végeznénk keresést.

Tesztek

A hatékonyságát úgy mérjük le a k-elérhetőségi indexnek, hogy összehasonlítjuk a jelenleg használatban lévő legjobb indexelési módszerekkel, amik a klasszikus elérhetőségi problémát oldják meg. Ezen algoritmusok a PTree[24],3-hop[23],GRAIL[32], és a PWAH[28]. Minden rendszer C++ban lett implementálva és ugyanazzal a gcc fordítóval fordítva. Minden tesztet egy Intel Quad Q9400 2.66 GHz CPU és 4GB RAM paraméterekkel rendelkező gépen futtattunk, amin Scientific Linux 6.0 operációs rendszer volt. A tesztek 10 alkalommal futottak és az eredmények konzisztensek voltak a 10 futtatás alatt.

Adathalmazok

A tesztjeinket 15 valós adatbázison futtattuk, amiknek használata népszerű a jelenlegi elérhetőségi indexek tesztelésében[23,24,28,32]. Az AgroCyc,Anthra,Ecco,Human,Mtbrv és Vchocyc adathalmazok ecocyc.org-ról származnak és a génállomány és biokémiai működését írják le az E. coli K-12 MG1655-nek. Az aMaz és Kegg anyagcsere adatait tartalmazó adatbázisok. Nasa és Xmark XML dokumentumok. Az ArXiv, CiteSeer, PubMed adathalmazok idézet hálózatok. A GO adathalmaz egy gén ontológia gráf. A YAGO egy gráf ami leírja a kapcsolati struktúráját a szemantikus tudásnak a YAGO adatbázisában.

A "Table 2." kép a csúcsok(V) és élek(E) számát mutatja, a legmagasabb fokú csúcs fokát(Deg_{max}), az átmérőt (d), és az átlag hosszát az összes legrövidebb útnak(μ) az adathalmazban. Ezen kívül az átlakaított DAG gráf élei és csúcsainak számát is, mivel az összes többi módszer az eredeti gráfokból átalakított DAG adatszerkezeten dolgozik.

A "Table 3" kép az egyes módszerek indexeinek elkészítési idejét mutatja, vastaggal jelölve azokat az módszereket, akik az adott valós adathalmazra a legjobb eredményt nyújtották.

A “Table 4” kép a létrehozott indexek helyigényét mutatják, vastaggal jelölve azon módszereket, amelyek a legkevesebb helyet igényeltek az adott adatbázis indexének eltárolásához.

Végül pedig a “Table 5” kép a lekérdezések hatékonyságát teszteli. Ehhez véletlenszerűen generáltunk 1 millió lekérdezést. Ezen eredményeknél ki kell emelni, hogy az adathalmazok nem úgy lettek választva, hogy előnyben részesítsék az ebben a cikkben publikált eljárást.

Az eredmény azt mutatja, hogy a lekérdezés feldolgozása a klasszikus elérhetőségi problémára jelentősen jobb mint az összes többi indexé. Átlagosan az n-elérhetőség 2.2-szer gyorsabb mint a Ptree lekérdezésben, de 7.9-szer gyorsabb az index megépítésében. A PWAH-nál átlagosan 3.2-szer gyorsabb, és valamivel lassabban építi fel az indexet. A GRAIL és a 3-hop módszerekkel összehasonlítva az n-elérhetőségi index egyértelműen viszi a prímet, mivel minden lekérdezést egy nagyságrendel gyorsabban elvégez.

Table 2: Datasets

	$\|V\|$	$\|E\|$	$\|V_{DAG}\|$	$\|E_{DAG}\|$	Deg_{max}	d	μ
AgroCyc	13,969	17,694	12,684	13,657	5,488	10	2
aMaze	11,877	28,700	3,710	3,947	3,097	11	2
Anthra	13,766	17,307	12,499	13,327	5,401	10	2
ArXiv	6,000	66,707	6,000	66,707	700	20	4
CiteSeer	10,720	44,258	10,720	44,258	192	18	3
Ecoo	13,800	17,308	12,620	13,575	5,435	10	2
GO	6,793	13,361	6,793	13,361	71	11	3
Human	40,051	43,879	38,811	39,816	28,571	10	2
Kegg	14,271	35,170	3,617	4,395	3,282	16	2
Mtbrv	10,697	13,922	9,602	10,438	4,005	12	2
Nasa	5,704	7,942	5,605	6,538	32	22	7
PubMed	9,000	40,028	9,000	40,028	432	11	4
Vchocyc	10,694	14,207	9,491	10,345	3,917	10	2
Xmark	6,483	7,654	6,080	7,051	887	24	5
YAGO	6,642	42,392	6,642	42,392	2,371	9	1

Table 3: Index construction time (elapsed time in milliseconds) of *n*-reach, PTree, 3-hop, GRAIL, and PWAH (shortest time shown in bold)

	n-reach	PTree	3-hop	GRAIL	PWAH
AgroCyc	27.71	129.14	-	10.86	**4.40**
aMaze	18.09	476.69	959,821	**2.92**	7.01
Anthra	24.08	123.43	-	10.74	**3.90**
ArXiv	352.51	6,319.66	-	**10.58**	111.00
CiteSeer	245.46	403.35	44,328	**16.04**	93.26
Ecoo	26.70	129.74	-	10.88	**4.47**
GO	106.84	110.83	11,914	**6.50**	19.57
Human	67.78	397.05	-	41.45	**6.71**
Kegg	21.01	537.17	-	**2.92**	6.77
Mtbrv	20.24	98.13	-	7.92	**3.86**
Nasa	57.93	62.22	13,739	**4.51**	10.54
PubMed	166.23	437.16	73,243	**11.63**	70.63
Vchocyc	19.77	97.34	-	7.60	**4.00**
Xmark	44.50	136.87	68,219	**4.96**	11.53
YAGO	32.47	282.45	5,006	**9.47**	36.49

Table 4: Index size (in MB) of *n*-reach, PTree, 3-hop, GRAIL, and PWAH (smallest size shown in bold)

	n-reach	PTree	3-hop	GRAIL	PWAH
AgroCyc	0.39	0.29	-	**0.19**	0.44
aMaze	0.13	0.09	5.41	**0.06**	0.22
Anthra	0.36	0.29	-	**0.19**	0.42
ArXiv	1.61	0.38	-	**0.09**	2.46
CiteSeer	3.17	0.45	0.20	**0.16**	3.08
Ecoo	0.40	0.29	-	**0.19**	0.43
GO	1.28	0.20	0.11	**0.10**	0.63
Human	1.17	0.89	-	**0.59**	1.25
Kegg	0.16	0.08	-	**0.06**	0.23
Mtbrv	0.29	0.22	-	**0.15**	0.34
Nasa	0.66	0.13	**0.06**	0.09	0.40
PubMed	2.03	0.50	0.29	**0.14**	2.80
Vchocyc	0.28	0.22	-	**0.14**	0.33
Xmark	0.49	0.13	0.43	**0.09**	0.45
YAGO	0.48	0.22	**0.09**	0.10	0.96

Table 5: Total running time (elapsed time in milliseconds) of n-reach, PTree, 3-hop, GRAIL, and PWAH, for processing 1 million randomly generated queries (shortest time shown in bold)

	n-reach	PTree	3-hop	GRAIL	PWAH
AgroCyc	**5.50**	17.74	-	135.14	15.68
aMaze	**14.39**	20.68	28404.20	2982.61	39.71
Anthra	**5.39**	17.66	-	121.12	14.92
ArXiv	87.86	**75.28**	-	2032.96	311.55
CiteSeer	115.64	**58.28**	1225.25	268.33	339.23
Ecoo	**5.47**	17.73	-	154.41	15.77
GO	**27.00**	35.77	455.83	113.46	59.10
Human	**5.95**	28.48	-	300.23	13.35
Kegg	**16.27**	22.51	-	4030.89	44.52
Mtbrv	**5.47**	17.48	-	104.15	16.12
Nasa	**18.26**	23.62	359.16	64.27	43.94
PubMed	**39.31**	103.44	1198.70	239.40	368.44
Vchocyc	**5.49**	17.72	-	103.23	16.13
Xmark	**14.49**	22.02	491.44	245.11	69.78
YAGO	106.25	**42.32**	705.09	116.43	137.09

Eredmények

Ez a papír a következő eredményekre jutott:

- Tudomásunk szerint mi vizsgáltuk meg először a **k-lépésbeni elérhetőség** problémáját.
- Kidolgoztunk egy hatékony indexelési eljárást, a k-lépésbeni indexelési eljárást arra, hogy feldolgozzunk k lépésbeni elérhetőségi kérdéseket. Ez az eljárás egyszerűen érthető és könnyű implementálni
- A k-elérhetőségi index a klasszikus elérhetőségi problémát is meg tudja válaszolni, hatékonyabban, mint az eddigi módszerek.

Tovább Kutatási terv

További munkaként tervezzük a hatékony indexelési eljárások vizsgálatát nagyon nagy gráfokra, amire a jelenlegi eljárás nem skálázódik megfelelően, és túl nagy indexet ad. I/O-hatékony eljárások és párhuzamosítás szükséges lehet, hogy ezekre a nagy gráfokra is alkalmazni tudjuk az eljárást, úgy mint az indexekhez a legrövidebb utak rendelése [8].

Felhasznált Irodalom

[1] I. Abraham, A. Fiat, A. V. Goldberg, and R. F. F. Werneck.

Highway dimension, shortest paths, and provably efficient

algorithms. In SODA, pages 782–793, 2010.

[2] R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient

management of transitive relationships in large data and

knowledge bases. In SIGMOD Conference, pages 253–262,

1989.

[3] R. Bramandia, B. Choi, and W. K. Ng. On incremental

maintenance of 2-hop labeling of graphs. In WWW, pages

845–854, 2008.

[4] B. Bresar, F. Kardos, J. Katrenic, and G. Semanisin.

Minimum k-path vertex cover. Discrete Applied

Mathematics, 159(12):1189–1195, 2011.

[5] L. Chen, A. Gupta, and M. E. Kurul. Stack-based algorithms

for pattern matching on DAGs. In VLDB, pages 493–504,

2005.

[6] Y. Chen and Y. Chen. An efficient algorithm for answering

graph reachability queries. In ICDE, pages 893–902, 2008.

[7] Y. Chen and Y. Chen. Decomposing DAGs into spanning

trees: A new way to compress transitive closures. In ICDE,

pages 1007–1018, 2011.

[8] J. Cheng, Y. Ke, S. Chu, and C. Cheng. Efficient processing

of distance queries in large graphs: a vertex cover approach.

In SIGMOD Conference, pages 457–468, 2012.

[9] J. Cheng, Y. Ke, A. W.-C. Fu, and J. X. Yu. Fast graph query

processing with a low-cost index. VLDB Journal,

20(4):521–539, 2011.

[10] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding

maximal cliques in massive networks by h*-graph. In

SIGMOD Conference, pages 447–458, 2010.

[11] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding

maximal cliques in massive networks. ACM Trans. Database

Syst., 36(4):21:1–21:34, 2011.

[12] J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: towards

verification-free query processing on graph databases. In

SIGMOD Conference, pages 857–872, 2007.

[13] J. Cheng and J. X. Yu. On-line exact shortest distance query

processing. In EDBT, pages 481–492, 2009.

[14] J. Cheng, J. X. Yu, X. Lin, H. Wang, and P. S. Yu. Fast

computation of reachability labeling for large graphs. In

EDBT, pages 961–979, 2006.

[15] J. Cheng, J. X. Yu, X. Lin, H. Wang, and P. S. Yu. Fast

computing reachability labelings for large graphs with high

compression rate. In EDBT, pages 193–204, 2008.

[16] J. Cheng, J. X. Yu, and N. Tang. Fast reachability query

processing. In DASFAA, pages 674–688, 2006.

[17] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick.

Reachability and distance queries via 2-hop labels. In SODA,

pages 937–946, 2002.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein.

Introduction to Algorithms, Second Edition. The MIT Press

and McGraw-Hill Book Company, 2001.

[19] H. V. Jagadish. A compression technique to materialize

transitive closure. ACM Trans. Database Syst.,

15(4):558–598, 1990.

[20] R. Jin, H. Hong, H. Wang, N. Ruan, and Y. Xiang.

Computing label-constraint reachability in graph databases.

In SIGMOD Conference, pages 123–134, 2010.

[21] R. Jin, L. Liu, B. Ding, and H. Wang. Distance-constraint

reachability computation in uncertain graphs. PVLDB,

4(9):551–562, 2011.

[22] R. Jin, N. Ruan, S. Dey, and J. X. Yu. Scarab: scaling

reachability computation on large graphs. In SIGMOD

Conference, pages 169–180, 2012.

[23] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. 3-hop: a

high-compression indexing scheme for reachability query. In

SIGMOD Conference, pages 813–826, 2009.

[24] R. Jin, Y. Xiang, N. Ruan, and H. Wang. Efficiently

answering reachability queries on very large directed graphs.

In SIGMOD Conference, pages 595–608, 2008.

[25] M. E. J. Newman. The structure and function of complex

networks. SIAM Review, 45(2):167–256, 2003.

[26] K. Simon. An improved algorithm for transitive closure on

acyclic digraphs. Theor. Comput. Sci., 58(1-3):325–346,

1988.

[27] S. Trißl and U. Leser. Fast and practical

indexing and

querying of very large graphs. In SIGMOD Conference,

pages 845–856, 2007.

[28] S. J. van Schaik and O. de Moor. A memory efficient

reachability data structure through bit vector compression. In

SIGMOD Conference, pages 913–924, 2011.

[29] H. Wang, H. He, J. Yang, P. S. Yu, and J. X. Yu. Dual

labeling: Answering graph reachability queries in constant

time. In ICDE, page 75, 2006.

[30] F. Wei. TEDI: efficient shortest path query answering on

graphs. In SIGMOD Conference, pages 99–110, 2010.

[31] Y. Xiao, W. Wu, J. Pei, W. Wang, and Z. He. Efficiently

indexing shortest paths by exploiting symmetry in graphs. In

EDBT, pages 493–504, 2009.

[32] H. Yildirim, V. Chaoji, and M. J. Zaki. Grail: Scalable

reachability index for large graphs. PVLDB, 3(1):276–284,

2010.

[33] J. X. Yu and J. Cheng. Graph reachability queries: A survey.

In Managing and Mining Graph Data, pages 181–215. 2010.

[34] L. Zhu, B. Choi, B. He, J. X. Yu, and W. K. Ng. A uniform

framework for ad-hoc indexes to answer reachability queries

on large graphs. In DASFAA, pages 138–152, 2009.