Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Számítástudományi és Információelméleti Tanszék

ÚTVÁLASZTÁS HÁLÓZATI HIBA ESETÉN

Berghammer Tamás
II. évfolyam

Konzulens:
Friedl Katalin, SZIT

2012

Tartalom

1 Bevezetés

Csomagkapcsolt hálózatoknál (mint például az internet vagy a modern telefonrendszerek) a logikai címekkel ellátott csomagok útját a forrástól a célig az útválasztás alapján határozzák meg. Az útválasztás általában a csomópontokban eltárolt, lokális útválasztó táblák segítségével történik úgy, hogy e táblák segítségével kerül meghatározásra az, hogy az egyes csomópontokban melyik célpontokba tartó csomagokat melyik élen (porton) kell továbbítani.

Az útválasztás megoldására a triviális (naiv) algoritmuson kívül – amelyik minden pontban eltárolja minden címhez a legrövidebb úthoz tartozó portot – több különböző egyéb útválasztó séma is létezik. Ezek abban hasonlítanak egymásra, hogy a lokális tárigény csökkentése érdekében nem feltétlenül a legrövidebb úton juttatják el a csomagot a forrástól a célig, hanem valamilyen multiplikatív hibával (a használt út legfeljebb k-szor hosszabb, mint a legrövidebb út). Valamennyi fenti útválasztó séma közös tulajdonsága, hogy a gráf szerkezeti felépítésére érzékenyek. Azaz, ha a gráfban bekövetkezik valamilyen változás, mint például meghibásodik egy él – ami a valóságban egy elég gyakori esemény – akkor nem képesek csomag továbbítására tetszőlegesen kiválasztott két pont között még abban az esetben sem, ha a gráf továbbra is összefüggő maradt.

A naiv sémára és egy viszonylag egyszerű, de sok bonyolultabb séma alapjaként szolgáló sémára kidolgoztam, hogy milyen módosítások szükségesek ahhoz, hogy az útválasztó séma képes legyen működni akkor is, ha a gráfban legfeljebb egy él meghibásodott. A módosított sémával szembeni legfőbb követelmény, hogy amennyiben egyetlen él sem hibásodott meg a rendszerben, akkor a hatékonyság ne változzon meg a kiindulási sémához képest. Továbbá sem a szükséges lokális tárigénynek, sem a csomagok címének hossza nagyságrendjében ne változzon meg. Ezen felül elvárás még, hogy a séma működése legyen determinisztikus, azaz mindig ugyanazon az úton továbbítsa a csomagot egy adott él esetén, és a csomag azon az úton haladva véges sok lépésben (lehetőleg minél rövidebb úton) jusson el a célpontjáig.

A dolgozat 2. fejezetében definiálom a használt jelöléseket és az alap feltevéseket. A 3. fejezetben bemutatom a helyettesítő utak megkeresésére létező algoritmusokat. Ezek után a 4. és az 5. fejezetben bemutatok két útválasztó sémát, amelyekre a 6. és a 7. fejezetben bemutatom a dolgozat fő eredményét, a helyettesítő utak kezelésére általam kidolgozott útválasztó sémát. Ezek után a 8. fejezetben összegzem az elért eredményeket, majd a 9. fejezetben bemutatok néhány szimulációs eredményt a kidolgozott sémák működésének szemléltetése céljából.

2 Használt jelölések és alapfeltevések

A dolgozatban súlyozott, pozitív élsúlyú, irányítatlan gráfokat fogok vizsgálni, amennyiben az adott részben azt máshogy nem specifikálom. Ezen kívül minden algoritmusnál fel fogom tenni, hogy a gráfban szereplő bármely két út hossza különböző. Ez az általánosság megszorítása nélkül feltehető, mivel az élsúlyok nagyon kismértékű megváltoztatásával elérhető ez a jelenség. (Például úgy, ha a legkisebb élsúlyt egységnek tekintve az i. élnek a súlyához hozzáadunk $\frac{1}{2^i}$-t)

A dolgozatban az alábbi jelöléseket fogom használni további magyarázat nélkül:

G:	Aktuálisan vizsgált gráf (súlyozott, irányítatlan)
E:	Az aktuálisan vizsgált gráf élhalmaza
V:	Az aktuálisan vizsgált gráf csúcshalmaza
n:	A gráfban lévő pontok száma (|V|)
m:	A gráfban lévő élek száma (|E|)
(u, v):	Az u és v pont közötti él
c(u, v):	Az u és v pont közötti él súlya (pozitív)
P(u, v):	Az u és v pontok közötti legrövidebb út az aktuálisan vizsgált gráfban
P(u, v, G):	Az u és v pontok közötti legrövidebb út az aktuálisan vizsgált gráf G részgráfjában
d(u, v):	Az u és v pontok közötti legrövidebb út hossza az aktuálisan vizsgált gráfban
d(u, v, G):	Az u és v pontok közötti legrövidebb út hossza az aktuálisan vizsgált gráf G részgráfjában
e_u(v):	Az első él az u-ból v-be menő legrövidebb út mentén

3 Helyettesítő út keresése

Egy gráfban két tetszőleges pont közötti legrövidebb út mindig meghatározható O($m + nlogn$) futásidőben a Dijkstra-algoritmus segítségével. Amennyiben viszont ennek a legrövidebb útnak valamelyik éle meghibásodik, akkor attól függően, hogy melyik él hibásodott meg, más-más út lesz a legrövidebb út a vizsgált két pont között.

DEFINÍCIÓ:

> Két pont közötti legrövidebb út adott éléhez tartozó helyettesítő út az a legrövidebb út, amelyik az eredeti út két végpontja között halad, és elkerüli a megadott élt. ▮

Egy adott út összes éléhez tartozó helyettesítő út legegyszerűbben a naiv algoritmus segítségével található meg. A naiv algoritmus úgy működik, hogy egyenként eltávolítja a legrövidebb út éleit a gráfból, majd a módosított gráfban mindig lefuttat egy Dijkstra-algoritmust, és az ezek által az algoritmusok által megtalált legrövidebb utakat tekinti a helyettesítő utaknak. Mivel a legrövidebb úton legfeljebb O(n) darab él lehet és egy Dijkstra-algoritmus futásideje O($m + nlogn$) ezért ennek az algoritmusnak a futásideje O($nm + n^2logn$), ami meglehetősen lassú.

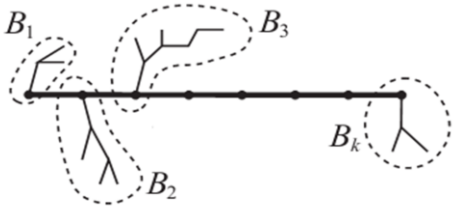
Az útválasztó sémáknál a helyettesítő utat egy kicsit másképpen kell definiálni. Ott a hibás élt elkerülő utak közül akarjuk a legrövidebbet megtalálni úgy, hogy a helyettesítő út kiindulópontja a hibás élnek a legrövidebb úton hamarabb elhelyezkedő végpontja legyen.

3.1 Helyettesítő út keresése irányítatlan gráfokban

Irányítatlan gráfokban két pont közötti legrövidebb út összes éléhez tartozó helyettesítő út megtalálható O($m + nlogn$) futásidőben. Ehhez a feladathoz a J. Hershberg és S. Suri [1][2] által kidolgozott algoritmust fogom használni. Az algoritmus kidolgozását a Vickrey pricing (a kialakuló tisztességes ár olyan aukció esetén, ahol a résztvevő felek egymás ajánlatainak az ismerete nélkül határozzák meg a saját ajánlatukat [3]), hálózatokra történő meghatározása motiválta, amihez a legrövidebb helyettesítő utak meghatározása csak a gráfelméleti eszköz volt. A Vickrey pricing meghatározása azért egy érdekes kérdés, mivel a hálózatokat általában egymástól független ügynökök birtokolják, ahol az egyes élek értéke jól közelíthető azzal, hogy mennyivel tudja az az él lerövidíteni a két adott pont közötti legrövidebb utat.

J. Hershberger és S. Suri algoritmusa úgy működik, hogy meghatározza a kiindulási pontból indított, a legrövidebb utakat tartalmazó fát egy Dijkstra-algoritmus segítségével. Ekkor, ha a legrövidebb út valamelyik élét elhagyjuk, akkor ez a fa két részfára esik szét. A két részfa között lesz egy vágás. Az elhagyott élhez tartozó helyettesítő útról bebizonyították, hogy pontosan egyszer fog áthaladni az így keletkezett vágáson, és a vágáson való áthaladás után azon az úton fog haladni, amit a célpontból indított Dijkstra-algoritmus által meghatározott fa tartalmaz.

Az algoritmus futásidejének csökkentéséhez a bejárási fáknak több tulajdonságát is kihasználták. Egyik az, hogy egy adott élhez tartozó vágáson átmenő élek meghatározhatóak úgy, hogyha az előző élhez tartozó vágás éleiből eltávolítjuk azokat, amelyikeknek az y-hoz közelebbi végpontja most került át az x-hez közelebbi részfába, és hozzáadjuk azokat, amelyikeknek az x-hez közelebbi végpontja most került át az x-hez közelebbi részfába. A másik fontos tulajdonság pedig az, hogy egy adott élen átmenő összes olyan helyettesítő út hossza azonos ahol az adott él keresztezi az éppen aktuális vágást x és y között. Ezt a két tulajdonságot felhasználva az algoritmus megvalósítható úgy, hogy az x és y pontokból futtatott Dijkstra-algoritmusok után, egy prioritásos sorba beletesszük azokat az éleket, amelyek az éppen aktuális vágást keresztezik, úgy hogy hozzájuk azt a súlyt rendlejük, amilyen hosszú a rajtuk áthaladó helyettesítő út hossza. Ekkor a legoptimálisabb eset a prioritásos sorban szereplő legkisebb súlyú elemmel egyezik meg, ami könnyen meghatározható. Ezek után pedig a prioritási sornak az elemeit kell csak módosítani, ami szintén egyszerűen megoldható. Amennyiben mind a Dijkstra-algoritmust, mind a prioritásos sort Fibonacci kupacok segítségével valósítjuk meg, akkor ennek az algoritmusnak a lépésszáma csak O($m + nlogn$) lesz, ami megegyezik a Fibonacci-kupacokat használó (optimális) Dijkstra-algoritmus futásidejével.



A legrövidebb utakat tartalmazó fa által meghatározott bolck-ok (Forrás: [2])

Formálisan:

Jelölje a legrövidebb útban szereplő pontokat v_i, az éleket pedig $e_i=(v_i, v_{i+1})$, az x és y közötti legrövidebb út hosszát pedig k=d(x, y).
X az x pontból, Y az y pontból indított Dijkstra-algoritmus által meghatározott bejárási fa
B_i az X fán belüli v_i pontból, kiinduló részfa, a v_{i+1} pontból kiinduló részfa nélkül.
block(u)=i ha $u \in B_i$
$e=(u, v) \notin P(x, y)$ esetén bal(e)=block(u) és jobb(e)=block(v) úgy, hogy bal(e) ≤ jobb(e)
Q egy kezdetben üres prioritásos sor melynek elemei (súly, él) párok, ahol a súly az index
L és R egy-egy k elemű tömb melyeknek elemei élek halmaza
Minden $e \in E \setminus P(x, y)$ esetén
    Ha bal(e) < jobb(e) akkor adjuk hozzá e-t L[bal(e)] és R[jobb(e)] halmazokhoz
Ciklus i=1-től (k-1)-ig
    Minden $e=(u, v) \in L[i]$
        Adjuk hozzá (w, e)-t Q-hoz ahol $w=d_X(x, u) + c(u, v) + d_Y(v, y)$
    Távolítsuk el Q-ból minden (w, e) párt ahol $e \in R[i]$
    Tároljuk el a Q-ban szereplő legkisebb súlyt az x, y út e_i éléhez tartozó helyettesítő út hosszaként

3.2 Helyettesítő út keresése irányított gráfokban

Irányított gráfokra - az irányítatlan esettel ellentétben - nem ismert a naiv, $O(nm + n^2 log(n))$ futásidejű algoritmusnál lényegesen gyorsabb determinisztikus algoritmus két pont közötti legrövidebb út összes éléhez tartozó legrövidebb helyettesítő út meghatározására, ezért a probléma megoldására közelítő, illetve véletlenen alapuló algoritmusokat fejlesztettek ki az elmúlt időszakban. (J. Hershberg és S. Suri eredeti cikkében[1] az szerepel, hogy az algoritmusuk irányított gráfra is működik, ami viszont később kiderült, hogy nem igaz.[2])

3.2.1 Véletlenen alapuló algoritmus

A véletlenen alapuló algoritmusok közül L. Roditty és U. Zwick [4] algoritmusát emelném ki, amelyik súlyozatlan (és kis egész élsúlyú) gráfokra működik. Futásideje $O(m\sqrt{n})$ és nagy valószínűséggel megtalálja a legrövidebb út mindegyik éléhez a legrövidebb helyettesítő utat. Az algoritmus azon alapul, hogy azokat a helyettesítő utakat, amelyeknél a kerülőút hossza kisebb, mint valamilyen L konstans, meg lehet találni $O(L)$ darab Dijkstra algoritmus felhasználásával, ami összesen $O(mL)$ futásidőnek felel meg. Az L-nél hosszabb kerülőutak pedig nagy valószínűséggel érintenek legalább egy csúcsot a gráfból véletlenszerűen kiválasztott $O(n/L)$ csúcs közül. Amennyiben ez a feltétel teljesül, akkor a legrövidebb, legalább L hosszú kerülőutak már csúcsonként $O(1)$ szélességi bejárás segítségével megtalálhatóak, ezért ebben az esetben a teljes futásidő $O(mn/L)$. A két eredményt összegezve, és $L=\sqrt{n}$ választást használva az algoritmus futásideje tényleg $O(m\sqrt{n})$ lesz.

3.2.2 Közelítő algoritmus

A közelítő algoritmusok közül a legismertebb Aaron Bernstein [5] algoritmusa, ami a fentebb leírt algoritmussal ellentétben tetszőleges pozitív élsúlyú irányított gráfban talál egy adott legrövidebb út összes éléhez tartozó helyettesítő utat úgy, hogy a megtalált helyettesítő út hossza legfeljebb (1+ε)-szerese az adott élhez tartozó helyettesítő út hosszának, ahol $\varepsilon \in (0, 1)$. Ennek az

algoritmusnak a futásideje O($mlog(nC/c)/\varepsilon$) ahol c a legkisebb, C pedig a legnagyobb súlyú él súlyát jelöli.

4 A naiv útválasztó séma

A legegyszerűbb útválasztó séma a naiv útválasztó séma, amely úgy működik, hogy minden pontban, minden címhez eltárolja az adott címhez tartozó kimeneti portot. Ez azt eredményezi, hogy minden csomag a legrövidebb úton fog eljutni a céljába, mivel minden csúcspontban az útválasztó tábla megmondja, hogy melyik porton kell továbbmennie a legrövidebb út használatához. Ennek a sémának a legnagyobb hátránya az, hogy minden csomópontban el kell tárolni egy $logn$ hosszúságú portot az összes lehetséges címhez, ami azt eredményezi, hogy a szükséges lokális tárigény O($nlogn$) a teljes tárigény pedig O(n^2logn) ami egy nagyobb méretű hálózat esetén túlságosan sok. Ezt a hátrányt valamelyest ellensúlyozza, hogy a címek csak $logn$ hosszúak, tetszőlegesen megválaszthatóak és statikusak, azaz a csomagokhoz rendelt cím nem változik meg a teljes útválasztó folyamat során.

Az útválasztó táblák generálásához meg kell határozni bármely két pont közötti legrövidebb utat, ami O(n^3) futásidőben megtehető például a Floyd-Warshall vagy a Johnson [6] algoritmus segítségével (vagy akár O($mn + n^2logn$) futásidőben is megvalósítható úgy, hogy minden pontból indítunk egy Dijkstra-algoritmust). Ezt követően az útválasztó táblák meghatározása már nagyon egyszerű, mivel minden pontnál csak meg kell nézni az onnan kiinduló legrövidebb utaknak az első élét, és az annak megfelelő portot kell eltárolni az útválasztó táblában.

5 L. J. Cowen útválasztó sémája

L. J. Cowen [7] kidolgozott egy útválasztó táblát, amelyik minden irányítatlan gráfra működik. A lokális útválasztó táblák mérete O($n^{2/3}log^{4/3}n$) és bármely két pont között az útválasztó tábla alapján közlekedő csomagok által megtett út legfeljebb 3-szor olyan hosszú, mint a legrövidebb út. A csomópontokban a kimeneti port meghatározása konstans időben lehetséges, a címek a naiv sémához hasonlóan itt is statikusak, viszont a címek 3-szor olyan hosszúak, és nem választhatóak meg tetszőlegesen.

5.1 A séma működése

A séma működése azon alapul, hogy minden v ponthoz meghatározzuk a hozzá legközelebbi n^α darab pontot, amelyeket az adott pont környezetének hívunk, ahol $\alpha = 1/3 + 2loglogn/3logn$. Ekkor a környezetek mérete $n^\alpha = n^{1/3}log^{2/3}n$ lesz. Ezekben a pontokban eltároljuk a v pontba vezető legrövidebb úthoz tartozó portot. Ezt követően kiválasztjuk a fontos csomópontokat a gráfban úgy, hogy minden környezet legalább egy fontos csomópontot tartalmazzon, vagy legyen szomszédos legalább egy fontos csomóponttal. Továbbá legyen fontos csomópont az a pont is, amelyik több mint $n^{(1+\alpha)/2}$ pont környezetében van benne. Ezek után minden pontban eltároljuk a fontos csomópontokba vezető legrövidebb utak első éléhez tartozó portot. A csomagok címe legyen olyan, hogy az első log_2n bit tartalmazza a célpont azonosítóját, a következő log_2n bit a célhoz legközelebbi fontos csomópont azonosítóját, a harmadik log_2n bit pedig a célhoz legközelebbi fontos csomópontból a célba vezető legrövidebb út első éléhez tartozó portot. Ezzel az összeállítással, ha a kiindulási pont és a célpont közötti távolság kicsi, akkor a csomag a legrövidebb út mentén lesz továbbítva a célig mivel az a célpont környezetén belül lesz, ahol minden pontban el van tárolva a

legrövidebb úthoz tartozó port. Amennyiben a kiindulási pont és a célpont közötti távolság nagy, akkor pedig a cím második része alapján meg lehet határozni a célponthoz tartozó fontos csomópontot, ahova a legrövidebb úton el lehet juttatni a csomagot, mivel minden pontban el van tárolva a fontos csomópontokba vezető legrövidebb utak első éléhez tartozó port. Ezt követően a csomag címének harmadik része alapján a fontos csomópontból el lehet juttatni a csomagot a célpont környezetébe, ahonnan pedig már az első esethez hasonlóan a legrövidebb úton fog menni a csomag. A cím harmadik részére azért van szükség, mivel ugyanaz a fontos csomópont nagyon sok ponthoz tartozhat, ami a fontos csomópontokban szükséges lokális tárigényt túlságosan megnövelné.

5.2 A csomópontokban működő útválasztó algoritmus

Ha a csomópontban el van tárolva a célpontba vezető legrövidebb út első éléhez tartozó port, akkor azon továbbítja a csomagot a csomópont. Ha az nincsen eltárolva, de a jelenlegi csomópont megegyezik a célponthoz tartozó fontos csomóponttal (ez a cím alapján eldönthető) akkor a cím harmadik része által meghatározott porton továbbítja a csomagot, egyéb esetben pedig a cím által meghatározott fontos csomópontba vezető legrövidebb út első élének megfelelő porton továbbítja a csomagot.

5.3 A címeket generáló algoritmus

A séma működéséhez meg kell határozni az egyes pontok címeit. Az első $log_2 n$ bit az adott, mivel az a pont egyedi azonosítója (indexe), a következő $2log_2 n$ bitet viszont ki kell számolni. Ehhez először két részletben meghatározzuk a fontos csomópontok listáját (kiterjesztett dominál ponthalmaz meghatározása és a több mint $n^{(1+\alpha)/2}$ környezetben benne levő pontok meghatározása), majd minden ponthoz megkeressük a hozzá legközelebb lévő fontos csomópontot. Ezek után a címek már egyszerűen meghatározhatóak mivel a fontos csomópontok címének második és harmadik része nem használt, ezért oda tetszőleges értékeket írhatunk, a többi pontnál pedig ismerjük a hozzá legközelebb lévő fontos csomópontot és az onnan a hozzá vezető legrövidebb utat is, amiből a cím harmadik része azonnal meghatározható.

Formálisan:

Minden v ∈ V-re futtassunk csonkolt-Dijkstra (n^{α}) algoritmust
  B_v <- {y| y benne van v n^{α} legközelebbi szomszédjában}
  R_v <- {y| v ∈ B_y}
C <- {v| $|R_v| > n^{(1+\alpha)/2}$}
D <- {Kiterjesztett dominál halmaz}
L <- C U D /* L a fontos csomópontok halmaza */
Minden v ∈ V-re
  l_v <- $arg\ min_{l \in L}\ d(l, v)$
Minden l ∈ L-re futtassunk csonkolt-Dijkstra (n^{α}) algoritmust
Minden v ∈ V \ L
  v <- (v, l_v, e_{lv}(v)) /* e_{lv}(v) az első port az l_v és v közötti legrövidebb út mentén */
Minden v ∈ L
  v <- (v, 0, 0)

Az algoritmusban szereplő csonkolt-Dijkstra(n^α) algoritmus azt jelenti, hogy az adott pontból indított Dijkstra-algoritmust csak addig futtatjuk, amíg az n^α legközelebbi pontot meg nem találja. Ennek az algoritmusnak a futásideje O($n^{2\alpha}$) [8]

A kiterjesztett dominálló halmaz [9][10]: Legyen G=(V, E) súlyozott, irányítatlan gráf. Jelölje B_v a v ponthoz legközelebbi n^α darab csúcsot. Ekkor létezik egy D ⊆ V halmaz, melyre |D|=O($n^{1-\alpha}logn$) és bármely v ∈ V, D ∩ B_v ≠ 0 és ez a halmaz megtalálható mohó algoritmus segítségével O($m + n^{1+\alpha}$) futásidőben.

5.4 Az útválasztó táblát generáló algoritmus

A címek meghatározása után meg kell határozni a lokális útválasztó táblákat is. Ehhez először minden pontból megkeressük a hozzá legközelebb lévő n^α pontot, amelyek a környezetét alkotják, majd az így megtalált pontok közül azokban, ahonnan a környezet középpontjába vezető legrövidebb úton nincsen fontos csomópont, ott eltároljuk a környezet középpontjába vezető legrövidebb út első éléhez tartozó portot. Ezen kívül minden fontos csomópontból meghatározzuk az össze többi pontba vezető legrövidebb utat egy Dijkstra-algoritmus segítségével, és ezek után az összes pontban eltároljuk az összes fontos csomópontba vezető legrövidebb út első éléhez tartozó portot.

Formálisan:

Minden v ∈ V-re futtassunk csonkolt-Dijkstra (n^α) algoritmust és jegyezzük fel, hogy melyik u pontokat értük el a v pontból kiindulva és, hogy létezik-e l ∈ L a v-ből u-ba vezető legrövidebb úton.
    Minden u-ra, amit elértünk a csonkolt-Dijkstra során v-ből
        Ha nem létezik l ∈ L a legrövidebb úton v-ből u-ba
            Tároljuk el (v, e_u(v))-t u-ban
Minden l ∈ L futtassunk teljes-Dijkstra-algoritmust
    Minden e ∈ V
        Tároljuk el (l, e_u(l))-t u-ban

6 Helyettesítő utak használata a naiv útválasztó séma esetén

Ahhoz, hogy a gráfban bármely pontból el lehessen jutni bármelyik pontba abban az esetben is, ha egy él megsérült, szükség van arra, hogy a gráf legalább kétszeresen élösszefüggő legyen. A dolgozat további részében feltételezem, hogy a vizsgált hálózat rendelkezik ezzel a tulajdonsággal, mivel különben a helyettesítő utak keresésének nem lenne értelme.

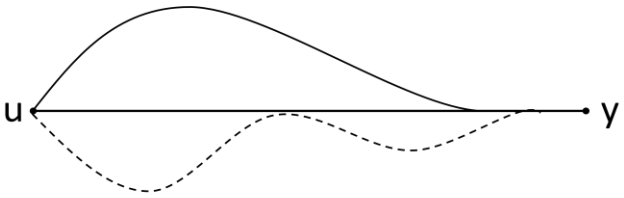
Tegyük fel, hogy el szeretnénk jutni x pontból y pontba. Amennyiben a gráf egyik éle sem hibásodott meg, vagy az x és y közötti legrövidebb út nem megy át a hibás élen akkor a naiv séma mindenféle kiegészítés nélkül működik. Amennyiben viszont az x és y közötti útnak meghibásodott valamelyik éle, akkor már a naiv séma nem nyújt elég információt a csomópontoknak ahhoz, hogy az x-ből y-ba menő csomagot eljuttassák a céljáig. A probléma megoldásához minden csomópontban tároljunk el minden címhez még egy másodlagos portot is a naiv sémában lévő elsődleges port mellé. Ez sem a lokális sem a teljes tárigény nagyságrendjét nem változtatja meg, csak a tárigényben szereplő konstans értékét növeli meg a duplájára.

LEMMA:

Ha a P(x, y) út valamelyik éléhez tartozó helyettesítő út átmegy az u ponton, akkor a helyettesítő út u pont utáni része pontosan egyszer fog visszacsatlakozni a P(u, y) útba és onnantól végig ugyanazokon az éleken fog haladni, mint a P(u, y) út. (Ez a csatlakozási pont a köztes pontok mellett az u és az y pont is lehet.)

BIZONYÍTÁS:

A helyettesítő út legalább egyszer biztosan vissza fog csatlakozni a P(u, y) útba, mivel a két út végpontja azonos. Tegyük fel, hogy a helyettesítő út 1-nél többször csatlakozik vissza a P(u, y) útba. Ekkor a helyettesítő útnak van legalább két különböző szakasza melyeknek a két végpontja rajta van a P(u, y) úton, az élei viszont diszjunktak a P(u, y) út éleitől. Ekkor viszont a P(u, y) út mentén kihagyott szakaszok közül legfeljebb egyiken lehet a hibás él, amiből következik, hogy a másik szakaszon a helyettesítő út haladhatott volna a P(u, y) út mentén, ami rövidebb lett volna, mivel a P(u, y) egy legrövidebb út. Ez viszont ellentmond annak, hogy a legrövidebb helyettesítő utakat kerestük meg. ▉



A szaggatott út nem lehet egy legrövidebb helyettesítő út

A másodlagos portok definiálásához először határozzuk meg a gráfban szereplő bármely két pont közötti legrövidebb út minden egyes éléhez a hozzá tartozó legrövidebb helyettesítő utat. Ezt követően a gráf minden u pontjában az y címhez tartozó másodlagos portot definiáljuk úgy, hogy az az u ponton átmenő, y pontba tartó helyettesítő utak közül a P(u, y) útba legkésőbb visszacsatlakozó helyettesítő úthoz tartozó port legyen.

LEMMA:

A fenti definíció mindig egyértelműen meghatároz egy utat.

BIZONYÍTÁS:

Tegyük fel, hogy van két különböző helyettesítő út P1 és P2, amelyek átmennek az u ponton és ugyanott csatlakoznak vissza a P(u, y) útba. Legyen ez a visszacsatlakozási pont v. Ekkor $u \neq v$ biztosan teljesül, mivel u=v esetén az út hossza a visszacsatlakozásig 0 él, ami csak 1-féle útnak felelhet meg. Ekkor P1 és a P2 biztosan a P(u, y) út valamelyik éléhez tartozó helyettesítő út, mivel bármely más élhez tartozó helyettesítő út a P(u, y) utat használná, mivel az rövidebb, mint P1 vagy P2. P1 és P2 közül viszont valamelyik hosszabb, mint a másik, mivel feltettük, hogy a gráfban szereplő bármely két út hossza különböző. Legyen $d(P1)>d(P2)$. Ekkor viszont P1 nem lehet egy legrövidebb helyettesítő út, mivel beláttuk, hogy csak a P(u, y) út v előtti éleit helyettesítheti, azokat viszont helyettesíti a nála rövidebb P2 út is. Ezzel ellentmondásra jutottunk, amivel igazoltuk az állítást. ▉

LEMMA:

A legkésőbb visszacsatlakozó helyettesítő út hossza a legnagyobb az u és y közötti helyettesítő utak közül.

BIZONYÍTÁS:

Mint azt az előbbi bizonyításban már láttuk, az összes u és y közötti helyettesítő út a P(u, y) út valamelyik éléhez tartozó helyettesítő út. Tegyük fel, hogy a legkésőbb visszacsatlakozó út hossza kisebb, mint egy korábban visszacsatlakozó helyettesítő úté. Ekkor a később visszacsatlakozó helyettesítő út minden olyan élet elkerül a P(u, y) út mentén, amelyiket a nála hosszabb, de korábban visszacsatlakozó helyettesítő út is elkerül. Ez viszont ellentmondáshoz vezet, mivel ekkor a korábban visszacsatlakozó, de hosszabb helyettesítő út nem a legrövidebb helyettesítő út az adott élet helyettesítő utak közül. ▉

6.1 Az útválasztó algoritmus

Minden csúcspontban működjön úgy az útválasztó algoritmus, hogy ha az elsődleges port mentén nem tudja továbbküldeni a csomagot, mivel az az él hibás, vagy a csomag az elsődlegesen meghatározott port irányából érkezett, akkor a másodlagos porton küldje tovább, egyébként pedig az elsődleges porton.

Formálisan:

```
Naiv():
Ha(Elsődleges port hibás vagy a csomag az elsődleges portról érkezett)
    Továbbítás a másodlagos port szerint
Egyébként
    Továbbítás az elsődleges port szerint
```

TÉTEL:

A fenti útválasztó stratégiát és a fentebb definiált adatokat tárolva minden csúcsban, bármely két pont között működik a kommunikáció amennyiben legfeljebb egy hibás él van a gráfban. Amennyiben a forrás és a cél közötti legrövidebb út egyik éle sem hibás, akkor a csomagok a legrövidebb úton mennek végig.

BIZONYÍTÁS:

A tétel második része következik abból, hogy amennyiben minden csomópont tudja a csomagot az elsődleges porton továbbítani, akkor a csomag a naiv útválasztó séma szerint meghatározott úton megy végig, ami a naiv séma felépítése miatt biztosan a forrás és a cél közötti legrövidebb út.

A tétel első részének a bizonyítása egy kicsit bonyolultabb. Tegyük fel, hogy egy y-ba tartó csomag a legrövidebb út mentén haladt az u pontig, ahol viszont nem tudott továbbmenni a legrövidebb úton, mivel a következő él, amit jelöljön (u, u') hibás volt.

Először mutassuk meg, hogy a csomag biztosan nem fog visszatérni az u pontba. Az útválasztó algoritmus definíciója miatt a csomag soha sem fog visszafordulni az után, hogy egy hibás élbe ütközött. (Az ütközés utáni első lépés lehet egy visszafordulás például akkor, ha a hibás él miatt u egy zsákutca végpontja.) Amíg a csomag a másodlagos portok szerint van továbbítva, addig az (u, u') élhez tartozó helyettesítő úton halad, ami triviális, hogy nem tér vissza az u ponthoz. Ha v pontban a csomag az elsődleges porton lett továbbítva, akkor onnan pedig végig a

P(v, y) úton fog haladni, amelyen nincsen rajta az u pont. Ezt bizonyítsuk be indirekt módon. Tegyük fel, hogy rajta van az u pont a P(v, y) úton. Ekkor a csomag P(v, u) úton fog haladni u pontig, amely útnak legyen az első éle (v, v'). Ekkor tudjuk, hogy a csomag nem v' irányából érkezett v pontba, mivel a csomag soha sem fordul meg a hibás éllel való találkozást követően. Ez viszont ellentmond annak, hogy a csomag a legrövidebb helyettesítő úton közlekedik. Ezzel megmutattam, hogy a csomag nem fog visszatérni az u pontba u'-től különböző irányból. (Az u' pontra azért nem igaz a fenti bizonyítás, mivel a helyettesítő út kihagyja az (u, u') élet a P(v, y) legrövidebb út viszont nem feltétlenül.

Ezek után megmutatom, hogy P(v, y) nem fog átmenni az (u', u) élen sem. Az u pont irányából az előző bizonyítás miatt biztosan nem fog áthaladni. Tegyük fel, hogy P(v, y) áthalad az (u', u) élen u' pont irányából. Ebben az esetben viszont a P(v, y) kétszer használja az (u, u') élt, mivel a P(u, y) út is áthalad rajta (u, u') irányból. Ez viszont ellentmond annak, hogy P(v, y) egy legrövidebb út, mivel két pont közötti legrövidebb út minden élen legfeljebb egyszer haladhat végig.

A fenti két bizonyítással bebizonyítottam, hogy a naiv séma ilyen módon történő kiterjesztése minden esetben eljuttatja a csomagot a célig legfeljebb egy hibás él esetén, mivel a csomag csak egyszer ütközhet bele a hibás élbe, azt követően viszont a hibás élhez és a célponthoz tartozó helyettesítő úton haladva el fog jutni a céljáig. ■

MEGJEGYZÉS:

A csomagok nem kerülhetnek végtelen ciklusba a hibás éllel történő találkozás után, mivel az előbbiekben már megmutattam, hogy a hibás éllel legfeljebb egyszer találkoznak. Ha valamikor elsődleges port mentén lettek továbbítva, akkor onnantól kezdve a naiv útválasztó séma működése miatt biztosan nem kerülnek körkörös pályára. Az az eset pedig nem lehetséges, hogy másodlagos portok mentén haladjanak körbe-körbe a következő indirekt bizonyítás miatt. Tegyük fel, hogy a csomag körpályán mozog. Ekkor létezik, egy pont ahol a kör bezárul, amit jelöljön w (ebben a pontban jár először a csomag kétszer). Ebbe a w pontba az első és a második esetben két különböző irányból érkezett a csomag, mivel ez volt az első olyan pont ahol kétszer járt. Ekkor viszont az egyik esetben biztosan az elsődleges port mentén fog továbbmenni, ami ellentmond annak a feltevésnek, hogy a másodlagos pontok mentén körkörösen mozog. ■

Ezzel megmutattam, hogy a fent definiált útválasztó táblák és a hozzá tartozó útválasztó algoritmus segítségével el tudjuk érni a célunkat a naiv útválasztó sémára támaszkodva a lokális és az összes tárigény nagyságrendjének változtatása nélkül, megtartva a statikus, $log n$ hosszú címeket és a konstans idejű döntéseket a csúcspontokban. A csomagok által megtett út hossza minimális marad abban az esetben, ha egyetlen él sem hibásodott meg a gráfban. Ha egy él meghibásodott és a hiba csak a hibás él végpontjaiban detektálható, akkor a csomag által megtett út csak kis mértékben lesz hosszabb az optimálisnál. Az útválasztás során az egyetlen hozzáadódó nehézség az, hogy a csomópontoknak tudniuk kell a döntéshez azt, hogy a csomag melyik csomóponttól érkezett, ez viszont a valóságban egy könnyedén leküzdhető akadály.

6.2 Az útválasztó tábla generálása

Miután pontosan definiáltam, hogy milyen adatokat kell eltárolni az egyes csúcsokban, térjünk rá arra a kérdésre, hogy hogyan lehetséges ezeket az információkat meghatározni. Az

útválasztó táblák generálási sebessége nem egy kritikus faktor a probléma megoldása szempontjából, mivel csak a hálózat megépítésekor, vagy egy esetleges módosításkor kell őket újragenerálni offline módon, amikor nem okoz problémát, ha viszonylag sokáig tart, de ennek ellenére meg kell próbálni a generálási sebességet maximalizálni.

Először is meg kell határozni bármely két pont között a legrövidebb utat. Erre az információra szükség van az alap naiv útválasztó séma legenerálásához, illetve annak az eldöntéséhez, hogy az egyes helyettesítő utak hol csatlakoznak vissza a legrövidebb útba. A bármely két pont közötti legrövidebb út meghatározható például a Floyd-Warshall-algoritmus vagy a Johnson-algoritmus [6] segítségével O(n^3) futásidőben. Ezek után meg kell határozni a gráfban szereplő minden legrövidebb út minden egyes élére a hozzá tartozó helyettesítő utat. Erre egy viszonylag egyszerű megoldás, ha minden élre elvégezzük, hogy az adott élt eltávolítjuk, majd az eltávolított él mind a két végpontjából futatunk egy-egy Dijkstra-algoritmust. Az így megtalált utak lesznek az eltávolított élhez tartozó, az adott pontból kiinduló helyettesítő utak. Ezzel a módszerrel az összes helyettesítő utat megtaláljuk, mivel a csomagok csak a hibás él végpontjában észlelik, hogy az az él hibás, ezért a helyettesítő út a hibás él végpontjából a célpontba vezető legrövidebb olyan út lesz amelyik elkerüli a hibás élt. Ennek az algoritmusnak a futásideje O($m^2 + mnlogn$), mivel összesen $2m$ darab Dijkstra-algoritmus futtatására van szükség. Ez a futásidő sűrű gráfokra megegyezik, ritka gráfokban pedig kisebb, mint ha J. Hersenberg és S. Suri [1] algoritmusát futtatnánk le minden pontpárra ami O($n^2m + n^3logn$) futásidejű lenne. Következő lépésként minden (x, y) pontpárhoz meg kell határozni az x ponton áthaladó y pontba tartó helyettesítő utak közül azt, amelyiknek az x pont utáni része a leghosszabb. Az így meghatározott helyettesítő út első éléhez tartozó portot kell eltárolni x-ben, mint az y címhez tartozó másodlagos port. Ehhez a Dijkstra-algoritmusok futtatása során minden x ponthoz eltároljuk a távolságot és az x pontból induló út első éléhez tartozó portot azokhoz az y pontokhoz, amelyek a bejárási fában alatta helyezkednek el. A tárolás során amennyiben az x pontban az y ponthoz tartozóan már van információ eltárolva, akkor megvizsgáljuk, hogy a jelenleg vizsgált távolság kisebb vagy nagyobb, mint a már letárolt távolság és amennyiben nagyobb, akkor felülírjuk a távolságot és az adott úthoz tartozó portot, egyébként pedig eldobjuk az adatot. Az eltároláshoz O(n^2) futásidőre van szükség minden egyes Dijkstra-algoritmus futtatásakor mivel minden pontnál legfeljebb n különböző adat ellenőrzésére, illetve frissítésére lehet szükség, ahol egy adat ellenőrzése és frissítése konstans időben megvalósítható. Ezzel a teljes generáló algoritmus futásideje O($n^3 + m^2 + mnlogn + n^2m$) lett, ami legfeljebb O(n^4)-nel egyenlő.

Formálisan:

Útválasztó tábla generálása:
Floyd-Warshall algoritmus futtatása
Minden $v_1 \in V$, $v_2 \in V \backslash v_1$
    Tároljuk el v_1-ben a v_2-be vezető legrövidebb út portját
Minden $e=(v_1, v_2) \in E$
    T = Dijkstra(v_1-ből, G\e-re)
    Leghosszabb helyettesítő utak frissítése(T)
    T = Dijkstra(v_2-ből, G\e-re)
    Leghosszabb helyettesítő utak frissítése(T)
Minden $v_1 \in V$, $v_2 \in V \backslash v_1$
    Tároljuk el v_1-ben leghosszabb_helyettesítő_út(v_1, v_2) portját másodlagos portként

Leghosszabb helyettesítő utak frissítése (Adott bejárási fával):
Minden $v_1 \in V$, $v_2 \in V \backslash v_1$
    Ha(v_1 gyermeke v_2 és (leghosszabb_helyettesítő út(v_1, v_2) nem definiált, vagy
    $d(v_1, v_2) >$ leghosszabb_helyettesítő_út(v_1, v_2)))
        leghosszabb_helyettesítő_út(v_1, v_2) = $P(v_1, v_2)$

7 Helyettesítő utak használata L. J. Cowen útválasztó sémája esetén

L. J. Cowen útválasztó sémája sokkal bonyolultabb, mint a naiv útválasztó séma. Emiatt a helyettesítő utak használatának megoldása is bonyolultabb ebben az esetben. Amennyiben a gráfban egyik él sem hibásodott meg, akkor a L. J. Cowen által definiált útválasztó sémát lehet használni változtatás nélkül. Amennyiben viszont valamelyik él meghibásodott (kiesett) akkor a séma nem működik megfelelően. A hibás él a kiindulópont és a célpont közötti úton háromféle helyzetben helyezkedhet el. A célhoz legközelebbi fontos csomóponthoz vezető úton, a fontos csomópont utáni élen, aminek a portja a címben van eltárolva, illetve a célpont környezetében ahol már a célponthoz vezető legrövidebb úthoz tartozó port el van tárolva.

A fontos csomópontig való eljutás biztosítható úgy, hogyha minden porthoz amely egy fontos csomóponthoz vezető legrövidebb úthoz tartozik, eltárolunk egy másodlagos portot, amely ugyanúgy van definiálva, mint a másodlagos port a naiv út választó séma esetén. Ezzel a megoldással minden csomagot el tudunk juttatni a célpontjához tartozó fontos csomóponthoz ugyanazért, amiért a naiv útválasztó séma esetén minden csomagot el tudunk juttatni a céljáig. A lokális tárigény nagyságrendje nem növekszik ennek a változtatásnak a hatására.

A fontos csomópontoktól a célpontig vezető úttal sokkal több a probléma. A csomóponttól a célpontig vezető út első éle csak a csomag címében van eltárolva. Amennyiben ez az él meghibásodik, akkor semmilyen lokális adat nem segíti az ahhoz az élhez tartozó helyettesítő út meghatározását mivel egy fontos csomópont nagyon sok célponthoz tartozhat. A probléma megoldásához szükség van egy helyettesítő útra, amelyik mentén a csomagot be lehet juttatni a célpont környezetébe. Erre két megoldás lehetséges.

Az egyik megoldás a csomagnak egy másodlagos fontos csomópontba való eljuttatása, ahonnan a csomag egy élen be tud jutni a célpont környezetéhe. Ehhez a megoldáshoz a csomagok címében szükség van egy másodlagos fontos csomópont eltárolására és a másodlagos fontos csomóponttól a célig vezető út első éléhez tartozó port eltárolásra. Ez a sémában eredetileg használt $3logn$ hosszú címet további $2logn$ mennyiségű adattal megnöveli, illetve a címben tárolni kell, hogy a csomagot az elsődleges vagy a másodlagos fontos csomópontba kell továbbítani. Ennek az eltárolásához vagy szükség van még egy bitre, vagy amikor kiderül, hogy a másodlagos fontos csomóponthoz kell továbbítani a csomagot, akkor a csomag címében fel kell cserélni az elsődleges és a másodlagos fontos csomóponthoz tartozó adatokat (fontos csomópont indexe és az onnan kiinduló legrövidebb út első élhez tartozó port).

A másik megoldás a csomagnak a célpont környezetébe való bejuttatására az, ha a fontos csomópont a környezetnek legalább 2 pontjával szomszédos (ez a környezet megfelelő megválasztásával érhető el) és e két ponttól a célig vezető utak nem mennek át a másik ponttól a környezetbe vezető élen. Ebben a felállásban a címet a fontos csomóponttól a cél környezetébe

vezető másodlagos élhez tartozó porttal kell kibővíteni, ami csak $logn$ extra bitet jelent és a cím továbbra is maradhat statikus.

Amennyiben a hibás él a célpont környezetén belül van akkor két részeset lehetséges. Az egyszerűbb, ha P(u, v) összes éléhez tartozó helyettesítő út végig a környezeten belül halad, ahol u az első csúcspont a környezeten belül, v pedig a célpont. Ebben az esetben a naiv sémára kidolgozott megoldás itt is alkalmazható. Amennyiben viszont valamelyik helyettesítő út elhagyja a v pont környezetét, akkor ott már a csúcspontokban lévő lokális útválasztó táblák nem fogják tartalmazni a legrövidebb úthoz, illetve a helyettesítő úthoz tartozó portot, ezért a csomag ismételten a fontos csomóponthoz lesz visszairányítva, ahonnan körkörösen vissza fog térni ahhoz a helyettesítő úthoz ahonnan már egyszer vissza lett irányítva. (A körkörös visszairányítás nem fog mindig bekövetkezni, mivel előfordulhat, hogy a csomag nem a fontos csomóponttól érkezett be a környezetbe, hanem a fontos csomóponthoz vezető út „véletlenül” áthaladt a célpont környezetén.)

Ennek a problémának az egyik megoldása, ha a környezeteket úgy definiáljuk, hogy bármely v-be tartó út éleihez tartozó helyettesítő út a környezeten belül haladjon. Ez akkor és csak akkor teljesülhet, ha a környezet kétszeresen él összefüggő. Ekkor a csomagok címének hossza nem nőtt meg, viszont a szükséges lokális tárkapacitás nőhet abban az esetben, ha az egyes környezetek mérete az extra feltételek miatt nagyságrendileg is nagyobb lesz, mint az eredeti sémában használt környezetek méretei.

A másik megoldás, hogy az előző fajta hibánál használt másodlagos csomópontot felvesszük úgy, hogy a két fontos csomópontból a célpontig vezető utak legyenek diszjunktak. Ezek után, amikor a csomag belép a célpont környezetébe, akkor a csomag címében felcseréljük az elsődleges és a másodlagos fontos csomópontokhoz tartozó adatokat. Ekkor, ha a csomag kikerül a célpont környezetéből, akkor a célponthoz tartozó másik fontos csomóponthoz fog menni, mint azelőtt, ezért amennyiben eléri azt a fontos csomópontot, akkor onnan el tud jutni a célig. Probléma akkor van, ha a fontos csomópontig vezető úton a csomag áthalad a célpont környezetén, ahonnan már közvetlenül a célhoz fogják irányítani a csomópontok, ami lehet, hogy ugyanazon a hibás élen keresztül történik, mint amelyiknek a helyettesítő útjáról letértünk és ezért ciklikusan fogunk ezen az úton haladni. Ennek az elkerülésére ismételten két lehetőség is adódik. Az egyik a helyettesítő utak olyan megválasztása, ami megakadályozza ezt a hatást, amihez viszont nagyon bonyolult feltételrendszert kellene definiálni, a másik lehetőség pedig, hogy egy, a címben szereplő biten eltároljuk, ha a csomag a környezeten belül hibás éllel találkozott. Ebben az esetben mindenképpen a fontos csomóponthoz továbbítjuk, ahol majd ennek a bitnek az értékét visszaállítjuk alapállapotba.

A fenti lehetőségek közül a legoptimálisabbnak az az eset ígérkezik, ha minden ponthoz definiáljuk a másodlagos csomópontot. A jelenleg aktív (elsődlegesnek tekintendő) fontos csomópontot úgy változtatjuk meg, hogy felcseréljük az elsődleges és a másodlagos csomóponthoz tartozó adatokat, és felveszünk egy extra bitet, ami alapján kényszeríteni tudjuk a rendszert, hogy először a fontos csomópontba küldje el a csomagot annak ellenére, hogy esetlegesen tudja a célponthoz vezető legrövidebb úthoz tartozó portot. Ekkor a csomagok címének hossza az eredeti sémánál használt $3log_2 n$ bit helyett $5log_2 n + 1$ bit lesz, viszont ez nem nagyságrendi növekedés. A másodlagos csomópontok felvételéhez pedig a fontos csomópontok számát, illetve a környezetek méretét kellhet kis mértékben megnövelni, ami viszont megoldható úgy, hogy a lokális tárigény nagyságrendje ne változzon.

Az előző részekben felvetett többi megoldási lehetőséget azért vetem el, mivel azok mindegyikéhez a környezetet nagymértékben át kellene definiálni. Ez nagyon könnyen elronthatja a hibátlan hálózat esetén elért eredményeket (multiplikatív hiba nagyságát és a lokális tárigényt) illetve megnehezíti a helyettesítő utakra kidolgozott séma használatát más, a vizsgált sémához hasonló, viszont annál újabb és fejlettebb sémákra, mivel a környezet nagymértékű átdefiniálása miatt újra kellene vizsgálni az eredeti sémában elért eredményeket is.

A másodlagos csomópontok könnyebb megválasztásához, még egy egyszerű módosításra lesz szükségünk. Még pedig arra, hogy a csomag címében egy további bit jelezze azt, ha a csomag éppen a másodlagos csomóponttól a célponthoz vezető helyettesítő úton halad. Erre az információra azért lesz szükség, mivel így megválasztható a másodlagos fontos csomópont úgy is, hogyha az onnan a célpontba vezető legrövidebb út nem él diszjunkt az elsődleges fontos csomóponttól a célpontba vezető úttal, de létezik olyan, a másodlagos fontos csomópontból a célpontba vezető út, amelyik él diszjunkt tőle. Ez a változtatás csak a címek hosszát növeli meg további egy bittel, ami a nagyságrend szempontjából nem számít, viszont későbbiekben nagyon sok helyen fogjuk majd kihasználni. Ez után a változtatás után a címek végleges hossza $5log_2 n + 2$ bit lett.

A két jelzőbit által felvehető értékeket és az értékek jelentését az alábbi táblázatban foglaltam össze:

Kötelező fontos csomópont	Kötelező másodlagos helyettesítő út	Magyarázat
0	0	Normál állapot
0	1	A fontos csomópontból a célig vezető helyettesítő úton kell haladnia a csomagnak
1	0	A jelenleg elsődleges fontos csomóponthoz kell mennie a csomagnak
1	1	A jelenleg elsődleges fontos csomóponthoz kell mennie a csomagnak majd onnan a helyettesítő úton kell mennie a célig

7.1 A másodlagos csomópontok és a módosított környezetek definiálása

A másodlagos csomópontokkal és a módosított (kibővített) környezetekkel szemben az alap útválasztó séma működése érdekében az az elvárásunk, hogy az eredeti környezetek mindig legyenek részhalmazai a módosított környezeteknek és a másodlagos csomópontok felvétele során az eredeti sémában fontos csomópontként definiált pontok továbbra is maradjanak fontos csomópontok. Ezen felül még elvárás, hogy minden ponthoz a másodlagos csomópont legyen szomszédos a célpont környezetével és a másodlagos csomópontból a célpontba vezessen egy út, amelyik legfeljebb $4n^{\alpha}$ él hosszú és diszjunkt az elsődleges fontos csomóponttól a célhoz vezető úttól.

LEMMA:

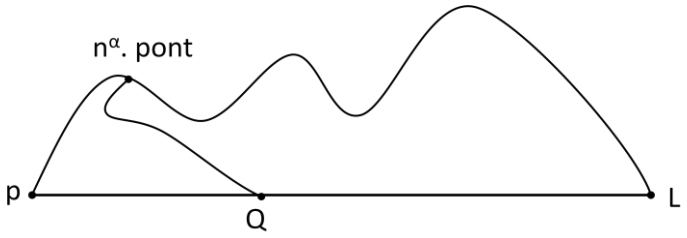
Legfeljebb $n^{1-\alpha}$ új fontos csomópont felvételével elérhető, hogy minden ponttól legfeljebb $4n^{\alpha}$ él távolságra legyen a hozzá tartozó másodlagos fontos csomópont.

BIZONYÍTÁS:

Nevezzük problémás pontnak azokat a pontokat, amelyekre nem teljesül az a feltétel, hogy a vizsgált pont és a hozzá tartozó fontos csomópont közötti legrövidebb út éleit eltávolítva a gráfból, a legkevesebb él távolságra lévő fontos csomópont legfeljebb $4n^{\alpha}$ él távolságra van tőle.

Mivel a gráf kétszeresen él-összefüggő, ezért összefüggő marad abban az esetben is, ha eltávolítjuk belőle két tetszőlegesen kiválasztott pont közötti legrövidebb út összes élét.

Amennyiben a gráfunkban nincsen problémás pont, akkor a lemma állítása triviálisan teljesül. Amennyiben van problémás pont, akkor azt jelöljük p-vel, a hozzá tartozó fontos csomópontot pedig L-el. Keressük meg a p pontból az L pontba vezető, legkevesebb élből álló utat abban a gráfban, amelyikből eltávolítottuk a p pontból L pontba vezető legrövidebb út éleit. Ezt az utat jelölje X. Ekkor X biztosan több mint $4n^{\alpha}$ él hosszú, mivel egyébként L jó lenne másodlagos fontos csomópontnak is. Amennyiben X út első n^{α} pontjából valamelyikhez nem az L pont tartozik, mint fontos csomópont, akkor az ahhoz a ponthoz tartozó fontos csomópont p-től legfeljebb $2n^{\alpha}$ él távolságra lenne, és ezért megfelelne másodlagos fontos csomópontnak. Ebből következik, hogy az X út első n^{α} pontja legfeljebb n^{α} él távolságra van L-től az eredeti gráfban, mivel L a hozzájuk tartozó fontos csomópont. Az X út n^{α}-adik pontjából L-be vezető legrövidebb út biztosan visszacsatlakozik a p-ből L-be vezető útba, mivel az X úton haladva több mint n^{α} él távolságra van L-től. Ez a visszacsatlakozási pont legyen a Q pont. Vegyük fel Q pontot, mint fontos csomópontot. Ekkor Q pont megfelelő másodlagos fontos csomópontja lesz p-nek, és az X út első n^{α} pontjának, és bármelyiktől legfeljebb $2n^{\alpha}$ él távolságra lesz. Ezzel meghatároztuk, hogy melyik pontot kell felvenni fontos csomópontnak ahhoz, hogy legyen p-hez tartozó másodlagos fontos csomópont. Ezt követően frissítsük a problémás pontok listáját, ami biztosan kevesebb pontot fog tartalmazni, mint előbb mivel új pont nem kerülhet bele, p pont pedig eddigi benne volt, de most már nem lesz benne. Amennyiben még van problémás pont, akkor folytassuk az eljárást mindaddig, amíg el nem fogynak a problémás pontok.



Ezek után még meg kell még mutatni, hogy legfeljebb $n^{1-\alpha}$ új fontos csomópontot vettünk fel a fenti algoritmussal. Definiáljuk a megoldott pontok halmazát úgy, hogy minden problémás pontra miután lefutattuk az algoritmust, beletesszük p pontot és az X út első n^{α} pontját a halmazba. Ekkor minden esetben n^{α} új pont fog bekerülni a halmazba a következő okokból. Tegyük fel, hogy van egy pont, amelyik már benne volt a megoldott pontok halmazában és most ismét bele kéne tennünk. Legyen ez a pont a w pont. Ebben az esetben a most vizsgált problémás pont legfeljebb n^{α} él távolságra van a w ponttól. Amikor w pont be lett téve a megoldott pontok halmazába, akkor felvettünk egy fontos csomópontot, amelyik a w ponttól legfeljebb $2n^{\alpha}$ él távolságra van. Ez a fontos csomópont megfelelő másodlagos fontos csomópont lesz a most vizsgált ponthoz is, amiből viszont következik, hogy a most vizsgált pont nem lehet problémás pont. Ezzel ellentmondásra jutottam, és bebizonyítottam, hogy minden pontot legfeljebb 1-szer fogunk hozzáadni a megoldott pontok halmazához. Mivel a megoldott pontok halmazának a mérete legfeljebb n lehet, és egy problémás csomópont vizsgálatakor n^{α} új pontot adtunk hozzá, ezért legfeljebb $n^{1-\alpha}$ problémás pontot kellett megvizsgálni, amiből következik, hogy legfeljebb ugyanennyi új fontos csomópontot kellett felvennünk. Ezzel bizonyítottam a fenti lemma állítását. (A $4n^{\alpha}$ korlátra azért volt szükség mivel a bizonyítás során elhanyagoltam az $O(1)$ nagyságrendű tagokat, amelyek a lemma állítását nem tudják befolyásolni.) ▮

Ezek után terjesszünk ki minden környezetet úgy, hogy hozzáadjuk a kiválasztott másodlagos fontos csomópontból az eddigi környezetig vezető út (amelyik mentén megtaláltuk a fontos csomópontot) pontjait a másodlagos fontos csomópont kivételével. Ezzel a környezetek mérete megnőtt, de felülről becsülhető $5n^{\alpha}$-val mivel az előző lemmából következik, hogy legfeljebb $4n^{\alpha}$ pontot adtunk hozzá. A környezetnek az a fontos tulajdonsága, hogy a környezet bármely pontjából a célpontig vezető út a környezeten belül halad, továbbra sem változott meg. A környezetek kibővítése miatt előfordulhat viszont, hogy bizonyos pontok nagyon sok környezethez tartoznak, és ezért a lokális útválasztó tábla mérete nagyságrendileg is megváltozna. Ennek a problémának a megoldására vegyük fel újra a fontos csomópontok listáját úgy, hogy a kiterjesztett dominál halmaz által meghatározott fontos csomópontokat nem változtatjuk meg, viszont a másik definiáló algoritmust, ami a nagyon sok környezetben szereplő csomópontokat határozza meg (az eredeti algoritmust az 5.4-es részben írtam le, aminek azt a részét kell újra lefuttatni, ami a C-vel jelölt csomópont halmazt számolja ki) ismételten lefuttatjuk.

LEMMA:

Az újradefiniált csomópontok száma továbbra is O($n^{1-\alpha}logn + n^{(1+\alpha)/2}$) lesz.

BIZONYÍTÁS:

A bizonyítás gyakorlatilag megegyezik a L. J. Cowen által a csomópontok számának bizonyítására használt bizonyítással. A kiterjesztett dominál ponthalmaz mérete továbbra is O($n^{1-\alpha}logn$) marad, mivel azon semmit sem változtattunk. A másodlagos fontos csomópontok meghatározásánál legfeljebb O($n^{1-\alpha}$) új fontos csomópontot vettünk fel az ott bebizonyított lemma miatt. Azoknak a pontoknak a száma, amelyek azért fontos csomópontok mivel sok környezetben vannak benne a definíciójukat felhasználva =O($n^{(1+\alpha)/2}$) mivel C={y| $|R_y|>n^{(1+\alpha)/2}$}. $\sum_y |R_y| = \sum_v |B_v| \leq \sum_v |5n^{\alpha}| = 5n^{1+\alpha}$. Ebből pedig következik, hogy y ∈ C legfeljebb $5n^{(1+\alpha)/2}$=O($n^{(1+\alpha)/2}$) y-ra teljesülhet. A három kifejezést összegezve azt kapjuk, hogy a fontos csomópontok száma ténylegesen O($n^{1-\alpha}logn + n^{(1+\alpha)/2}$), ami megegyezik az L. J. Cowen által definiált fontos csomópontok számával. Ezzel bebizonyítottuk, hogy az újonnan felvett csomópontok nem növelték meg a csomópontok számának nagyságrendjét. ▉

Az új fontos csomópontok felvétele után frissítenünk kell a másodlagos fontos csomópontok listáját is úgy, hogy ha a másodlagos fontos csomópontból a célba vezető útra került egy új fontos csomópont, akkor legyen az a fontos csomópont az adott célponthoz tartozó másodlagos fontos csomópont. Amennyiben több fontos csomópont is került az útra, akkor a célponthoz legközelebb lévőt választjuk.

LEMMA:

A fontos csomópontokat és a másodlagos fontos csomópontokat kivéve minden pont legfeljebb O($n^{(1+\alpha)/2}$) eredeti környezetben és legfeljebb O($n^{(1+\alpha)/2}$) darab másodlagos fontos csomóponttól a célig vezető úton van rajta.

BIZONYÍTÁS:

A tétel első részét L. J. Cowen bizonyította az eredeti útválasztó séma megkonstruálásánál. A második rész pedig következik abból, hogy ha egy csomópont több mint $n^{(1+\alpha)/2}$ másodlagos

helyettesítő úton van rajta, akkor azt a csomópontot felvettük a másodlagos fontos csomópontok listájára és ezért arra a pontra nem kell, hogy érvényes legyen a tétel állítása. (A másodlagos fontos csomópontokra ezért nem kell igaznak lennie a tétel állításának, mivel ott a portokat nem a lokális útválasztó tábla fogja tartalmazni, hanem a csomagok címe.) ▮

7.2 Az útválasztó séma által tárolt adatok

Először is el kell tárolni az összes olyan információt, amit az eredeti, a helyettesítő utakat nem kezelő sémában eltároltunk. Ezt követően minden pontban még el kell tárolni az összes újonnan felvett fontos csomópontokba vezető legrövidebb úthoz tartozó portot. Miután az összes fontos csomópontba vezető legrövidebb út meg lett határozva, minden fontos csomóponthoz vezető út portjához el kell tárolni egy másodlagos portot is ugyanúgy, mint ahogyan a naiv helyettesítő utas sémánál eltároltuk a másodlagos portokat. Ezen kívül még az összes olyan porthoz, ami azért lett eltárolva, mivel az eredeti környezeteknél benne volt a célpont környezetében szintén el kell tárolni egy másodlagos portot amennyiben a helyettesítő út végig az eredeti környezeten belül halad. Ellenkező esetekben pedig egy speciális értéket (például -1 vagy Null) ami azt mutatja, hogy a helyettesítő út elhagyja a környezetet. Ezen kívül a másodlagos csomópontból a célpontig vezető út élein el kell tárolni a megfelelő portokat úgy, hogy a másodlagos pontból a célpontba vezető út legyen diszjunkt az elsődleges csomóponttól a célpontba vezető úttal.

TÉTEL:

Minden csomópontban legfeljebb O($n^{1-\alpha}log^2 n + n^{(1+\alpha)/2}logn$) lokális tárigényre van szükség. (Ez megegyezik a L. J. Cowen által használt lokális tárkapacitással.)

BIZONYÍTÁS:

Minden csomópontban 2 port van tárolva az összes fontos csomóponthoz. Ennek a tárkapacitás igénye minden pontban O($n^{1-\alpha}log^2 n + n^{(1+\alpha)/2}logn$) mivel a Lemma szerint O($n^{1-\alpha}logn + n^{(1+\alpha)/2}$) fontos csomópont van és egy csomóponthoz O($logn$) információt kell eltárolni. Ezen kívül minden pont legfeljebb O($n^{(1+\alpha)/2}$) pont eredeti környezetében lehet benne a Lemma miatt, amihez még O($n^{(1+\alpha)/2}logn$) tárkapacitásra van szükség, mivel minden környezet miatt, amiben egy pont benne van O($logn$) mennyiségű adatot kell eltárolni az elsődleges és a másodlagos port miatt. A harmadik-féle adat a másodlagos fontos csomóponttól a célig vezető úthoz tartozó portokat tartalmazza. Ennél az adatnál is igaz, hogy minden pont legfeljebb O($n^{(1+\alpha)/2}$) úton lehet rajta, mivel különben fontos csomópontként lenne definiálva, ezért ebben az esetben is minden ponthoz legfeljebb O($n^{(1+\alpha)/2}logn$) lokális tárkapacitásra lehet szükség az előzőeken kívül. A három-féle adat méretét összegezve azt kapjuk, hogy minden pontban legfeljebb O($n^{1-\alpha}log^2 n + n^{(1+\alpha)/2}logn$) lokális tárkapacitásra van szükség, mivel egyéb adat tárolását a séma nem igényli. Ezzel bebizonyítottam, hogy a módosított séma nem igényel nagyságrendileg több tárolókapacitást, mint a helyettesítő utakat nem kezelő kiindulási verzió. ▮

7.3 Az útválasztó algoritmus

Ha a „Kötelező fontos csomópont” bit van bekapcsolva és jelenleg nem az elsődleges csomópontban vagyunk, akkor ugyanúgy kell működnie a csomópontokban az útválasztásnak, mint a naiv sémánál. Ebben az esetben a célpont az elsődleges fontos csomópont, ami a csomag címéből kiolvasható.

Ha a „Kötelező fontos csomópont” bit van bekapcsolva és jelenleg az elsődleges csomópontban vagyunk, akkor a „Kötelező fontos csomópont” bitet ki kell kapcsolni, majd az útválasztó algoritmust újra lefuttatni a megváltozott állapotra.

Ha a „Kötelező másodlagos helyettesítő út” bit be van kapcsolva, és a másodlagos helyettesítő úthoz tartozó él ép akkor azon kell továbbmenni. Amennyiben az az él nem ép, akkor a „Kötelező másodlagos helyettesítő út” bitet ki kell kapcsolni, a „Kötelező fontos csomópont” bitet be kell kapcsolni és meg kell cserélni az elsődleges és a másodlagos fontos csomóponthoz tartozó információkat. Ezt követően újra kell futtatni az útválasztó algoritmust a módosított feltételekre.

Ha egyik jelző bit sincsen bekapcsolva, azaz a csomag „Normál állapot”-ban van akkor számos eset lehetséges.

1. A célba vezető legrövidebb úthoz tartozó port ismert
   a. Ha az általa meghatározott él nem hibásodott meg, és a csomag sem abból az irányból érkezett akkor továbbítható az elsődleges porton.
   b. Ha az általa meghatározott él meghibásodott vagy a csomag abból az irányból érkezett, akkor, ha a másodlagos csomópont egy valós port érték, akkor arra kell továbbítani a csomagot. Egyébként pedig be kell kapcsolni a „Kötelező fontos csomópont” bitet és a „Kötelező másodlagos helyettesítő út” bitet, meg kell cserélni az elsődleges és a másodlagos fontos csomópontra vonatkozó adatokat, majd újra kell futtatni az útválasztó algoritmust a módosított feltételekre.
2. Jelenleg az elsődleges fontos csomópontban vagyunk
   a. Ha a címben az elsődleges csomóponthoz meghatározott él ép, akkor a csomagot azon az élen kell továbbítani.
   b. Ha a címben az elsődleges csomóponthoz meghatározott él megsérült, akkor a „Kötelező fontos csomópont” bitet és a „Kötelező másodlagos helyettesítő út” bitet be kell kapcsolni és az elsődleges és a másodlagos fontos csomóponthoz tartozó adatokat fel kell cserélni, majd újra kell futtatnia az útválasztó algoritmust a módosított feltételekre.
3. A célba vezető legrövidebb út nem ismert, akkor ugyanúgy kell eljárni, mint a naiv útválasztó sémánál.

Formálisan:

```
Útválasztás():
Ha(„Kötelező fontos csomópont”)
    Ha(Elsődleges fontos csomópontban vagyunk)
        „Kötelező fontos csomópont” = 0
        Útválasztás()
    Egyébként
        Naiv(Elsődleges fontos csomópont)
Egyébként Ha(„Kötelező másodlagos helyettesítő út”)
    Ha(Másodlagos helyettesítő úthoz tartozó él ép)
        Továbbítás a másodlagos helyettesítő úthoz tartozó élen
    Egyébként
        „Kötelező másodlagos helyettesítő út” = 0
        „Kötelező fontos csomópont” = 1
        Csere(Elsődleges fontos csomópont, Másodlagos fontos csomópont)
        Útválasztás()
Egyébként
    Ha(Célhoz vezető legrövidebb út ismert)
        Ha(Célhoz vezető legrövidebb út éle ép)
            Továbbítás a célhoz vezető legrövidebb út élén
        Egyébként Ha(Célhoz vezető helyettesítő út valós éték)
            Továbbítás a célhoz vezető helyettesítő út élén
        Egyébként
            „Kötelező másodlagos helyettesítő út” = 1
            „Kötelező fontos csomópont” = 1
            Csere(Elsődleges fontos csomópont, Másodlagos fontos csomópont)
            Útválasztás()
    Egyébként Ha(Elsődleges fontos csomópontban vagyunk)
        Ha(Címben az elsődleges fontos csomóponthoz tartozó él ép)
            Továbbítás az elsődleges fontos csomóponthoz tartozó élen
        Egyébként
            „Kötelező másodlagos helyettesítő út” = 1
            „Kötelező fontos csomópont” = 1
            Csere(Elsődleges fontos csomópont, Másodlagos fontos csomópont)
            Útválasztás()
    Egyébként
        Naiv(Elsődleges fontos csomópont)
```

Ahol az Útválasztás() utasítás az útválasztó algoritmus újrakezdését jelenti, a Naiv(cél) utasítás pedig a naiv útválasztó sémán használt algoritmus lefuttatását a paraméterben specifikált célpontba.

LEMMA:

A két jelzőbit értékén csak az alábbi átmenetek lehetségesek: (0, 0) => (1, 1) => (0, 1) => (1, 0) => (0, 0) => … ahol a két jelzőbit pár a („Kötelező fontos csomópont”, „Kötelező másodlagos helyettesítő út”) párnak felel meg.

BIZONYÍTÁS:

A bizonyítás az útválasztó algoritmusban szereplő esetek megvizsgálásából azonnal látszik. ▉

TÉTEL:

A fenti definíció szerinti útválasztó tábla minden esetben eljuttatja a csomagot a céljáig és amennyiben a gráfban egyik él sem hibásodott meg, akkor a csomag által használt út legfeljebb háromszor olyan hosszú lesz, mint a kiindulópont és a célpont közötti legrövidebb út.

BIZONYÍTÁS:

A tétel második része következik abból, hogy amennyiben egyik él sem hibásodott meg a gráfban, akkor a csomag végig az elsődleges portok mentén halad, ami definíció miatt a L. J. Cowen által kidolgozott séma szerint eltárolt adatok, esetlegesen valamilyen plusz kibővítéssel. Ezért a L. J. Cowen által leírt bizonyítások alapján a használt út ténylegesen legfeljebb háromszor olyan hosszú lesz, mint a legrövidebb lehetséges út.

Az első rész bizonyítását bontsuk fel több részre a hibás él elhelyezkedésének függvényében figyelembe véve azt, hogy amennyiben a csomag a fontos csomópontig vezető út folyamán belép a célpont környezetébe, akkor onnantól a környezet által tárolt adatokat fogja használni a további útválasztáshoz.

Az első eset legyen az, ha a hibás él a célponthoz tarozó fontos csomópontig vezető úton helyezkedik el. Ez azt jelenti, hogy a hibás él egyik végpontja (amelyik felől a csomag érkezik) nincsen benne a célpont környezetében. Ebben az esetben, a hibás él végpontjában is (mint minden más pontban) el van tárolva az össze fontos csomópontba vezető legrövidebb úthoz, illetve a helyettesítő úthoz tartozó port. Ezeknek az információknak a felhasználásával a naiv sémára bemutatott helyettesítő utakat kezelő algoritmus segítségével a csomag eljuttatható a fontos csomópontig. Onnan már az eredeti útválasztó algoritmus segítségével a csomag eljuttatható a célpontig, mivel a fontos csomóponttól végig a célpont környezetén belül fog haladni ahol nem találkozhat ismételten a hibás éllel, mivel feltettük, hogy annak egyik vége a célpont környezetén kívül van. Ugyanez az eset áll fent akkor is, ha a csomag a fontos csomópontba való eljutás előtt lép be a célpont környezetébe. (Megjegyzés: Ebben az esetben a csomag végig normál állapotban van.)

A második eset legyen az, ha a fontos csomóponttól a célpont környezetébe vezető él hibásodott meg és a célponthoz tartozó fontos csomópont nincsen benne a célpont környezetében. (Ez azt jelent, hogy a vizsgált él egyik végpontja a célpont környezetén kívül a másik pedig belül található.) Ebben az esetben a csomagot el lehet juttatni a másodlagos fontos csomópontba ugyanazzal a technikával, mint amit az első esetben használtunk azzal a különbséggel, hogy a csomag biztosan el fog jutni a másodlagos fontos csomópontig, mivel a

„Kötelező fontos csomópont” bit be lett kapcsolva. A másodlagos fontos csomópont elérését követően a csomag a másodlagos helyettesítő úton fog tovább haladni, ami a definíció miatt éldiszjunkt az elsődleges fontos csomóponttól a célba vezető úttól, ezért a hibás éllel a másodlagos fontos csomópont elérése után nem találkozhat a csomag. Ezért ebben az esetben is biztosan célba jut a csomag. (Megjegyzés: Ebben az esetben a csomag a (0, 0) az (1, 1) majd a (0, 1) állapotban lesz.)

Harmadik és egyben utolsó eset az, ha a hibás él mindkét vége a célpont környezetén belül van. Ebben az esetben különböztessünk meg két különböző alesetet a szerint, hogy a hibás él melyik utakon van rajta.

Az első aleset legyen az, ha a hibás él nincs rajta a másodlagos fontos csomópontból a célpontba vezető másodlagos helyettesítő úton. Ekkor a hibás éllel történő találkozás után a csomag az (1, 1) állapotba kerül, aminek következtében biztosan el fog jutni a másodlagos fontos csomópontba, ahonnan pedig a másodlagos helyettesítő út mentén a célpontba. A másodlagos fontos csomópontig a naiv helyettesítő utas séma működése miatt fog a csomag biztosan eljutni, onnan pedig a célpontig vezető úton nem találkozhat a hibás éllel, mivel feltettük, hogy a hibás él nincs rajta a másodlagos helyettesítő úton. (Megjegyzés: Ebben az esetben a csomag a (0, 0) az (1, 1) majd a (0, 1) állapotban lesz.)

A második aleset ezek után legyen az, ha a hibás él rajta van a másodlagos fontos csomóponttól a célpontig vezető másodlagos helyettesítő úton. Ebben az esetben az első alesettel megegyezően a csomag el fog jutni a másodlagos fontos csomópontig, majd onnan el fog indulni a másodlagos helyettesítő úton ahol egy hibás éllel fog találkozni. Ekkor a bitek megváltoztatásának következtében, a csomag továbbítva lesz az elsődleges fontos csomópontba ahova biztosan el fog jutni a naiv helyettesítő utas séma működése miatt. Majd az elsődleges fontos csomóponttól a legrövidebb úton el fog jutni a célpontig, mivel azon az úton nem lehet a hibás él, mert az a másodlagos helyettesítő út mentén van ami diszjunkt az elsődleges fontos csomóponttól a célba vezető legrövidebb úttól. (Megjegyzés: Ebben az esetben a csomag a (0, 0) az (1, 1) a (0, 1) az (1, 0) majd ismét a (0, 0) állapotban lesz.)

Ez a három eset lefedi a hibás él összes lehetséges elhelyezkedését, ezért a három eset megvizsgálásával bebizonyítottam a tételt, ami kimondja, hogy az útválasztó séma működőképes és teljesíti a vele szemben támasztott követelményeinket. ▮

7.4 Az útválasztó tábla generálása

Először is végre kell hajtani a Lewnor J. Cowen által használt útválasztó séma generálását végző algoritmust és minden pontban el kell tárolni az általa letárolt adatokat. (Ezzel biztosítjuk a megfelelő működést abban az esetben, ha egyik él sem hibásodott meg a gráfban.)

Ezt követően minden csomóponthoz meg kell határozni a másodlagos fontos csomópontot. Ehhez minden x pontból le kell futtatni egy szélességi bejárást úgy, hogy előtte el kell távolítani az elsődleges fontos csomópontból a célpontba vezető út összes élét a gráfból. Amennyiben a szélességi bejárás során először megtalált fontos csomópont legfeljebb $4n^{\alpha}$ él távolságra van az x ponttól akkor legyen ez a csomópont az x ponthoz tartozó fontos csomópont. Amennyiben minden fontos csomópont messzebb van x-től mint $4n^{\alpha}$ él, akkor az x-ből az xl-be (ahol xl az x-hez tartozó fontos csomópont) vezető út mentén (amit a szélességi bejárás során találtunk meg) keressük meg az n^{α}-

adik pontot amit jelöljön u. Ekkor P(u, xl) vissza fog csatlakozni a P(x, xl) útba az eredeti gráfban. Ezt a visszacsatlakozási pontot (ami rajta van a P(x, xl) úton) vegyük fel a fontos csomópontok listájára, és legyen ez a fontos csomópont az x-hez tartozó másodlagos fontos csomópont. Ennek a résznek a futásideje így O($nm + n^2 + n^{1+\alpha}$) mivel összesen n darab szélességi bejárásra van szükség és mindegyik bejárásnál legfeljebb O(n^{α}) él eltávolítása szükséges.

Ezek után, amelyik csomóponton több mint $n^{(1+\alpha)/2}$ másodlagos fontos csomópontot a célponttal összekötő út megy keresztül, akkor azt ki kell nevezni fontos csomópontnak, majd minden pontban el kell tárolni az előző részben és a most felvett új fontos csomópontokhoz vezető legrövidebb úthoz tartozó portot. Következő lépésként meg kell határozni minden pontból a fontos csomópontokba vezető úthoz tartozó helyettesítő utat ugyanazzal az algoritmussal, amit a naiv helyettesítő utas séma esetén használtunk. Majd minden pont környezetére, mint részgráfra tekintve meg kell határozni a helyettesítő utakat, amik a környezet középpontjába tartanak. Az ezekhez tartozó portokat el kell tárolni, mint másodlagos portokat. Ez ugyanazzal a technikával megoldható, mint amivel a másodlagos portokat meghatároztuk a naiv helyettesítő utas útválasztó séma esetén. Amennyiben valamelyik pontból nem megy helyettesítő út a célpontba akkor ott a helyettesítő út portja helyett egy speciális jelet (-1 vagy Null) kell eltárolni a port sorszáma helyett. Utolsó lépésként minden pontnál a másodlagos fontos csomópontból a célpontba vezető úton lévő pontokban a másodlagos fontos csomópontok kivételével el kell tárolni a másodlagos helyettesítő úthoz tartozó portot, ami már meghatározásra került a másodlagos fontos csomópontok megkeresése során. Amennyiben a másodlagos fontos csomóponttól a célpontig vezető út mentén új fontos csomópontok kerültek felvételre, akkor természetesen csak az azon út mentén elért legközelebbi csomópontig kell eltárolni a másodlagos helyettesítő úthoz tartozó portokat.

Formálisan:

Legyen L a fontos csomópontok halmaza, K_v a v pont környezete, l_v a v-hez tartozó elsődleges fontos csomópont, $l2_v$ a v-hez tartozó másodlagos fontos csomópont, $P2_v$ az $l2_v$ és a v közötti út

Útválasztó tábla generálása:
(L, l_v, K_v) = L. J. Cowen -féle útválasztó tábla generáló algoritmus()
Minden v ∈ V
    Szélességi bejárás v-ből G=(V, E \ P(l_v, v))-n
    Ha(először elért fontos csomópont távolsága kisebb mint $4n^{\alpha}$ él)
        $l2_v$ legyen az először elért fontos csomópont
    Egyébként
        u legyen P(v, l_v, G\P(l_v, v)) út n^{α}-adik pontja
        $l2_v$ legyen P(u, l_v) és P(v, l_v) első közös pontja
        C2 = C2 U {$l2_v$}
Minden v ∈ V
    R2v <- {y|v E $P2_y$}
C2 = C2 U {v | |$R2_v$| > $n^{(1+\alpha)/2}$}
Minden l ∈ C2
    Dijkstra-algoritmus l kiindulási ponttal
    Minden u ∈ V
        Tároljuk el u-ban (l, e_u(l))-t
Minden v ∈ V
    Ha létezik u ∈ C2 és u ∈ $P2_v$ /* Ha több ilyen u is létezik akkor a legközelebbi */
        $l2_v$ = u;
        $P2_v$ = $P2_v$ $l2_v$ utáni része
L = L U C2
Minden e=(v_1, v_2) ∈ E
    T = Dijkstra(v_1-ből, G\e-re)
    Leghosszabb helyettesítő utak frissítése (T) /* Ugyanaz, mint a naiv séma esetén */
    T = Dijkstra(v_2-ből, G\e-re)
    Leghosszabb helyettesítő utak frissítése (T)
Minden v_1 ∈ V, v_2 E L\v1
    Tároljuk el v_1-ben leghosszabb_helyettesítő_út(v_1, v_2) út portját másodlagos portként
Minden v ∈ V
    Minden e=(v_1, v_2) ∈ K_v
        T = Dijkstra(v_1-ből, K_v\e-re), Nem elérhető pont távolsága legyen végtelen
        Leghosszabb helyettesítő utak frissítése(T)
        T = Dijkstra(v2-ből, K_v\e-re), Nem elérhető pont távolsága legyen végtelen
        Leghosszabb helyettesítő utak frissítése(T)
    Minden v_1 ∈ K_v, v2 ∈ K_v\v_1
        Ha(leghosszabb_helyettesítő_út(v_1, v_2) nem végtelen)
            Tároljuk el v_1-ben leghosszabb_helyettesítő_út(v_1, v_2) portját másodlagos portként
        Egyébként
            Tároljunk el v1-ben -1-et másodlagos portként
Minden v ∈ V, u ∈ $P2_v$\$l2_v$
    Tároljuk el u-ban a $P2_v$ u pont utáni pontjába vezető élhez tartozó portot

7.5 A címek meghatározása

A címek első része megegyezik a L. J. Cowen által használt címekkel, ezért az a rész az általa használt algoritmus segítségével meghatározható. A kiegészítések meghatározásához pedig fel kell használnunk az útválasztó tábla legenerálása közben meghatározott másodlagos fontos csomópontokat, illetve a másodlagos fontos csomópontokból a célpontba vezető út első éléhez tartozó portot. Mind a két adat nagyon könnyen kinyerhető az útválasztó táblákat generáló algoritmustól, így a címek meghatározásához nincsen szükség új algoritmus kifejlesztésére.

8 Elméleti összefoglalás

A dolgozatnak az eddigi részében bemutattam, hogy milyen útválasztó sémák illetve milyen helyettesítő út keresési eljárások lettek kidolgozva a múltban. Ezeket az algoritmusokat sikerült úgy továbbfejlesztenem, hogy működjenek akkor is, ha a hálózatban legfeljebb egy él meghibásodik, és ezen kívül teljesítsék a teljesítménybeli elvárásaimat is. Azaz hibátlan hálózat esetén a működésük megegyezik a kiindulási algoritmus működésével és sem a lokális tárigény sem a csomagok címének hosszának nagyságrendje nem nőtt meg a változtatás hatására.

A témában további elméleti kutatásra még nagyon sok területen van lehetőség. Érdekes kérdés lehet akár az általam kidolgozott sémára, akár egy másik sémára bebizonyítani, hogy a helyettesítő utak hossza legfeljebb hányszorosa lehet az optimális, a kiesett élet elkerülő helyettesítő út hosszának. Másik nyitott téma, hogy az általam bemutatott sémát hogyan lehet alkalmazni L. J. Cowen útválasztó sémájához hasonló, viszont annál fejlettebb sémákra, akár olyanokra is, amik kézfogásosan működnek. A harmadik érdekes kutatási iránynak az ígérkezik, ha ugyanezt a problémát vizsgáljuk meg arra az esetre, ha nem egy él, hanem több él, vagy egy, illetve több csomópont eshet ki a hálózatból.

9 Szimulációk

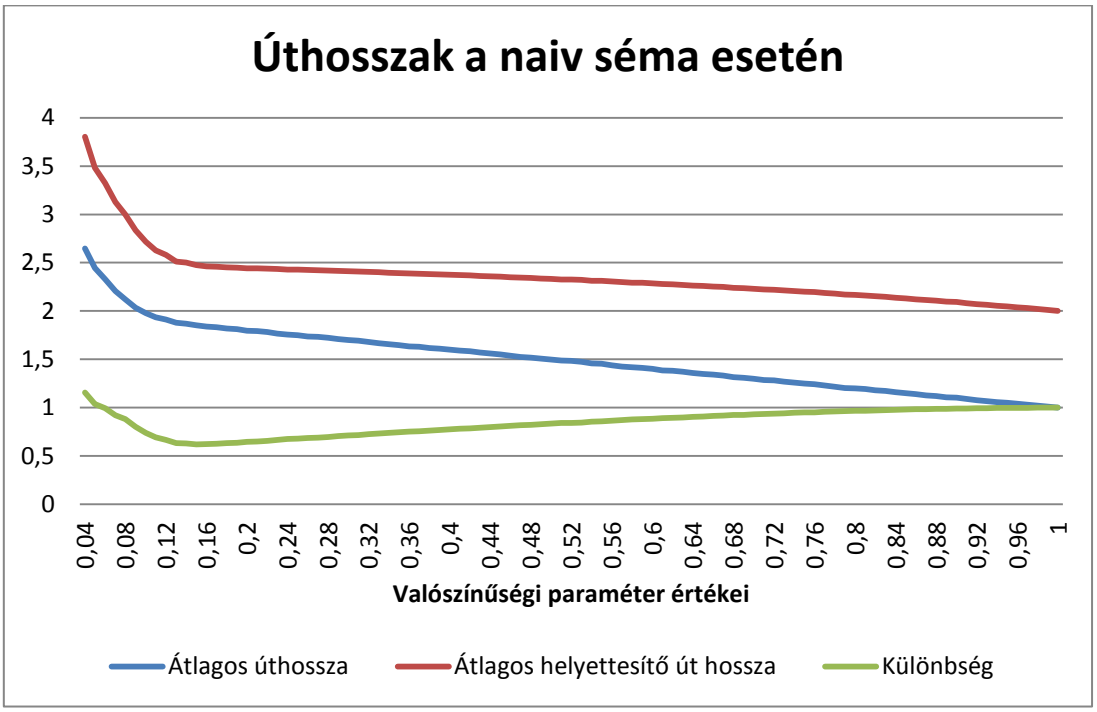
A dolgozat első részében elméleti szempontból vizsgáltam meg, hogy milyen útválasztó sémák léteznek és továbbfejlesztettem őket arra a szintre, hogy képesek legyenek a csomagok továbbítására abban az esetben is, ha van egy hibás él a hálózatban. Ezekre a sémákra nagyon sok felső korlát be van már bizonyítva, és az újonnan kidolgozott sémákra is bizonyítottam felső korlátokat. Ebben a részben a sémákon szimulációkat fogok elvégezni, hogy a paraméterek várható értékét meg tudjam határozni, illetve a sémák gyakorlati implementációjával (C++ környezetben) meg lehessen figyelni a sémák működését a különböző esetekben.

9.1 Véletlen gráfok

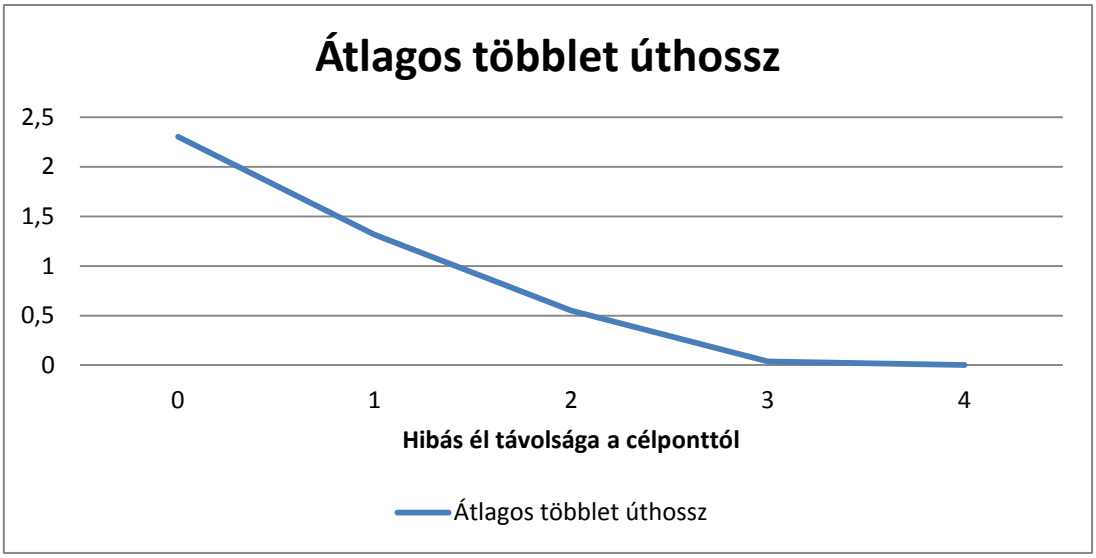
A szimulációkhoz szükség van véletlenszerűen generált gráfokra. Az elmúlt időszakban több különböző véletlen gráf séma lett kifejlesztve. Ezek közül én az Erdős-Rényi [11] modell valószínűségen alapuló változatát fogom használni az egyszerűség kedvéért kizárólag súlyozatlan eseteke. Az Erdős-Rényi modell valószínűségen alapuló változatának lényege, hogy minden élt azonos P valószínűséggel adunk hozzá a gráfhoz. Az Erdős-Rényi gráf nagy valószínűséggel összefüggő ha $\mathrm{P} > (1+\varepsilon)logn/n$, és ebben az esetben nagy valószínűséggel kétszeresen is összefüggő lesz, amire szükségünk van azért, hogy mindig legyen helyettesítő út a gráfban.

9.2 Naiv útválasztó séma vizsgálata

A naiv útválasztó sémánál szimulációk segítségével megvizsgáltam, hogy az Erdős-Rényi modellben szereplő p valószínűségi paraméter különböző értékeinél átlagosan milyen hosszúak lesznek a legrövidebb utak, illetve az általam kidolgozott séma által használt helyettesítő utak. A modell kis világ tulajdonsága (bármely két pont viszonylag közel van egymáshoz) tükröződött a szimulációs eredményekben, ahol azt lehet megfigyelni, hogy a helyettesítő utak átlagosan csak kicsivel hosszabbak, mint a legrövidebb utak. Ez leginkább a közepes sűrűségű gráfokra igaz, mivel a ritka gráfok esetén gyakrabban kell nagy kerülőt tenni a csomagnak a helyettesítő úton, nagyon sűrű gráfok esetén pedig a legrövidebb utak nagy része csak egy él hosszú, amihez ennek ellenére legalább kettő hosszú helyettesítő út tartozik, mivel a gráf egyszerű volt. (A vizsgálatot 250 pontú, súlyozatlan és irányítatlan gráfokkal végeztem el.)

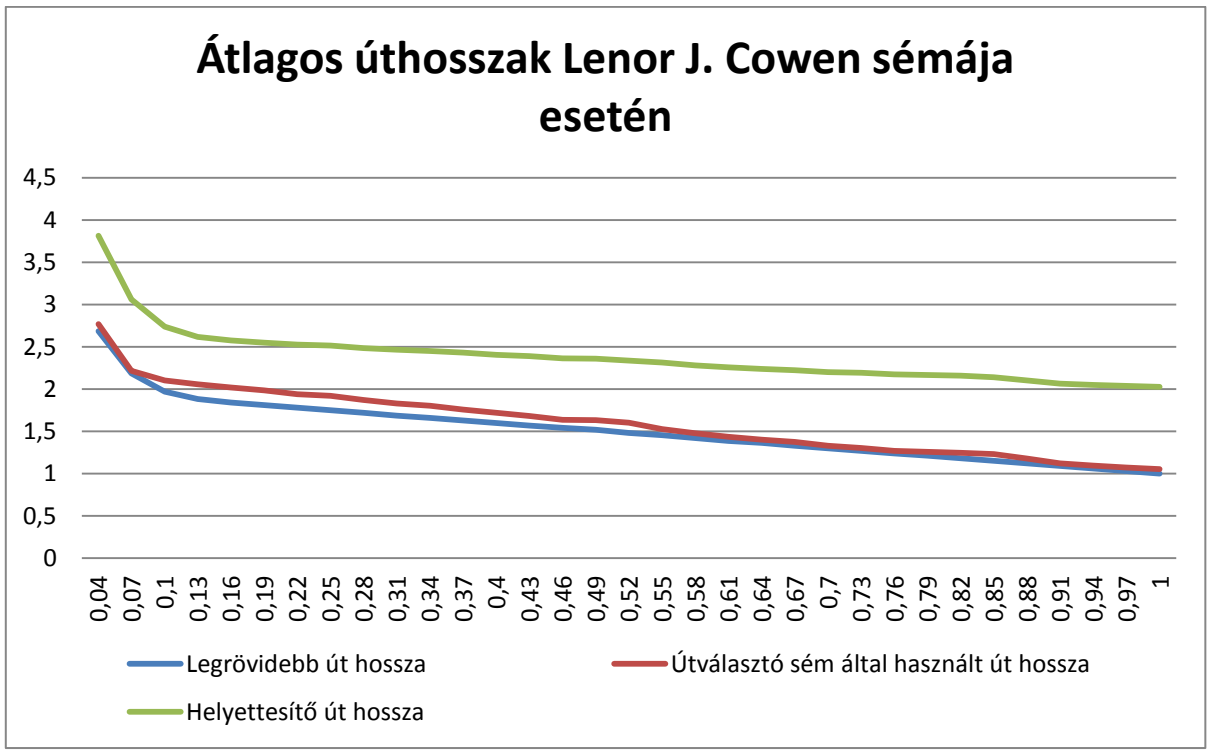


Ezen kívül megvizsgáltam, hogy hogyan függ a helyettesítő út hossza attól, hogy a hibás él a legrövidebb út melyik részén helyezkedik el. Ennek a vizsgálatnak elsősorban ritka gráfokra van értelme, mivel a sűrű gráfoknál az úthosszak túlságosan rövidek, ezért nem igazán lehet megkülönböztetni az út korai, illetve késői szakaszában bekövetkezett hibák hatását a helyettesítő utak hosszára. A vizsgálat során azt tapasztaltam, hogy minél közelebb van a hibás él a célponthoz, annál nagyobb a helyettesítő út hossza a legrövidebb úthoz képest. Ennek az az oka, hogy amennyiben a hibás él a legrövidebb út korai szakaszán van, akkor a helyettesítő út hamar vissza tud térni az éppen aktuális pontból a célpontba tartó legrövidebb útra, ami irányítatlan és súlyozatlan gráfoknál gyakran azt eredményezi, hogy a helyettesítő út hossza megegyezik a legrövidebb út hosszával. Ezzel szemben, ha a hibás él közel van a célponthoz (például az út utolsó éle), akkor szinte mindig hosszabb a helyettesítő út, mint a legrövidebb út, mivel ekkor viszonylag sokáig kell a csomagnak másodlagos portok mentén haladnia addig, amíg az aktuális pontból a célba menő legrövidebb út már másik irányba fog haladni, mint ahonnan a csomag a helyettesítő úton érkezett.



9.3 Lenor J. Cowne útválasztó sémájának

L. J. Cowen bebizonyította, hogy a sémája legfeljebb 3-szor olyan hosszú úton továbbítja a csomagokat, mint a legrövidebb út, viszont azzal a kérdéssel nem foglalkozott, hogy egy véletlen gráfban milyen hosszú úton továbbítja az általa kidolgozott séma átlagosan a csomagokat. Ennek a megvizsgálására - ugyanazokkal a feltételekkel, mint a legelső szimulációnál (250 pontú, súlyozatlan Erdős-Rényi gráf, különböző valószínűségi paraméterekkel) - végeztem egy szimulációt. A szimuláció során látszott, hogy legrosszabb esetben tényleg 3-szor olyan hosszú úton küldi a rendszer a csomagot, mint a legrövidebb út. Az átlagos hiba értéke pedig a vizsgált gráfokon nagyon kicsi (kisebb mint 1,1) volt, ami elsősorban az Erdős-Rényi modell kis-világ tulajdonságának köszönhető, mivel a legtöbb esetben a kiindulási pont benne van a célpont környezetében vagy szomszédos vele. Ugyanennek a következménye az is, hogy a helyettesítő utak erre a sémára is csak nagyon kevéssel hosszabbak, mint a kiindulási séma által használt utak.



9.4 Összefoglalás

A szimulációs eredményekből látható, hogy annak ellenére, hogy nem bizonyítottam be sem elméleti felsőkorlátot, sem várható értéket a helyettesítő utak hosszára vonatkozóan, a szimulációk során vizsgált gráfokban az átlagos helyettesítő út hosszak csak kis mértékben nagyobbak, mint a helyettesítő utak kezelésére nem képes sémák által használt utak hossza. Érdekes kérdés lehet ugyanezeknek a szimulációknak a lefuttatása más gráf osztályokra is, mivel amennyiben a kis-világ tulajdonság nem teljesül a gráfra, akkor sokkal jobban látszhatnak a különbségek az úthosszak között. Nem kis-világ tulajdonságú gráfokra emellett meg lehetne vizsgálni, hogy az L. J. Cowen sémáját alapul vevő helyettesítő utas sémában hogyan alakul a csomagok által használt helyettesítő utak hossza annak függvényében, hogy a hibás él az útvonal melyik szakaszán helyezkedik el (fontos csomópont előtt, közvetlenül utána, vagy a környezet belsejében)

10 Irodalomjegyzék

[1] J. Hershberger and S. Suri. 2001. Vickrey Prices and Shortest Paths: What is an Edge Worth?. In Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS '01). IEEE Computer Society, Washington, DC, USA, 252-259.

[2] J. Hershberger and S. Suri. 2002. Erratum to "Vickrey Pricing and Shortest Paths: What is an Edge Worth?". In Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS '02). IEEE Computer Society, Washington, DC, USA, 809.

[3] W. Vickrey (1961). Counter speculation, Auctions and Competitive Sealed Tenders. Journal of Finance, vol. 16, pp. 8-37.

[4] L. Roditty and U. Zwick. 2012. Replacement paths and k simple shortest paths in unweighted directed graphs. ACM Trans. Algorithms 8, 4, Article 33 (October 2012), 1-11

[5] A. Bernstein. 2010. A nearly optimal algorithm for approximating replacement paths and k shortest simple paths in general graphs. In Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '10). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 742-755.

[6] D. B. Johnson. 1977. Efficient Algorithms for Shortest Paths in Sparse Networks. J. ACM 24, 1 (January 1977), 1-13.

[7] L. J. Cowen. 1999. Compact routing with minimum stretch. In Proceedings of the tenth annual ACM-SIAM Symposium on Discrete Algorithms (SODA '99). Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 255-260.

[8] D. Dor, S. Halperin, and U. Zwick. 1996. All pairs almost shortest paths. In Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS '96). IEEE Computer Society, Washington, DC, USA, 452-461.

[9] B. Awerbuch, A. Bar-Noy, N. Linial, and D. Peleg. 1989. Compact distributed data structures for adaptive routing. In Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC '89), D. S. Johnson (Ed.). ACM, New York, NY, USA, 479-489.

[10] L. Lovász, On the ratio of optimal integral and fractional covers, Discrete Mathematics, Volume 13, Issue 4, 1975, Pages 383-390

[11] P. Erdős and A. Rényi (1960) On the evolution of random graphs. MTA Mat. Kut. Int. Kozl. 5 17-61