Eötvös Loránd Tudományegyetem

Természettudományi Kar

Kelen Domokos Miklós

Útvonaltervezés bizonytalan adatokkal

BSc Szakdolgozat

Témavezető:

Jüttner Alpár

Operációkutatás Tanszék

Budapest, 2013

Köszönetnyilvánítás

Szeretnék köszönetet mondani témavezetőmnek, Jüttner Alpárnak azért, hogy tanácsaival és útmutatásával segített ezen dolgozat elkészítésében. Köszönettel tartozom továbbá mindenki másnak is, aki támogatásával vagy segítségével hozzájárult a munkámhoz.

Tartalomjegyzék

Bevezetés

Alapvető feladat a matematikában egy úthálózaton a legrövidebb utak, azaz egy pontból egy másikba jutás valamilyen feltétel szerint optimális módjának keresése. Az úthálózatokat hagyományosan gráfokkal modellezzük, legrövidebb útnak pedig azt tekintjük, amire a felhasznált élekhez rendelt költségek összege minimális.

A feladat sok esetben megoldottnak tekinthető. A fent említett modellben például ismerünk hatékony algoritmusokat pozitív élköltségek esetén (Dijkstra-algoritmus), tetszőleges valós élköltségek esetén (Bellman-Ford algoritmus), stb.

Ezek a modellek azonban – mint ahogy gyakorlatilag minden matematikai modell – egyszerűsítik a valóságot. Egy ilyen egyszerűsítés például az, hogy egy útszakaszon az annak megtételéhez szükséges időt mint költséget egy konkrét értéknek tekinti, holott a valóságban a legritkább esetben fordul elő, hogy pontosan ismerjük előre. Sokkal általánosabb, hogy az egyes útszakaszokon az áthaladás idejét csak megbecsülni tudjuk, azaz – például korábbi tapasztalatok alapján – valószínűsítjük annak a hosszát.

Maga az optimalitási feltétel sem nyilvánvaló ha bizonytalan az egyes útszakaszok megtételéhez szükséges idő. Kereshetjük például azt az utat, amin a végighaladási idő várható értéke minimális. Ez a feladat gyakorlatilag megegyezik a klasszikussal, az egyes éleken az áthaladási idő várható értékét véve költségfüggvénynek. Ezzel viszont nem vesszük figyelembe a áthaladási idők szórását. A való életben sokszor előfordul, hogy valaki inkább „lassan de biztosan” szeretne eljutni a céljához, mint gyorsan, de annak a kockáztatásával, hogy az utazás (kis eséllyel ugyan, de) hosszúra nyúlhat.

Végül, de nem utolsó sorban a megoldás sem kell, hogy egy előre meghatározott utiterv legyen. Mint ezt látni fogjuk, előfordulhat olyan eset, amikor egy bizonyos útszakasz megtétele után érdemesebb visszafordulni, mint az eredeti úton továbbhaladni. Az ilyen tipusú megoldások megvalósítása a modern navigációs rendszerek (GPS eszközök, okostelefonok, stb.) elterjedésével már megvalósítható a mindennapokban.

A dolgozatban az úgynevezett *Stochastic on time arrival* vagy *sztochasztikus időben érkezési problémát* és annak megoldására született algoritmusokat, módszereket járjuk körül. Az 1. fejezetben ismertetjük a feladatot valamint annak egy közelítő megoldását adó algoritmust. A 2. fejezetben egy, a feladatot elméleti, valamint diszkrét modellben pontosan megoldó algoritmust mutatunk be. A 3. fejezetben az egyes algoritmusok hatékonyságának növelésére használható gyors konvolúciós algoritmusokat ismertetjük.

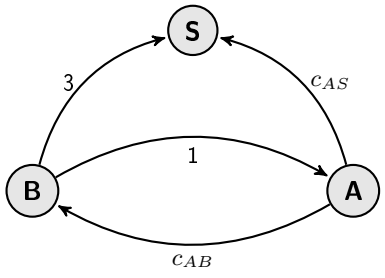
A 4. fejezetben néhány kitekintést teszünk a feladattal kapcsolatban felmerülő egyéb kérdésekre, végül az 5. fejezetben különböző teszteket végzünk az egyes konvolúciós algoritmusok összehasonlítására.

1. fejezet

A SOTA probléma és a fokozatos közelítéses algoritmus

A sztochasztikus időben érkezési, vagy röviden *SOTA*[1] probléma alapfeladata a következő: adott egy úthálózat és egy utazó, aki valamely pontból szeretne eljutni egy másik pontba T időkeret alatt, minél nagyobb valószínűséggel. Az úthálózat tulajdonságai viszont *bizonytalanok*, azaz nem lehetséges előre tudni például egy utcán való áthaladáshoz szükséges időt, csak annak a valószínűségi eloszlását. Célunk egy olyan megoldás megtalálása, ami szerint haladva az utazó az időben érkezés esélyét maximalizálja. Mivel a rendelkezésre álló információ mennyisége az út során változik (addig csak valószínűsíthető adatok vesznek fel konkrét értéket), ezért a determinisztikus esetekkel ellentétben nem lehetséges egy optimális útvonalat a priori meghatározni. Mint ezt az 1.0.1. példában látni fogjuk, a bejárt útvonal egy gráfban nem is feltétlenül *út*, hanem sok esetben *séta*. Az optimális megoldás amit keresünk tehát nem egy optimális út, hanem egy stratégia arra nézve, hogy milyen feltételek mellett merrefelé érdemes továbbhaladni.

1.0.1. Példa. [Samaranayake, Blandin és Bayen (2011)]
Nem feltételezhető, hogy minden élen csak egyszer haladunk végig. Tekintsük például az 1.1. ábrán látható irányított gráfot.



t	$P(c_{AS} = t)$	$P(c_{AB} = t)$
5	0,9	0
2	0	0,1
1	0,1	0,9

1.1. ábra. Gráf amin a bejárt útvonal nem feltétlenül út

A gráf négy éle közül kettőn az áthaladás bizonytalan kimenetelű. Az áthaladás idejét jelölő valószínűségi változók neve c_{AS} illetve c_{AB}. Ezek eloszlását szintén az 1.1. ábra mutatja. A utazó az A pontból

[1] Az angol *Stochastic on time arrival* rövidítése.

indul és az S pontba szeretne eljutni nem több mint $t = 4$ időegység alatt.

Ha az (A,S) élet veszi igénybe, az időben érkezés esélye 0,1 míg az (A,B) élen indulva legalább 0,9 (hiszen ha 1 időegység letelte után megérkezik B pontba, a (B,S) élet igénybe véve biztosan időben érkezik S-be).

Ha viszont megtette az (A,B) élet és az áthaladás ideje 2 időegység volt, akkor nincs értelme a (B,S) élen haladnia tovább, hiszen ekkor biztosan nem érkezik időben. A (B,A) élen visszafordulva viszont még mindig van 0.1 esélye S-be jutni a maradék 2 időegység alatt. Így tehát visszafordul és másodszor is érinti az A pontot.

1.1. A folytonos modell

Adott egy $G(V,A)$ irányított gráf ($|V| = n$, $|A| = m$), egy $S \in V$ pont és egy $T > 0$ időkeret. Egy *utazó* halad a gráfból egy pontjából az S pont felé és szeretne nem több mint T idő alatt megérkezni oda. Minden élen az áthaladás költségét egy-egy (egymástól független[2]) nemnegatív valószínűségi változó, c_{ij} modellezi.[3] Ezeket egy valós úthálózat esetében például mérési adatokból lehet becsülni.

A célunk minden $i \in V$ ponthoz egy olyan $q : V \times \mathbb{R} \to V \cup \{\emptyset\}$ függvény megtalálása, amire $q(i,t)$ vagy röviden $q_i(t)$ az i pontból való induláskor a továbbhaladás egy optimális irányát jelzi, azaz S-be t vagy annál kevesebb idő alatt való érkezés, azaz az *időben érkezés* valószínűségét maximalizálja. Ennek birtokában az utazó minden i kereszteződésnél az aktuálisan rendelkezésre álló időkerettől függően meghatározhatja, hogy merre érdemes továbbhaladnia.

Legyen $u : V \times \mathbb{R} \to [0,1]$, ahol $u(i,t)$ vagy röviden $u_i(t)$ az i pontból t időkerettel indulva az időben érkezés (optimális) valószínűsége, azaz az i-ből S-be való eljutás idejéhez, mint valószínűségi változóhoz tartozó eloszlásfüggvény.

Az c_{ij} változó eloszlásfüggvényét P_{ij}-vel jelölve annak az esélye, hogy az (i,j) élen továbbhaladva t időkerettel időben érkezünk S-be

$$u_i(t) = \int_{\mathbb{R}} u_j(t-\omega)\, P_{ij}(\mathrm{d}\omega).$$

Az itt használt képlet egy, a valószínűségszámításból ismert azonosság: ha X és Y két független valószínűségi változó, akkor $X+Y$ eloszlásfüggvénye a t helyen $\mathbf{E}(F_X(t-Y))$ (a konvencionális jelölésekkel élve). Itt most $Y = c_{ij}$, X pedig a j-ből S-be jutás idejét reprezentáló valószínűségi változó.

Természetesen egy pontból általában nem csak egy élen lehetséges továbbhaladni. Ha több i-ből kifele haladó él is adott, akkor – az optimalitás érdekében – a megfelelő integrálok pontonkénti maximumára van szüségünk:

$$u_i(t) = \max_{(i,j)\in A} \int_{\mathbb{R}} u_j(t-\omega)\, F_{ij}(\mathrm{d}\omega). \tag{1.1.1}$$

A továbbiakban feltételezzük, hogy minden c_{ij} valószínűségi változónak létezik sűrűségfüggvénye.

[2] A nem független esetről későbbi fejezetek során lesz szó.

[3] Vegyük észre, hogy ezzel a (formalizmus egyszerűsítése érdekében) implicit módon azt is feltételeztük, hogy adott i pontból adott j pontba minösszesen egy él haladhat.

Ezeket p_{ij}-vel jelölve 1.1.1 felírható a következőképpen is:

$$u_i(t) = \max_{(i,j)\in A} \int_0^t u_j(t-\omega) p_{ij}(\omega)\,\mathrm{d}\omega. \tag{1.1.2}$$

Vegyük észre, hogy az integrált nem az $\mathbb{R}$-en végezzük, hanem csak a [0,t] intervallumon. Ezt megtehetjük, hiszen az egyenlőség jobb oldalán álló szorzatban $\omega < 0$ esetén $p_{ij}(\omega) = 0$, $\omega > t$ esetén pedig $u_j(t-\omega) = 0$.

Az u_i függvényeket ismervén az optimális q_i megoldások meghatározása már magától értetődő: ezek t időpillanatban azt a j értéket kell, hogy felvegyék, amelyen az időben érkezés valószínűsége az 1.1.2 képletben a maximális, azaz

$$q_i(t) = \operatorname*{arg\,max}_{j\,:\,(i,j)\in A} \int_0^t u_j(t-\omega) p_{ij}(\omega)\,\mathrm{d}\omega. \tag{1.1.3}$$

Ez a formalizáció felteszi, hogy egy pontban sosem érdemes várakozni, azaz a várakozás nem növelheti az időben érkezés esélyeit. Ezt azért teheti meg, mert a fent leírt modellben az állítás az eloszlásfüggvények monoton növekedő jellege miatt nyilvánvalóan teljesül.

1.2. Fokozatos közelítéses algoritmus és ennek konvergenciája

A feladat keresett megoldásának megtalálásához tehát az u_i függvényeket kell megtalálnunk, amelyek optimalitására vonatkozó feltételek egy n egyenletből álló rendszer formájában állnak rendelkezésünkre: minden S-től eltérő i ponthoz a hozzá tartozó u_i függvényt egy, az 1.1.2. képletben megfogalmazott egyenlet ír le, az S-hez tartozó u_S függvényre pedig $u_S \equiv 1$ minden nemnegatív pontban. Fan és Nie (2006a) ezeknek a kiszámítására a következő algoritmus írják le:

1.2.1. Algoritmus. (Fokozatos közelítés) Kezdetben legyen (az iteráció számát felső indexben jelölve)

$$u_i^{(0)}(t) := \begin{cases} 0, & \text{ha } 0 \le t \text{ és } i \in V, i \ne S \\ 1, & \text{ha } 0 \le t \text{ és } i = S \end{cases},$$

majd rendre $k = 1,2,\ldots,K$-ra:

$$u_i^{(k+1)}(t) := \begin{cases} \max\limits_{(i,j)\in A} \int\limits_0^t p_{ij}(\omega) u_j^{(k)}(t-\omega)\,\mathrm{d}\omega, & \text{ha } 0 \le t \text{ és } i \in V, i \ne S \\ 1, & \text{ha } 0 \le t \text{ és } i = S \end{cases}.$$

A (közelítő) megoldást 1.1.3 szerint kapjuk meg, a K. iteráció eredményéből számolva.

1.2.2. Tétel. *Amennyiben az optimális megoldás szerint $0 \le T \in \mathbb{R}$ időkerettel bejárható leghosszabb séta hossza $k \in \mathbb{N}$, az algoritmus legfeljebb k lépésben konvergál a pontos megoldáshoz, $u_{i\ne S}$ függvények kezdeti értékeitől függetlenül.*

Bizonyítás. Azt szeretnénk belátni, hogy minden (i,t), $0 \leq t \leq T, i \in V$ párra $u_i(t)$ pontos értéket vesz fel legkésőbb a tételben megfogalmazott k. iteráció során.

Legyen $m(i,t)$ az i pontból t időkerettel az optimális megoldás szerint elindulva, majd az optimális megoldás szerint haladva a leghosszabb bejárható séta hossza, amennyiben az véges. A bizonyítás $m(i,t)$-re vonatkozó teljes indukcióval történik.

1. A $m(i,t) = 0$ esetre az állítás triviális: nulla hosszú optimális séta csak az S pontból lehetséges, u_S viszont már a $k = 0$ esetben is pontos.

2. Az indukciós lépéshez fel, hogy az állítás minden (i_1,t_1) párra igaz, ahol $m(i_1,t_1) = l_1$ és $l_1 < l_2$, és szeretnénk belátni, hogy ekkor minden (i_2,t_2) párra is igaz, ahol $m(i_2,t_2) = l_2$.

   Ha egy t_2 időpillanatban egy i_2 pontból az optimális megoldás szerint haladva bejárható séták maximális hossza l_2, akkor az $u_{i_2}(t_2)$ érték kiszámításához szükségképpen csak olyan $u_{i_1}(t_1)$ értékekre lehet szükségünk, ahol $m(i_1,t_1) < m(i_2,t_2)$, ugyanis minden i_2 pontból t_2 időkerettel induló séta egy élen való áthaladás után egy i_1 pontból t_1 időkerettel induló, egy éllel rövidebb sétában folytatódik, ezekre pedig már pontos az $u_{i_1}(t_1)$ érték az indukciós feltevés miatt.

Ezzel az állítást beláttuk. □

Az 1.2.1. algoritmussal azonban több probléma is adódik. A benne megjelenő integrálokat általános esetben nem lehetséges analitikusan kiszámolni: egyrészt nincs garancia arra, hogy a p_i sűrűségfüggvények „szép függvények" lennének (emlékezzünk vissza, ezek egy valós alkalmazás során például korábbi mérések statisztikáiból származhatnak), másrész magukat az $u_i^{(k)}$ függvényeket is több függvény pontonkénti maximumaként kapjuk meg. Valószínű tehát, hogy konkrét megvalósítás során valamilyen numerikus integrálási módszerre, vagy például a feladat idejének diszkretizálására kell hagyatkoznunk.

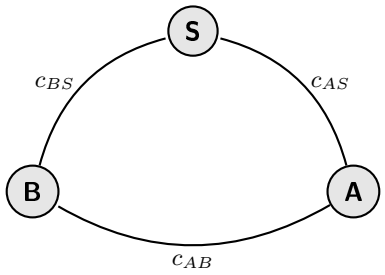
Probléma továbbá az is, hogy mint a bevezetőben láttuk, az optimális megoldás szerint bejárt útvonal nem feltétlenül *út* a gráfban, tehát nem triviális a maximális hosszát felülről becsülni a gráf méretéből. Mi több, mint ezt az 1.2.3. példa mutatja, a séta hosszára általános eseben nem is adható felső becslés az élek számának függvényében.

1.2.3. Példa.
Arra már láttunk példát korábban, hogy egy élet kétszer használunk (vagy irányított esetben ennek megfelelően oda-vissza haladunk két pont között). Konstruáljunk tehát először olyan példát, amelyben egy élet háromszor használunk.

Tekintsük a $G = (V,E)$ irányítatlan gráfot, ahol $V = \{A,B,S\}$ valamint $E = \{\{A,B\},\{B,S\},\{A,S\}\}$ (1.2. ábra). Legyen az éleken áthaladás idejét reprezentáló valószínűségi változók neve c_{AS}, c_{BS} valamint c_{AB}.

Az A pontból indul az utazó és S pontba szeretne eljutni nem több mint $t = 1000$ idő alatt. A konstrukció célja az, hogy olyan esetet demonstráljon, amikor az utazó háromszor egymás után végighalad az $\{A,B\}$ élen.

1. Az A pontból a $t = 1000$ időkerettel indulva ahhoz, hogy az utazó ne az $\{A,S\}$ élet vegye igénybe az szükséges, hogy $\{A,B\}$ él irányában nagyobb legyen az időben S-be érkezés esélye. Ennek elégséges



1.2. ábra. Gráf amin az utazó valamely esetben háromszor egymás után használja ugyanazt az élet

feltétele, ha $P(c_{AS} \leq 1000) \leq 0{,}2$ és $P(c_{AB} + c_{BS} \leq 1000) \geq 0{,}25$, azaz például a következők teljesülése:

$$P(c_{AS} = 1001) = 0{,}8$$
$$P(c_{AB} = 100) = 0{,}5$$
$$P(c_{BS} = 900) = 0{,}5.$$

2. Tegyük fel, hogy a fenti feltételek miatt az utazó az $\{A,B\}$ élt veszi igénybe és az áthaladás költsége 900 tehát a maradék időkeret $t = 100$ és az utazó a B pontban áll (ehhez természetesen az is szükséges, hogy $P(c_{AB} = 900) \neq 0$).

   Ahhoz hogy az utazó ne a $\{B,S\}$ élet vegye igénybe az szükséges, hogy $\{A,B\}$ él irányában nagyobb legyen az időben S-be érkezés esélye. Ennek elégséges feltétele, ha $P(c_{BS} \leq 100) \leq 0{,}02$ és $P(c_{AB} + c_{AS} \leq 100) \geq 0{,}025$, azaz például (a korábbiak mellett) a következők teljesülése:

$$P(c_{BS} = 101) = 0{,}48$$
$$P(c_{AB} = 10) = 0{,}05$$
$$P(c_{BS} = 90) = 0{,}05.$$

3. Tegyük fel, hogy a fenti feltételek miatt az utazó az $\{A,B\}$ élt veszi igénybe és az áthaladás költsége 90 tehát a maradék időkeret $t = 10$ és az utazó a B pontban áll (ehhez természetesen az is szükséges, hogy $P(c_{AB} = 90) \neq 0$).

   Ahhoz hogy az utazó ne az $\{A,S\}$ élet vegye igénybe az szükséges, hogy $\{A,B\}$ él irányában nagyobb legyen az időben S-be érkezés esélye. Ennek elégséges feltétele, ha $P(c_{AS} \leq 10) \leq 0{,}002$ és $P(c_{AB} + c_{BS} \leq 10) \geq 0{,}0025$, azaz például (a korábbiak mellett) a következők teljesülése:

$$P(c_{AS} = 11) = 0{,}148$$
$$P(c_{AB} = 1) = 0{,}005$$
$$P(c_{BS} = 9) = 0{,}005.$$

4. Az utazónk ekkor közvetlenül egymás után harmadszor is igénybe veszi az $\{A,B\}$ élt, tehát (mivel a fenti feltételek nem zárják ki egymást) a konstrukciónk sikerrel járt.

A feltételek összesítve az 1.1. táblázatban láthatóak, minden eloszlásban a valószínűségek 1-re kiegészítve. A nem definiált ϵ valamint δ értékekre az egyetlen megkötés, hogy elég kicsik, példánkban például legyen $\epsilon = \delta = 0{,}0001$.

t	$P(c_{AB} = t)$
900	ϵ
100	**0,5**
90	ϵ
10	**0,05**
1	**0,005**
δ	$0{,}445 - 2\epsilon$

t	$P(c_{AS} = t)$	$P(c_{AS} \leq t)$
1001	0,8	1
1000	0	**0,2**
90	**0,05**	0,2
11	0,148	0,15
10	0	**0,002**
δ	0,002	0,002

t	$P(c_{BS} = t)$	$P(c_{BS} \leq t)$
900	**0,5**	1
101	0,48	0,5
100	0	**0,02**
9	**0,005**	0,02
δ	0,015	0,015

1.1. táblázat. c_{AB}, c_{AS} és c_{BS} eloszlása

Könnyen látható, hogy a megkonstruált példához hasonló módon lehetséges tetszőleges k-ra is példát mutatni.

Szerencsésnek mondható viszont, hogy életszerű feladatokban az ilyen esetek előfordulása valószínűtlen. Egy valós úthálózaton ugyanis nem túl merész azt feltételezni, hogy minden utcán, azaz a gráfban minden élen létezik egy *minimális lehetséges áthaladási költség*, aminél gyorsabban nem lehetséges áthaladni az élen. Ekkor viszont kimondható a következő tétel:

1.2.4. Tétel. *Legyen adott minden (i,j) élen egy* minimális lehetséges áthaladási költség, *d_{ij}, amelyre $t \leq d_{ij}$ esetén $p_{ij}(t) = 0$, továbbá legyen $d = \min_{(i,j)\in A} d_{ij}$. Ekkor a gráf egy i tetszőleges pontjából, adott t időkeret esetén az optimális megoldás során bejárt séta hossza legfeljebb $\lceil \frac{t}{d} \rceil$.*

Az állítás bizonyítása triviális: tetszőleges élen áthaladva az aktuális t_0 időkeretből legalább d eltelik, tehát legfeljebb $\lceil \frac{t}{d} \rceil$ élen való áthaladás után az időkeret nullára (vagy az alá) csökken, tehát az időben érkezés nem lehetséges többé (kivéve természetesen azt az esetet, hogy pont S-be érkeztünk az utunk végén, de több élre ekkor sincsen szükségünk).

Könnyebbség lehet továbbá, hogy általában nem feltétlenül szükséges a pontos megoldást megtalálni, sok esetben elég lehet egy megfelelő pontosságú közelítést adni. A pontos megoldás kiszámítása az integrálok kiszámításának nehézsége miatt egyébként sem tűnik elérhető célnak. Fan és Nie (2006a) a következő, egy megoldás hibájának becslését segítő tételeket fogalmazzák meg:

1.2.5. Tétel. *Az 1.2.1. algoritmust $u_{i\neq S} \equiv 0$ függvényekkel inicializálva az u_i függvények pontbeli értékei monoton növekednek az iterációk során.*

Bizonyítás. A bizonyítás teljes indukcióval történik: $k = 1$ esetében az integrálok

$$\int_0^t p_{ij}(\omega) u_j^{(0)}(t-\omega)\,\mathrm{d}\omega$$

alakúak, ahol p_{ij} egy sűrűségfüggvény, $u_{ij}^{(0)}$ pedig azonosan 0 ha $j \neq S$ és azonosan 1 ha $j = S$, így a kapott érték nemnegatív (tehát nem lehet kisebb mint a kezdeti azonosan 0 érték).

Minden további iteráció során egy $u_i^{(k+1)}(t)$ értéke $u_i^{(k)}(t)$-hez képest annyiban változik, hogy az őt meghatározó

$$\int_0^t p_{ij}(\omega) u_j^{(k)}(t-\omega)\,\mathrm{d}\omega$$

integrálokban már eggyel későbbi iterációból származó eredményeket használunk. Mivel ezek értékei az indukciós feltevés szerint semelyik pontban nem csökkenhetnek, p_{ij} továbbra is nemnegatív és változatlan, ezért az integrálok értéke sem csökkenthet, tehát $u_i^{(k+1)}(t) \geq u_i^{(k)}(t)$. □

Hasonlóan látható be a következő tétel is:

1.2.6. Tétel. *Az 1.2.1. algoritmust $u_{i\neq S} \equiv 1$ függvényekkel inicializálva az u_i függvények pontbeli értékei monoton csökkennek az iterációk során.*

A kívánt pontosságú megoldás kiszámításához ezután következőt javasolják: futtassuk az algoritmust a gráfon kétszer, konstans 1 és konstans 0 kezdőértékekkel. A belátott tételek szerint a megoldások értékei ugyanahhoz a megoldáshoz konvergálnak felülről illetve alulról. Az algoritmusokat addig futtassuk, amíg a közelítő megoldások egymás kívánt ϵ közelségébe nem érnek.

1.3. Diszkretizáció és a futásidő elemzése

Az algoritmus egy meglehetősen időigényes lépése az integrálok kiszámítása. A pontos időigény az alkalmazott módszertől függ. A későbbi megoldásokkal való összehasonlíthatóság érdekében itt ezeket diszkretizációval számoljuk ki.

Legyen $L = \frac{T}{\delta}$, ha a T időkeretet δ hosszú intervallumokra osztjuk (feltételezzük, hogy $\delta \mid T$). A jelölések egyszerűségének érdekében legyen

$$\mathcal{L} \stackrel{\text{def}}{=\!=} \{0\delta, 1\delta, \ldots, L\delta\}$$

valamint

$$\mathcal{L}^+ \stackrel{\text{def}}{=\!=} \{1\delta, 2\delta, \ldots, L\delta\},$$

a diszkrét modellben az összes felmerülő, valamint az összes felmerülő nem nulla időpillanatok halmaza.

A $p_{ij} : \mathbb{R} \to [0,1]$, $u : V \times \mathbb{R} \to [0,1]$ valamint $q : V \times \mathbb{R} \to V \cup \{\emptyset\}$ függvények értelmezési tartományát szűkítsük le $\mathbb{R}$ helyett $\mathcal{L}$-re.[4] Az egyes élekhez tartozó, utazási időt jelölő, eddig folytonos eloszlások helyett így mostantól diszkrét eloszlásokkal kell dolgoznunk.

[4] Ezeket a diszkrét halmazról képező függvényeket értelmezhetjük $(L+1)$ dimenziós vektorokként is, ahol például $p_{ij}[k] = p_{ij}(k\delta)$. Ezt a – nullától indexelt – jelölésmódot a későbbiekben többször is alkalmazni fogjuk a függvény jelölés helyett.

Az optimalitási feltétel diszkrét formalizációja ekkor a következő [Fan és Nie (2006a)]:

$$u_i(l) = \begin{cases} \max\limits_{(i,j)\in A} \sum\limits_{\substack{h\in\mathcal{L}\\ h\le l}} p_i(h)u_{ij}(l-h), & \text{ha } l\in\mathcal{L} \text{ és } i\in V, i\neq S \\ 1, & \text{ha } l\in\mathcal{L} \text{ és } i=S \end{cases} \tag{1.3.1}$$

$$q_i(l) = \begin{cases} \arg\max\limits_{(i,j)\in A} \sum\limits_{\substack{h\in\mathcal{L}\\ h\le l}} p_i(h)u_{ij}(l-h), & \text{ha } l\in\mathcal{L} \text{ és } i\in V, i\neq S \\ \emptyset, & \text{ha } l\in\mathcal{L} \text{ és } i=S \end{cases} \tag{1.3.2}$$

Ennek megfelelően számoljuk magát az algoritmust is: minden lépésben az 1.3.1. egyenletnek megfelelő diszkért konvolúciót számítjuk ki minden élre.

1.3.1. Algoritmus. (Fokozatos közelítés diszkretizációval) Kezdetben legyen (az iteráció számát felső indexben jelölve)

$$u_i^{(0)}(l) := \begin{cases} 0, & \text{ha } l\in\mathcal{L} \text{ és } i\in V, i\neq S \\ 1, & \text{ha } l\in\mathcal{L} \text{ és } i=S \end{cases},$$

majd rendre $k = 1,2,\ldots,K$-ra:

$$u_i^{(k+1)}(l) := \begin{cases} \max\limits_{(i,j)\in A} \sum\limits_{\substack{h\in\mathcal{L}\\ h\le l}} p_i(h)u_{ij}^{(k)}(l-h), & \text{ha } l\in\mathcal{L} \text{ és } i\in V, i\neq S \\ 1, & \text{ha } l\in\mathcal{L} \text{ és } i=S \end{cases}.$$

A megoldást 1.3.2. szerint számítjuk ki, az utolsó iteráció eredményéből számolva.

Az algoritmus inicializálása lineáris, nL időben történik. Egy élen egy iteráció során L darab, egyre növekvő hosszúságú összegzést szükséges kiszámítani, ezeknek a költsége $1+2+\ldots+L = \mathcal{O}\left(L^2\right)$. Minden iteráció során minden egyes élen kiszámítjuk ezeket a konvolúciókat (összesen $\mathcal{O}\left(KmL^2\right)$ időben), majd pontonkénti maximumukat vesszük (összesen $\mathcal{O}\left(KmL\right)$ időben). Ezek közül konvolúciók kiszámításának nagyságrendje a nagyobb, tehát az algoritmus futásidejének nagyságrendje összesen $\mathcal{O}\left(KmL^2\right)$.

2. fejezet

Pontos megoldás a diszkrét és a folytonos modellben

Az itt következő algoritmusok elsősorban ott térnek el a korábban tárgyalt fokozatos közelítéses algoritmustól, hogy nem *térben* hanem *időben* haladva dolgozzák fel a gráfot. Ennek az előnye az, hogy minden kiszámított érték *pontos*, hiszen a számításhoz szükséges adatok annak megtörténtekor már rendelkezésre állnak, ezért a feladat tér-idő szerkezetén mindösszesen egyszer kell keresztülhaladnia az algoritmusnak.

2.1. Az idő szerinti iteráció

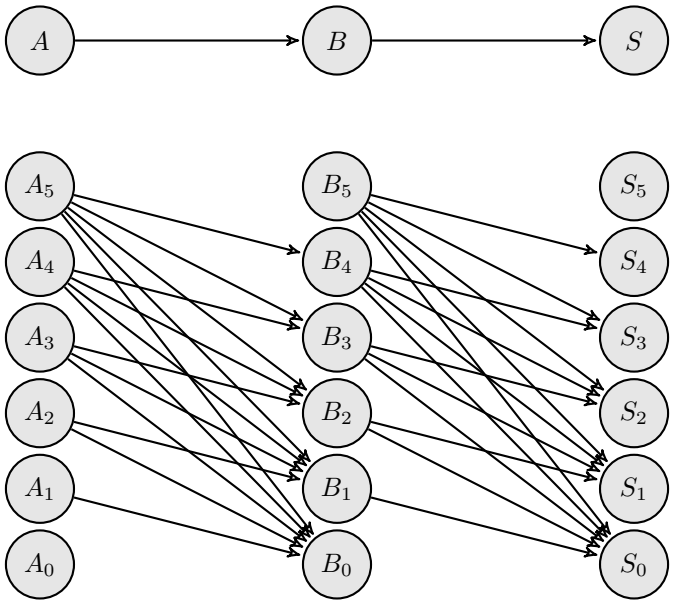
A tér-idő szerkezet könnyebb átlátásához Fan és Nie (2006b) a következőt javasolják: először készítsünk el képzeletben egy segédgráfot. Az új gráfnak egy pontja származzon $V \times T$ halmazból, azaz valójában az eredeti gráf egy pontja egy bizonyos időpontban. Ezek között az új gráfban attól függően legyenek élek pontok között, hogy melyikből melyikbe lehet eljutni az eredeti gráfban: ha $(i,j) \in A$ az eredeti gráfban, akkor legyen (i_{t_1}, j_{t_2}) él az új gráfban, amennyiben $t_1 > t_2$.

Ezzel implicit módon feltettük – és ez lesz a idő szerint iteráló algoritmusok alapötlete –, hogy az *utazónk* nem tartózkodhat egymás után két pontban ugyanabban az időpillanatban, tehát minden pont-idő párból az időben érkezés valószínűsége csak nála kisebb időkerettel rendelkező pontoktól függhet.

Máshogy megfogalmazva, ahogy az utazó halad a gráfban, a 2.1. ábrán látható elrendezés szerint ahogy telik az idő, egyre „lejjebb" jut, míg végül el nem ér S-be vagy el nem fogy az időkeret.

Az éleket az utazó természetesen nem tetszés szerint választja: azoknak egy $\{j_{t_2} \mid t_2 \leq t_1\}$ halmazát tudja megjelölni uticélként és az, hogy mennyire „lentre" érkezik a gráfban, a véletlenen múlik.

A fenti segédgráfot természetesen nem praktikus egy tényleges implementáció során megkonstruálni, az csak könnyebb átláthatóságot szolgálja. Ezek alapján fogalmazzák meg Fan és Nie (2006b) a 2.1.1. algoritmust.



2.1. ábra. Fent: eredeti gráf. Lent: öt diszkrét időpillanatra kiterjesztett segédgráf.

2.1.1. Algoritmus. (Pontos megoldás a diszkrét modellben) Legyen kezdetben

$$u_i(0) := \begin{cases} 0, & \text{ha } i \in V, i \neq S \\ 1, & \text{ha } i = S \end{cases},$$

majd növekvő sorrendben minden $l \in \mathcal{L}^+$-ra

$$u_i(l) := \begin{cases} \max\limits_{(i,j)\in A} \sum\limits_{\substack{h\in\mathcal{L}^+ \\ h\leq l}} p_i(h)u_{ij}(l-h), & \text{ha } i \in V, i \neq S \\ 1, & \text{ha } i = S \end{cases}.$$

2.1.2. Tétel. *A 2.1.1. algoritmus pontosan számítja ki az optimális megoldást és futásideje $\mathcal{O}\left(mL^2\right)$. (Ez a fokozatos közelítéses algoritmus egy iterációjának a futásideje.)*

Bizonyítás. A kiszámított megoldás pontos volta gyakorlatilag a képletekből látszik: az első lépést kivéve minden lépésben csak korábban (pontosan) kiszámított értékeket használunk fel. Vegyük észre: $h \in \mathcal{L}^+$ miatt $h > 0$, tehát $u_{ij}(t)$ értékére csak $t < l$ helyeken van szükségünk, így az újonnan kiszámított értékek is pontosak.

Az, hogy tetszőleges i pontból 0 időkerettel az időben érkezés valószínűsége 0, valamint hogy S pontból tetszőleges (nemnegatív) időkerettel az időben érkezés valószínűsége 1, nyilvánvaló.

Az algoritmus minden élen L darab összegzést számít ki $1 + 2 + \ldots + L = \mathcal{O}\left(L^2\right)$ idő alatt. A maximumkeresések összes futásideje $\mathcal{O}\left(mL\right)$, aminél az összegzések kiszámításának nagyságrendje (itt is) nagyobb, tehát az algoritmus futásidejének nagyságrendje összesen $\mathcal{O}\left(mL^2\right)$. □

2.2. Pontos megoldás a folytonos modellben

A 2.1 fejezetben tárgyalt idő szerinti feldolgozás ötlete átvihető a folytonos modellbe is. Ehhez viszont hasonló feltételezést kell tennünk, mint a diszkrét modellben. A megfelelő feltétel már korábban felmerült az 1.2.4. tétel kimondása során: legyen adott minden (i,j) élen egy *minimális lehetséges áthaladási költség*, d_{ij}, amelyre $t \leq d_{ij}$ esetén $p_{ij}(t) = 0$, továbbá legyen $d = \min_{(i,j)\in A} d_{ij}$. Nevezzük D-nek a $\frac{T}{d}$ értéket (feltételezzük, hogy $d \mid T$).

A folytonos modellben a számítás ekkor a következőképpen történhet [Samaranayake, Blandin és Bayen (2011)]:

2.2.1. Algoritmus. Végezzük el rendre minden $k = 1, \ldots, D$-re a következő számításokat:

$$u_i(t) := \begin{cases} \max\limits_{(i,j)\in A} \int\limits_0^t p_{ij}(\omega) u_j(t-\omega)\,\mathrm{d}\omega, & \text{ha } i \in V, i \neq S \\ 1, & \text{ha } i = S \end{cases} \qquad \forall t \in [(k-1)d, kd)\text{-re}$$

2.2.2. Tétel. *A 2.2.1. algoritmus pontosan megoldja a feladatot a folytonos modellben.*

Bizonyítás. Elég belátnunk, hogy az iterációs lépésben végzett számítások itt is csak a korábbi iterációk eredményeit használják fel. Ez valóban így van, hiszen a feltevésünk szerint minden élen 0 és d között minden $p_{ij}(t)$ azonosan nulla, ezért

$$u_i(t) = \max_{(i,j)\in A} \int_0^t p_{ij}(\omega) u_j(t-\omega)\,\mathrm{d}\omega = \max_{(i,j)\in A} \int_d^t p_{ij}(\omega) u_j(t-\omega)\,\mathrm{d}\omega$$

tehát a maximális hely, ahol u_j értékére még szükség van $t - d$. Ezzel az állítást beláttuk. □

A folytonos modell pontos megoldása persze nem sokat ér, ha a számításokat továbbra is diszkretizálással végezzük – ekkor ugyanis az algoritmus ugyanazokat a számításokat végzi, mint a 2.1.1. algoritmus, csak (d értékétől függően) esetleg más sorrendben. Az elméleti pontos megoldási módszer viszont lehetőséget nyújt arra, hogy az integrálokat tetszőleges numerikus integrálási módszerrel ki tudjuk számolni.

3. fejezet

Az FFT és ennek alkalmazása a konvolúciók futásidejének javítására

Amennyiben diszkretizációval számítjuk ki az algoritmusok során felmerülő integrálokat, úgy ezek diszkrét konvolúciókká alakulnak. Egy ilyen konvolúció futásideje $1 + 2 + \ldots + L = \mathcal{O}\left(L^2\right)$, amivel az algoritmus legköltségesebb számításává válik. Ismert tény viszont, hogy egy $v_1 * v_2 \in \mathbb{R}^N$ *diszkrét konvolúció* $\mathcal{O}\left(N \log N\right)$ időben is kiszámítható *gyors Fourier transzformáció*, vagy röviden *FFT*[1] segítségével.

3.1. DFT definíciója és az FFT algoritmus

A *diszkrét Fourier-transzformációt* vagy más néven *DFT*-t többféleképpen is lehetséges motiválni valamint definiálni. A mi céljainkra a legegyszerűbb a $\mathbb{C}^N$ vektortéren vett lineáris transzformációként kezelni. A fejezetben leírt állítások és bizonyítások Osgood *The Fourier Transform and its Applications* (2007) könyvén alapulnak.

A formulák egyszerűsítése érdekében egy N dimenziós vektor indexei menjenek 0-tól $(N-1)$-ig, valamint ennek megfelelően egy A mátrix n. sorának m. elemére is $A_{(n-1),(m-1)}$-gyel hivatkozunk (vagy másképpen, a mátrix „első" sorát, illetve oszlopát innentől kezdve nulladiknak nevezzük).

Az indexeket továbbá modulo N szükséges érteni, tehát például $v[N+n] = v[n]$ és $v[-n] = v[N-n]$. Ezzel a vektorokat gyakorlatilag „periodikusnak" tekintjük. Arról majd csak később lesz szó, hogy ez hogyan viszonyul azokhoz a számításokhoz, amelyekre nekünk szükségünk van.

3.1.1. Definíció. (Diszkrét Fourier-transzformáció) Legyen $f \in \mathbb{C}^N$ vektor. Ekkor f *diszkrét Fourier-transzformáltja* vagy egyszerűen *Fourier-transzformáltja*, $\mathcal{F}f \in \mathbb{C}^N$, amit a következőképpen definiálunk:

$$\mathcal{F}f[m] \overset{\text{def}}{=\!=} \sum_{n=0}^{N-1} f[n]\omega^{-mn} \tag{3.1.1}$$

ahol

$$\omega^n \overset{\text{def}}{=\!=} e^{\left(\frac{2\pi i n}{N}\right)}, \tag{3.1.2}$$

[1] Az angol *Fast Fourier-transform* rövidítése.

azaz ω^n egy N. primitív egységgyök n. hatványa.

A 3.1.1. képletet mátrixszorzásként is lehet értelmezni. Ekkor a transzformáció mátrixa, $\underline{\mathcal{F}}$, a következő (szimmetrikus) mátrix:

$$\underline{\mathcal{F}} \stackrel{\text{def}}{=\!=} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-(1\cdot 1)} & \omega^{-(1\cdot 2)} & \cdots & \omega^{-(1\cdot(N-1))} \\ 1 & \omega^{-(2\cdot 1)} & \omega^{-(2\cdot 2)} & \cdots & \omega^{-(2\cdot(N-1))} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-((N-1)\cdot 1)} & \omega^{-((N-1)\cdot 2)} & \cdots & \omega^{-((N-1)^2)} \end{pmatrix}$$

azaz

$$\underline{\mathcal{F}}_{n,m} = \omega^{-nm}.$$

Legyen továbbá

$$\underline{\boldsymbol{\omega}} \stackrel{\text{def}}{=\!=} (1,\omega,\omega^2,\ldots,\omega^{(N-1)}) \tag{3.1.3}$$

valamint

$$\underline{\boldsymbol{\omega}}^n \stackrel{\text{def}}{=\!=} (1,\omega^n,\omega^{2n},\ldots,\omega^{(N-1)n}). \tag{3.1.4}$$

Ezekkel a jelölésekkel

$$\underline{\mathcal{F}}_{n,\cdot} = \underline{\boldsymbol{\omega}}^{-n} \text{ és } \underline{\mathcal{F}}_{\cdot,m} = \left(\underline{\boldsymbol{\omega}}^{-m}\right)^T$$

teljesülnek a mátrix definíciójából adódóan (itt $A_{n,\cdot}$ és $A_{\cdot,m}$ a megfelelő sort és oszlopot jelölik).

Az $\mathcal{F}$ transzformáció meglehetősen könnyen invertálható, ugyanis a transzformációs mátrix $\underline{\mathcal{F}}$ „majdnem unitér". Ezt látja be a következő tétel.

3.1.2. Tétel. (A diszkrét Fourier-transzformáció inverze)

$$\underline{\mathcal{F}}^{-1} = \frac{1}{N}\underline{\mathcal{F}}^*$$

(Ahol $\underline{\mathcal{F}}^*$ a mátrix konjugált transzponáltját jelöli)

Bizonyítás. Mivel az $\underline{\mathcal{F}}$ szimmetrikus, a konjugált transzponált alatt valójában csak konjugáltat értünk. Mivel a mátrix egy elemének, $\underline{\mathcal{F}}_{n,m} = \omega^{-nm}$-nek a konjugáltja ω^{nm}, ezért

$$\left(\underline{\mathcal{F}} \cdot \underline{\mathcal{F}}^*\right)_{n,m} = \left\langle \underline{\boldsymbol{\omega}}^{-n}, \underline{\boldsymbol{\omega}}^{m} \right\rangle.$$

A $\underline{\boldsymbol{\omega}}$ vektor definíciója miatt

$$\left\langle \underline{\boldsymbol{\omega}}^{-n}, \underline{\boldsymbol{\omega}}^{m} \right\rangle = \sum_{k=0}^{N-1} \omega^{(m-n)k},$$

a mértani sorozat összegképlete alapján

$$\sum_{k=0}^{N-1} \omega^{(m-n)k} = \begin{cases} \frac{\left(\omega^{m-n}\right)^N - 1}{\omega^{m-n}-1}, & \text{ha } \omega^{m-n} \neq 1 \\ N, & \text{ha } \omega^{m-n} = 1 \end{cases}.$$

Mivel a definícióból következően ha ω egy N. egységgyök, akkor $\omega^n = 1$ minden $n \equiv 0 \mod N$-re, ezért

$$\frac{\omega^{(m-n)N} - 1}{\omega^{m-n} - 1} = \frac{1-1}{\omega^{m-n} - 1} = 0,$$

továbbá mivel tudjuk, hogy $0 \leq n < N$ és $0 \leq m < N$ ezért

$$\omega^{(m-n)} = 1 \Leftrightarrow m = n.$$

Ez egyenlőség-lánc elejét és végét egymás mellé rakva így

$$(\underline{\mathcal{F}} \cdot \underline{\mathcal{F}}^*)_{n,m} = \begin{cases} 0, & \text{ha } n \neq m \\ N, & \text{ha } n = m \end{cases}.$$

Ezzel az állítást beláttuk. □

Emlékezzünk vissza, a célunk két vektor konvolúciójának kiszámítása. Mivel a feladatban minden érték valós, ezért a következő tételt elég valós vektorokra belátnunk. Mivel ezt eddig nem tettük meg, definiáljuk először a diszkrét konvolúciót.

3.1.3. Definíció. (Diszkrét konvolúció) Két n dimenziós vektor, $v_1, v_2 \in \mathbb{R}^N$ *diszkrét konvolúciója* alatt azt a $(v_1 * v_2) \in \mathbb{R}^N$ vektort értjük, amire minden $n = 0, 1, \ldots, (N-1)$-re

$$(v_1 * v_2)[n] \overset{\text{def}}{=\!=} \sum_{i=0}^{N-1} v_1[i] v_2[n-i]$$

3.1.4. Tétel. (Diszkrét konvolúció kiszámítása DFT-vel) *Két $\mathbb{R}^N$-beli vektor, v_1 és v_2 Fourier-transzformáltjának a pontonkénti szorzata pontosan a két vektor konvolúciójának a Fourier-transzformáltja, azaz*

$$(\mathcal{F}v_1) \cdot (\mathcal{F}v_2) = \mathcal{F}(v_1 * v_2)$$

ahol a $\cdot$ művelet a pontonkénti szorzást jelöli, azaz

$$v_1 \cdot v_2 = (v_1[0] \cdot v_2[0], v_1[1] \cdot v_2[1], \ldots, v_1[N-1] \cdot v_2[N-1]).$$

Bizonyítás. A Fourier-transzformáció, a konvolúció és a pontonkénti szorzás műveletek minden változójukban lineárisak, így az állítást elég egy bázis elemeire belátnunk. Jelölje $e_0, e_1, \ldots, e_{(N-1)}$ a standard bázisát $\mathbb{R}^n$-nek.

Egyrészt

$$(e_m * e_k)[n] = \sum_{i=0}^{N-1} e_m[i] e_k[n-i] = e_m[m] e_k[n-m] =$$

$$= e_k[n-m] = \begin{cases} 1, & \text{ha } k + m \equiv n \mod N \\ 0, & \text{ha } k + m \not\equiv n \mod N \end{cases}$$

így tehát

$$e_m * e_k = \begin{cases} e_{(m+k)}, & \text{ha } n + m < N \\ e_{(m+k)-N}, & \text{ha } n + m \geq N \end{cases},$$

és mivel egy standard bázisvektorra $\underline{\mathcal{F}}\, e_n = \underline{\boldsymbol{\omega}}^{-n}$ ezért

$$\underline{\mathcal{F}}\,(e_m * e_k) = \begin{cases} \underline{\boldsymbol{\omega}}^{-(m+k)}, & \text{ha } n+m < N \\ \underline{\boldsymbol{\omega}}^{-(m+k)+N}, & \text{ha } n+m \geq N \end{cases}.$$

Másrészt ezek alapján

$$(\underline{\mathcal{F}}\, e_m) \cdot (\underline{\mathcal{F}}\, e_k) = \underline{\boldsymbol{\omega}}^{-m} \cdot \underline{\boldsymbol{\omega}}^{-k} = \underline{\boldsymbol{\omega}}^{-(m+k)} = \underline{\boldsymbol{\omega}}^{-(m+k)+N},$$

amivel az állítást beláttuk. □

Ezzel eljutunk a lényeghez:

3.1.5. Tétel. (Gyors Fourier-transzformáció) *Egy N dimenziós vektor diszkrét Fourier-transzformáltja, valamint ennek az inverze is kiszámítható $\mathcal{O}(N \log N)$ időben. (Algoritmust / algoritmusokat itt nem tárgyaluk.)*

Akad viszont egy kis kellemetlenség, a fenti definíció szerinti diszkrét konvolúció,

$$(v_1 * v_2)[n] = \sum_{i=0}^{N-1} v_1[i] v_2[n-i] \tag{3.1.5}$$

némileg más alakú mint amire nekünk szükségünk van,

$$u_i[n] = \sum_{k=0}^{n} p_{ij}[k] u_j[n-k]. \tag{3.1.6}$$

Ezt a következő okozza: az 1.1.2. képlettől kezdve az integrálokat nem az $\mathbb{R}$-en végeztük, hanem csak a $[0,t]$ intervallumon. Ezt megtehettük, hiszen az egyenlőség jobb oldalán álló $u_j(t-\omega)p_{ij}(\omega)$ szorzatban $\omega < 0$ esetén $p_{ij}(\omega) = 0$, $\omega > t$ esetén pedig $u_j(t-\omega) = 0$ (figyelem, ez az ω most nem ugyanaz, mint a fenti!). A kapott érték nem változik, ha az integrálokat a $[-T,T]$ intervallumon végezzük.

Ezt a diszkrét esetbe úgy tudjuk átvinni, ha a p_{ij} és u_j vektorok hosszát a kétszeresére növeljük, azaz kibővítjük azokat N darab negatív indexű taggal (amelyek mindegyikének értéke nulla).

Ekkor a 3.1.6. egyenlőség helyett a következő írható:

$$u_i[n] = \sum_{k=-N}^{N-1} p_{ij}[k] u_j[n-k].$$

ami az indexek periodikussága miatt átírható

$$u_i[n] = \sum_{k=0}^{2N-1} p_{ij}[k] u_j[n-k]$$

formába is. Ez már majdnem megegyezik 3.1.5 alakjával, csak más a kiszámolt összeg tagjainak száma – nem ugyanaz tehát a két esetben a vektorok dimenziója.

Ha a feladatunkban a diszkretizációs pontok száma N, azaz N dimenziósak a vektoraink, akkor a diszkrét Fourier-transzformáció segítségével végzett konvolúciót $2N$ hosszúságú vektorokra szükséges

kiszámítanunk, ahol a vektorok második fele azonosan 0. A kapott vektoroknak a „pozitív" koordinátáit véve – azaz a $0 \leq i \leq (N-1)$ indexeket – pontosan a kívánt eredményt kapjuk. Ez az algoritmus futásidejének nagyságrendjén nem változtat.

Amikor tehát egy

$$u_i[n] = \sum_{k=0}^{n} p_{ij}[k] u_j[n-k]$$

értéket szeretnénk számolni minden $0 \leq n \leq (N-1)$-re, először kiegészítjük p_{ij}-ét és u_j-t annyi nullával, hogy azok $2N$ hosszúak legyenek, majd (a kapott vektorokat p^*_{ij}-gal és u^*_j-gal jelölve) kiszámítjuk a

$$u^*_i = \underline{\mathcal{F}}^{-1}((\underline{\mathcal{F}}\, p^*_{ij}) \cdot (\underline{\mathcal{F}}\, u^*_j))$$

értéket és ennek az első N indexét véve kapjuk a megoldást. Ezt a továbbiakban *gyors konvolúciónak* fogjuk nevezni. A Fourier-transzformációt és ennek az inverzét gyors Fourier-transzformációval végezve a számítás költsége így $3 \cdot \mathcal{O}(\mathrm{FFT}) + 2N$ ami a 3.1.5. tétel alapján $\mathcal{O}(3 \cdot \mathcal{O}((2N)\log(2N))) = \mathcal{O}(N \log N)$.

3.2. Gyors konvolúció az idő szerinti iterációs algoritmusban

A 2.1.1. algoritmus a 2.2. fejezet elején tett, a minimális utazási költségekről szóló feltételt és a hozzá kapcsolódó jelöléseket megtartva felírható a következőképpen is:

3.2.1. Algoritmus. [Samaranayake, Blandin és Bayen (2011)]
Végezzük el rendre minden $k = 1, \ldots, D$-re a következő számításokat:

$$u_i(l) := \begin{cases} \max\limits_{(i,j)\in A} \sum\limits_{\substack{h \in \mathcal{L}^+ \\ h \leq l}} p_i(h) u_{ij}(l-h), & \text{ha } i \in V, i \neq S \\ 1, & \text{ha } i = S \end{cases} \qquad \forall l \in (\mathcal{L} \cap [(k-1)d, kd)\,)\text{-re}$$

Az algoritmus futásideje ideális esetben a 2.1.1 algoritmus futásidejénél kevesebb lenne annyival, amennyivel a konvolúciókat fel tudjuk gyorsítani, tehát $\mathcal{O}(mL^2) = \mathcal{O}\left(m\frac{T^2}{\delta^2}\right)$ helyett $\mathcal{O}(mL\log(L)) = \mathcal{O}\left(m\frac{T}{\delta}\log\left(\frac{T}{\delta}\right)\right)$. Viszont mivel a konvolúciókat a 2.1.1 algoritmus során elemenként, itt pedig nagyjából $\frac{d}{\delta}$ méretű blokkokban számoljuk ki, *nem egyben*, ezért a gyors Fourier-transzformációval minden egyes új blokkal együtt az összes korábbit is ki számolnunk. Ez így a futásidő nagyságrendjét nemhogy csökkenti, de emeli: minden élen D darab, egyre növekvő hosszúságú konvolúciót számolunk FFT-vel egyenként $c_f\, t\log(t)$ időben, ez összesen

$$\begin{aligned} & m\left[c_f\,\frac{d}{\delta}\log\left(\frac{d}{\delta}\right) + c_f\,\frac{2d}{\delta}\log\left(\frac{2d}{\delta}\right) + \ldots + c_f\,\frac{Dd}{\delta}\log\left(\frac{Dd}{\delta}\right)\right] \leq \\ \leq\ & m\frac{d}{\delta}\log\left(\frac{Dd}{\delta}\right) c_f\,\left[1 + 2 + \ldots + D\right] \approx \\ \approx\ & m\frac{d}{\delta}\log\left(\frac{Dd}{\delta}\right) c_f\, D^2 = \\ =\ & m\frac{d}{\delta}\log\left(\frac{Td}{d\delta}\right) c_f\,\frac{T^2}{d^2} = \\ =\ & m\frac{T^2}{d\delta}\log\left(\frac{T}{\delta}\right) c_f\, = \mathcal{O}\left(m\frac{T^2}{d\delta}\log(L)\right) \end{aligned}$$

Ez nagyságrendjében ugyan tényleg egy $\log(L)$ szorzóval nagyobb, mint az eredeti algoritmus $\mathcal{O}\left(mL^2\right) = \mathcal{O}\left(m\frac{T^2}{d^2}\right)$ futásideje, viszont vegyük észre, hogy T^2-et δ^2 helyett $d\delta$-val osztjuk. Ez δ-tól és d-től függően elég nagy lehet, akár akkora, hogy implementációtól függően egyes konkrét példákra érdemesebb használni az itt leírt algoritmust [Samaranayake, Blandin és Bayen (2011)]. A két módszer futásidejének összehasonlítására az 5. fejezetben fogunk vizsgálni konkrét implementációkat és adatokat.

Minden egyes új blokkra az összes korábbit végigszámolni mégis elég pazarló, de ennek kiküszöbölése nem triviális. Erre nyújt megoldást (valamelyest) a 3.3. fejezetben tárgyalt Zero Delay Convolution algoritmus.

3.3. Zero Delay Convolution és ennek alkalmazása

A 3.2. fejezetben bemutattunk egy algoritmust, ami idő szerint iterál és pontosan kiszámolja a megoldást, valamint a konvolúciókat gyors konvolúcióval számolja. Viszont mivel az adatok amikből a konvolúciót számolja csak d méretű blokkokban válnak egymás után elérhetővé, ezért a gyors konvolúció használatakor minden új blokkal ki kellett számolni az összes addigit is. Ez a diszkrét Fourier-transzformációnak abból a tulajdonságából adódott, hogy annak a segítségével egy konvolúciót az eddigi módszerrel csak egyben tudtunk kiszámítani, növekvő darabokban nem.

Ez a probléma viszont felmerül más területeken is: a jelfeldolgozásban teljesen életszerű szituációnak mondható, hogy egy bejövő jelet először valamilyen módon fel kell dolgozni, majd minimális késedelemmel továbbítani, limitált számítási kapacitás mellett. A feldolgozás tipikusan Fourier-transzformációval végzett konvolúció segítségével is történhet.

A probléma megoldására Gardner (1995) egy olyan megoldás ad, ami növekvő méretű blokkokra osztva számolja ki a konvolúciót a gyors Fourier-transzformált segítségével, így semmit nem számol kétszer.

Az algoritmus megértéséhez képzeljünk el egy konvolúciót egy táblázatként. Rendeljük hozzá u_j megfelelő indexű tagjait az oszlopokhoz, p_{ij} megfelelő indexű tagjait a sorokhoz, a cellákban pedig jelenjenek meg a megfelelő elemek szorzatai. Ekkor a konvolúció elemeit megkaphatjuk a táblázat elemeit diagonálisan összegezve. Lásd: 3.1. táblázat. Itt például

$$\begin{aligned}
(p_{ij} * u_j)[0] &= p_{ij}[0]u_j[0], \\
(p_{ij} * u_j)[1] &= p_{ij}[1]u_j[0] + p_{ij}[1]u_j[0], \\
(p_{ij} * u_j)[2] &= p_{ij}[2]u_j[0] + p_{ij}[1]u_j[1] + p_{ij}[0]u_j[2], \\
&\vdots \\
(p_{ij} * u_j)[N-1] &= p_{ij}[N-1]u_j[0] + \ldots + p_{ij}[0]u_j[N-1].
\end{aligned}$$

A konvolúció értékeire csak a mellékátlóig van szükségünk, de azokat a gyors Fourier-transzformációval végzett konvolúció számítása során valójában a $(p_{ij} * u_j)[2N-2] = p_{ij}[N-1]u_j[N-1]$ elemig kiszámoljuk.

A Samaranayake, Blandin és Bayen (2012) által javasolt *Zero Delay Convolution* algoritmus a következő elven működik: amennyiben egy ilyen konvolúciós táblázatból kiszámoljuk egy tetszőleges méretű, négyzet alakú blokk konvolúcióját, ennek eredményét fel tudjuk használni egy azt magában foglaló négyzet alakú blokk konvolúciójának kiszámításához úgy, hogy abból semmit nem kell újraszámolnunk.

	u_j				
p_{ij}	$p_{ij}[0]u_j[0]$	$p_{ij}[0]u_j[1]$	$p_{ij}[0]u_j[2]$	$\cdots$	$p_{ij}[0]u_j[N-1]$
	$p_{ij}[1]u_j[0]$	$p_{ij}[1]u_j[1]$	$p_{ij}[1]u_j[2]$	$\cdots$	$p_{ij}[1]u_j[N-1]$
	$p_{ij}[2]u_j[0]$	$p_{ij}[2]u_j[1]$	$p_{ij}[2]u_j[2]$	$\cdots$	$p_{ij}[2]u_j[N-1]$
	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	$p_{ij}[N-1]u_j[0]$	$p_{ij}[N-1]u_j[1]$	$p_{ij}[N-1]u_j[2]$	$\cdots$	$p_{ij}[N-1]u_j[N-1]$

3.1. táblázat. Konvolúció vizuálisan.

Amikor ugyanis a nagyobb blokkhoz tartozó konvolúció megfelelő elemét számoljuk, az egyes szorzatok kiszámolása helyett egyszerűen hozzá tudjuk adni a kisebb konvolúció megfelelő elemét.

3.3.1. Példa. A 3.1. táblázatból ha ismerjük a $K = (p_{ij}[1], p_{ij}[2]) * (u_j[0], u_j[1])$ eredményét, akkor $(p_{ij} * u_j)[2]$ kiszámolásához fel tudjuk használni ennek az eredményét:

$$(p_{ij} * u_j)[2] \;=\; (p_{ij}[2]u_j[0] + p_{ij}[1]u_j[1]) + p_{ij}[0]u_j[2] \;=\; K[1] + p_{ij}[0]u_j[2]$$

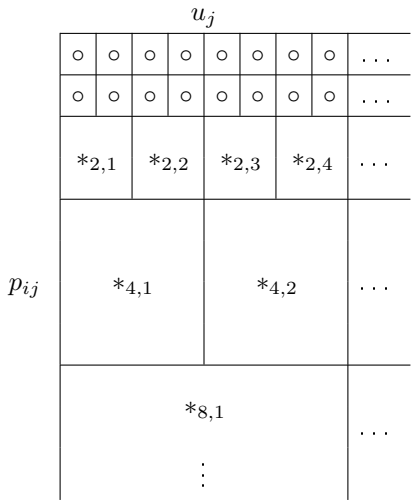
Ezt tudván egy táblázatot feloszthatunk kisebb, exponenciálisan növekvő méretű blokkokra (3.1. ábra). A fokozatosan (itt most egyenként) beérkező adatok vektora tartozzon a oszlopokhoz, az ismert vektor pedig a sorokhoz.

A 0. és az 1. sorban található szorzatokat számoljuk úgy, ahogy eddig: amikor először szükség van rá. A többi sort viszont blokkokra oszjuk: a 2. és 3. sorból képezzünk 2×2 méretű blokkokat, a 4., 5., 6. és 7. sorból képezzünk 4×4 méretű blokkokat és így tovább: általánosan, (az első két sor kivételével) a (2^n). sortól a $(2^{n+1} - 1)$. sorig tartó sorokból képezzünk $2^n \times 2^n$ méretű blokkokat. Az $n \times n$ méretű blokkok konvolúcióit jelöljük $*_{n,1}$, $*_{n,2}$, stb.-vel.

Ezeket a blokkokat, amint van elég információnk hozzá az algoritmus futása során kiszámoljuk. Célunk, hogy a konvolúció k. eleme kiszámítható legyen amint a darabokban érkező vektor (itt u_j) k. eleme rendelkezésünkre áll. Vegyük észre, hogy ezzel a konstrukcióval minden blokkot egy lépéssel korábban ki tudunk számolni, mint hogy annak az eredményére szükség lenne, a konvolúciók átlós elhelyezkedése miatt.

A számolás futásideje az egyes blokkokhoz tartozó konvolúciók kiszámolásának idejéből és a megfelelő eredmények összeadásának idejéből adódik. A blokkokhoz tartozó konvolúciók kiszámolásának időigénye (kettőhatvány N-et feltételezve)

$$\begin{aligned}
&2N + \frac{N}{2}(2\log(2)) + \frac{N}{4}(4\log(4)) + \ldots + \frac{N}{2^{\log_2(N)-1}}\left(2^{\log_2(N)-1}\log\left(2^{\log_2(N)-1}\right)\right) = \\
&= 2N + N\left(\log\left(2^1\right) + \log\left(2^2\right) + \ldots + \log\left(2^{\log_2(N)-1}\right)\right) = \\
&= 2N + N\log(2)\left(1 + 2 + \ldots + (\log_2(N) - 1)\right) \approx \\
&\approx 2N + N\log(2)\log^2(N) = \\
&= \mathcal{O}\left(N\log^2(N)\right)
\end{aligned}$$



3.1. ábra. Blokkokra osztás a Zero Delay Convolution algoritmusban

A megfelelő eredmények összeadása nem haladja meg az $\mathcal{O}(N\log(N))$ nagyságrendet, tehát a futásidő összesen $\mathcal{O}\left(N\log^2(N)\right)+\mathcal{O}(N\log N)=\mathcal{O}\left(N\log^2(N)\right)$.

Érdemes megemlíteni, hogy a gyors konvolúció számításának jellege miatt a Fourier-transzformáltak egy részét nem kell többször kiszámolni, csak egyszer. Egészen pontosan p_{ij} egy 2^n hosszú intervallumának a Fourier-transzformáltját minden $*_{2^n,k}$ konvolúció kiszámításához felhasználjuk. Amennyiben a számítás eredményét eltároljuk, nem szükséges azt mindig újraszámolni.

A 3.2.1. algoritmusban ha ezzel a módszerrel számoljuk a konvolúciókat, annak a futásideje a következőképpen változik: ugyanúgy minden élen ki kell számolnunk egyre növekvő hosszúságig egy-egy konvolúciót, viszont egy ilyennek a kiszámolása összesen most nem $\mathcal{O}\left(L^2\log(L)\right)$ hanem $\mathcal{O}\left(L\log^2(L)\right)$ időben történik, így az algoritmus futásideje összesen $\mathcal{O}\left(mL\log^2(L)\right)$.

Megjegyezzük, hogy a konkrét feladat esetében u_j értékei nem egyesével hanem d méretű intervallumokra bontva válnak elérhetővé, ezért minden blokk mérete egy d szorzóval nagyobb lehet, beleértve az első két sor 1×1 méretű blokkjait is – ekkor az első két sor helyett az első $2d$ sorról tudunk beszélni, hiszen már azokban is tudjuk blokkokban végezni a konvolúciót.[2] Kettőhatvány d érték esetében például a fenti számolás a következőképpen alakul (feltételezve, hogy N előáll $2^k d$ alakban):

$$
\begin{aligned}
&2\left(\frac{N}{d}d\log(d)\right)+\frac{N}{2d}(2d\log(2d))+\frac{N}{4d}(4d\log(4d))+\ldots+\frac{N}{2^{k-1}d}\left(2^{k-1}d\log\left(2^{k-1}d\right)\right)=\\
&=2N\log(d)+N\left(\log\left(2^1d\right)+\log\left(2^2d\right)+\ldots+\log\left(2^{k-1}d\right)\right)=\\
&=2N\log(d)+N\log(2)\left(1+2+\ldots+(k-1)\right)+N(k-1)\log(d))\approx\\
&\approx 2N+N\log(2)k^2+Nk\log(d)
\end{aligned}
$$

ez pedig $k=\log_2\left(\frac{N}{d}\right)$ miatt $\mathcal{O}\left(N\log^2\left(\frac{N}{d}\right)\right)+\mathcal{O}\left(N\log\left(\frac{N}{d}\right)\log(d)\right)$ nagyságrendű.

[2] Nem vettük figyelembe se itt, se korábban, hogy a gyors Fourier-transzformációval végzett konvolúciónak csak a nagyságrendje nagyobb, mint a definíció szerint számoló megoldásnak, nem feltétlenül lesz gyorsabb minden n-re. A kérdést bővebben az 5. fejezetben járjuk körbe.

4. fejezet

Egyéb kiegészítések

Ebben a fejezetben olyan kérdéseket vetünk fel, illetve járunk körbe röviden, amelyek ugyan szorosan kacsolódnak a feladathoz, de mindeddig nem kerültek említésre.

4.1. Időben változó eloszlások

Valós feladatoknál az éleken való áthaladási idők eloszlásai függhetnek attól, hogy például hézköznapon vagy hétvégén, reggel vagy délután, stb. történik az utazás. Ezt általánosságban úgy lehet megfogalmazni, hogy a c_{ij} valószínűségi változó eloszlása az időtől is függ, egészen pontosan legyen $c_{ij}^{(t_1)}$ az (i,j) élen való áthaladás idejét jelölő valószínűségi változó, az i pontból t_1 időpillanatban indulva (és hasonlóképpen $p_{ij}^{(t_1)}$ ennek a sűrűségfüggvénye).

Tegyük fel, hogy az utazó 0 időpillanatban indul T időkerettel. Ekkor a hátralévő időkeret és az aktuális időpillanat között egy-egy értelmű hozzárendelés áll fent, így a konvolúciók számolása során megtehetjük, hogy az egyes szorzatok számolásakor a megfelelő időpillanathoz tartozó eloszlásokból vesszük az értékeket.

Tegyük fel, hogy eltelt t_1 idő az indulás óta, tehát a jelenlegi időpillanat t_1, a maradék időkeret $T-t_1$ és az utazó a j pontban áll. Ekkor a (j,r) élen való továbbhaladás esetén az időben érkezés valószínűségét a következő formula határozza meg:

$$u_j^{(t_1)}(T-t_1) = \int_0^{T-t1} p_{ir}^{(t_1)}(\omega) u_j^{(t_1+\omega)}((T-(t_1+\omega))\,\mathrm{d}\omega$$

Az $u_j^t(T-t)$ értéket tehát jelölhetjük egyszerűen $u_j^*(T-t)$-vel, hiszen csak ilyen formában használjuk. A diszkrét modellben tehát egy u_i^* vektort a következő diszkrét konvolúció határoz meg:

$$u_i^*(l) = \max_{(i,j)\in A} \sum_{\substack{h\in\mathcal{L}\\ h\leq l}} p_{ij}^{(T-l)}(h) u_j^*(l-h), \qquad \forall\, l\in\mathcal{L}\ \ \forall\, i\in V, i\neq S$$

Ez a definíció szerint számoló (vagy más néven *brute-force*) konvolúció során minden nehézség nélkül megvalósítható (mindig a megfelelő p_{ij} vektor értékeit véve). A gyors konvolúciót viszont ebben az esetben nem lehet alkalmazni, ugyanis a számítás során a p_{ij} vektor értéke változik.

Ez a formalizáció nem vesz figyelembe egy fontos kérdést: érdemes lehet-e várakozni egy kereszteződéshez érkezve és csak adott idő eltelte után továbbindulni, növelheti-e a várakozás az időben érkezés esélyeit.

Ennek biztosítására Samaranayake, Blandin és Bayen (2011) a következő feltételt definiálják:

4.1.1. Definíció. (Sztochasztikus FIFO feltétel) Jelölje egy $\mathcal{P}$ sétára $U_{\mathcal{P}}^{(t_1)}$ a rajta való keresztülhaladás idejének eloszlásfüggvényét, t_1 időpillanatban indulva. Egy gráf teljesíti a *sztochasztikus FIFO*[1] *feltételt*, ha minden $\mathcal{P}$ sétára

$$U_{\mathcal{P}}^{(t_1)}(T) \geq U_{\mathcal{P}}^{(t_2)}(T - (t_2 - t_1)) \qquad \forall\, 0 \leq t_1 \leq t_2, \forall\, 0 \leq T \tag{4.1.1}$$

Majd megfogalmazzák a következő tételt:

4.1.2. Tétel. *Ha egy gráf teljesíti a sztochasztikus FIFO feltételt, akkor egy a SOTA probléma optimalitási feltétele szerint egy pontban való várakozás nem növelheti az időben érkezés esélyét.*

A tétel bizonyításával és a kérdéshez kapcsolódó egyéb fogalmakkal itt most nem foglalkozunk.

4.2. Nem független eloszlások

Sok valós úthálózaton az egyes utcákon való áthaladás valószínűsíthető ideje nem független egymástól, azaz a c_{ij} változók nem függetlenek. Ilyen lehet például két olyan él, amely valójában ugyanannak az utcának két szakasza vagy két egymásból ágazó utca: ha az egyiken nagy a forgalom, a forgalmat kerülni próbáló autósok miatt a másikon is megnőhet.

Samaranayake, Blandin és Bayen (2011) felteszik az úgynevezett Markov feltételt, ami itt annyit jelent, hogy feltéve, hogy a (r,i) él megtétele után szándékozunk az (i,j) élen továbbhaladni, akkor a c_{ij} változó eloszlása a $c_{r,i}$ értéket feltéve már független az összes többi, az út során nyert információtól.

Ekkor a keresett megoldás $q : V \times \mathbb{R} \times V \times \mathbb{R} \to V \cup \{\emptyset\}$ alakú, ahol $q(r, t_1, i, T)$ (vagy $q_i(r, t_0, T)$) a továbbhaladás optimális irányát jelzi, feltéve, hogy az r pontból érkezünk az i pontba és az (r,i) élen való áthaladás t_0 időt vett igénybe.

A szerzők ezután belátták, hogy mivel ekkor minden pontban ki kell számítani a beérkezés irányától és az utolsó megtett élen eltöltött időtől függően az u függvényt, ez a megoldó algoritmusok nagyságrendjében egy $L\varrho$ szorzót jelent (ahol ϱ a gráf pontjainak maximális be-fokszáma). Ez olyan nagy, hogy gyakorlati feladatokban sokszor már nem praktikus.

Felvetik viszont, hogy természetesen a feltétel t változóján az idő diszkretizációja tetszőleges lehet: például kétfelé választhatjuk a beérkezés idejét aszerint, hogy az előző útszakaszon a felvett idő forgalmi dugó esetén jellemző avagy sem. Ezzel minösszesen egy 2ϱ szorzóval kell több számolást végeznie az algoritmusnak.

4.3. Javítás a minimális útidők növelésével

A korábbi fejezetekben láthattuk, hogy mind a darabokban számolt gyors konvolúció, mind a ZDC algoritmus esetében a d érték növekedésével csökkent a futásidő. Az erre tett megszorítás – miszerint az az

[1] Az angol *First in first out*, azaz nagyjából „aki először indul, először érkezik” rövidítése

összes élen a lehetséges áthaladási költségek minimumával egyezik meg – némileg pazarló. Az algoritmust tovább tudjuk gyorsítani azzal, hogy az egyes éleken a nekik megfelelő d_{ij} értékekkel számolunk.

Ekkor viszont ügyelnünk kell arra, hogy az algoritmus csak olyan értékekből számoljon, amelyeket korábbi számolások eredményeként pontosan ismerünk. Samaranayake, Blandin és Bayen (2011) a következő tételt fogalmazzák meg:

4.3.1. Tétel. *Legyen $\beta_{ij} = d_{ij} - \epsilon$ ahol $0 < \epsilon < d_{ij}$ és τ_i az az érték, ameddig az u_i függvény értékét ismerjük. Ekkor ahhoz, hogy az algoritmus az u_i függvényen helyes eredményt adjon, a*

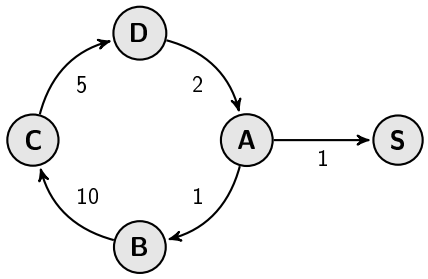
$$\tau_i \leq \min_{(i,j)\in A}(\tau_j + d_{ij})$$

feltételnek teljesülnie kell az algoritmus futásideje alatt.

Bizonyítás. Tegyük fel, hogy a feltétel megszeghető. Ekkor az algoritmus futásideje alatt valamikor teljesül, hogy $\tau_i > \min_{(i,j)\in A}(\tau_j + d_{ij})$ valamely i-re, azaz valamely (i,j) párra $\tau_i - \tau_j > d_{ij}$. Ebből azonban következik, hogy valamely t értékre $u_i(t)$ olyan $u_j(t-\omega)$ és $p_{ij}(\omega)$ értékekből lett kiszámítva, hogy a szorzatuk értéke ismeretlen, hiszen $\omega > d_{ij}$ miatt $p_{ij} > 0$, viszont $\tau_i - \tau_j > d_{ij}$ miatt $u_j(t-\omega)$ értéke ismeretlen. □

Ha viszont feloldjuk azt a megkötést, hogy mindig minden pontban csak d értékkel tovább számoljuk az u függvényeket, akkor számít az is, hogy milyen sorrendben dolgozzuk fel a gráfot. Ennek alátámasztására Samaranayake, Blandin és Bayen (2011) a következő példát hozzák:

4.3.2. Példa. Tekintsük a 4.1. ábrát. Ha minden lépésnél a megengedett legnagyobb τ értékig számítjuk



4.1. ábra. Gráf a különböző feldolgozási sorrendek hatékonyságának illusztrálására. A megfelelő d értékek az élek mellett vannak feltűntetve.

ki az u függvények értékeit, akkor a sorrendtől függően más-más értékekig tudjuk ezeket meghatározni. Két ilyen sorrend eredményeit illusztrálja a 4.1. táblázat. Látható, hogy a pontokat (a,b,c,d) sorrendben feldolgozva sokkal tovább tudja meghatározni a függvények értékeit ugyanannyi iteráció alatt.

Az optimális sorrend valamint az egyes lépésekben kiszámolt intervallumok hosszának meghatározására Samaranayake, Blandin és Bayen (2011) egy $\mathcal{O}\left(\frac{mT}{\delta}\log(n)\right)$ futásidejű algoritmust javasol, amivel numerikus tesztek során jelentős javulást érnek el az algoritmus futásidejében. Az algoritmust itt nem tárgyaljuk.

Iteráció	a	b	c	d
1	1	3	8	18
2	19	21	24	36
3	37	39	46	57
4	55	57	62	72

Iteráció	d	c	b	a
1	10	5	5	11
2	15	7	13	16
3	17	18	18	18
4	28	23	20	29

4.1. táblázat. Feldolgozás sorrendje a baloldalon: (a, b, c, d). Jobboldalon: (d, c, b, a)

4.4. A keresési tér szűkítése

Az eddig adott megoldásokban nem vettük figyelembe, hogy nem feltétlenül szükséges minden pontra és minden időkeretre megoldani a feladatot. Amennyiben egy teljes úthálózatra, illetve azon minden időkeretre, indulási és érkezési pontra kiszámítjuk a feladat megoldását, akkor elég lehet egy utazónak azt a rendelkezésére bocsátani, így nem szükséges magának kiszámolnia. Ez nagy számítási kapacitást igényel, viszont csak egyszer kell kiszámítani. Könnyen elképzelhető ráadásul, hogy egyes utazók számítási kapacitása limitált, ami ezt a fajta megoldást vonzóvá teheti.

Másrészről, a feladat futásideje $\mathcal{O}(N)$ nagyságrenddel nagyobb, hiszen azt minden ponttal, mint uticéllal újra kell számolni (továbbá az időkeretet is valószínűleg nagyobbnak kell venni, mint amire az esetek többségében szükség lehet). Nem lehetséges továbbá például forgalomfigyelők segítségével az egyes útvonalak áthaladási idejének az eloszlását az éppen aktuális szituációhoz igazítani.

Ha tehát valamilyen okból csak egy konkrét pontból konkrét időkerettel indulva szeretnénk meghatározni az optimális megoldást, akkor viszont sok esetben nem szükséges minden u_i függvényt minden $t \leq T$-re kiszámolni, egyes útvonalakat ki lehet zárni a keresési térből. (Ráadásul az algoritmus jellege miatt a megoldás szükség esetén tovább bővíthető, az addigi eredmények felhasználásával).

Samaranayake, Blandin és Bayen (2011) a következő lehetőséget vetik fel: ha minden útvonalon létezik egy minimális lehetséges áthaladási költség, akkor ezekkel, mint élsúlyokkal le tudjuk szűkíteni a feldolgozandó gráf méretét. Erről a következő tételt fogalmazzák meg:

4.4.1. Tétel. (A keresési tér szűkítése) *Legyen j tetszőleges pont a gráfban, S az uticél, i a pont ahonnan az utazó indul. Legyen a d_{ij} értékeket élsúlyként véve a legrövidebb i-ből j-be vezető út költsége α_{ij}, valamint a legrövidebb j-ből S-be vezető út költsége α_{jS}. Ekkor a következők teljesülnek:*

1. *Ha egy j pontra $\alpha_{ij} + \alpha_{jS}$ nagyobb, mint a rendelkezésre álló időkeret, ezt a pontot elhagyhatjuk a gráfból a feladat megoldásának számításakor.*

2. *Egy j ponthoz tartozó u_j függvénynek csak az $\alpha_{iS} \leq t \leq (T - \alpha_{ij})$ intervallumon felvett értékeire van szükségünk a megoldás kiszámításához.*

Bizonyítás.

1. A feltétel azt jelenti, hogy a legrövidebb lehetséges idő, ami alatt a j ponton keresztülhaladva i-ből S-be érhet az utazó hosszabb, mint az adott időkeret. Ekkor nyilvánvalóan elhagyhatjuk a pontot, hiszen az optimális megoldás szerint haladva nem érintheti azt az utazó semmilyen esetben.

2. A $t < \alpha_{ij}$ értékekre $u_j(t) = 0$, hiszen a legrövidebb lehetséges út is több, mint a rendelkezésre álló időkeret, tehát nem szükséges az értékeket kiszámítani.

   A $t > (T - \alpha j S)$ értékekre nem lehetséges az, hogy i-ből T időkerettel indulva j-be jutunk és t időkeretünk marad, így nem szükséges az $u_j(t)$ értékeket kiszámolnunk, hiszen az optimális megoldás kiszámításához ezekre az értékekre nem lehet szüksége az algoritmusnak.

Ezzel mind a két állítást beláttuk. □

A távolságok meghatározása például Dijkstra algoritmussal történhet – egyszer i-ből, egyszer pedig S-ből. Ennek a futásideje gyakorlatban jól alkalmazható, $\mathcal{O}(N \log(N))$ nagyságrendű, tehát érdemes lehet használni, ha az utána következő algoritmus futásidejét kellőképpen lerövidíti. Ez természetesen a konkrét feladattól is függ.

5. fejezet

Numerikus adatok

A korábbi fejezetekben többféle algoritmust írtunk le valamint többféle stratégiát az ezekben megjelenő konvolúciók kiszámítására. Itt ezeket fogjuk kipróbálni és összehasonlítani egymással.

A tesztelésre használt programokat C++ programozási nyelven készítettük el és azokat Linux operációs rendszer alatt fordítottuk és futtattuk egy kétmagos, 64 bites, 1.3GHZ-es processzorral felszerelt Lenovo laptopon. A fordításhoz a GCC fordítót használtuk az *-O3* optimalizációs kapcsolóval. A programokban a diszkrét Fourier-transzformált kiszámítására minden esetben az *FFTS – The Fastest Fourier Transform in the South* C++ könytár által biztosított gyors Fourier-transzformációt megvalósító algoritmust használtuk.

5.1. A brute-force és a gyors konvolúció összehasonlítása

A definíció szerint számolt konvolúciót, vagy más néven *brute-force* konvolúciót rendkívül egyszerű megvalósítani két egymásba ágyazott ciklus segítségével.

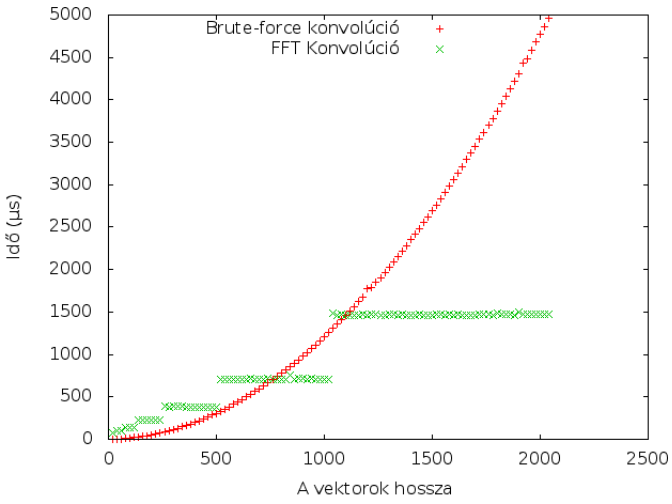
A gyors Fourier-transzformációt megvalósító algoritmusok sok esetben csak kettőhatvány hosszúságú vektorokra működnek és ez a helyzet az általunk használt FFTS könyvtár FFT algoritmusával is. Így amennyiben két nem-kettőhatvány n hosszúságú vektort szeretnénk konvolválni, azokat ki kell egészítenünk először azonos, $m = 2^{\lceil \log_2(n) \rceil}$ hosszúságú vektorokká (majd ez után még a 3. fejezetben tárgyalt módon kiegészíteni ezeket 0 tagokkal $2m$ hosszúságú vektorokká).

A gyors konvolúció kiszámításánál tehát sok „járulékos költséggel" kell számolnunk, ami miatt kis n értékekre hosszabb ideig tart velük a számolás, mint a brute-force megoldással. Ezek viszont az algoritmus nagyságrendjén nem változtatnak, keressük tehát azt az n_0-t, amitől kezdve érdemesebb az FFT-t használó algoritmussal számolnunk. A két algoritmus futásidejének összehasonlítása az 5.1. ábrán látható.

Vegyük észre, hogy a gyors konvolúció valójában többet számol ki, mint a brute-force megoldás: míg az utóbbi a

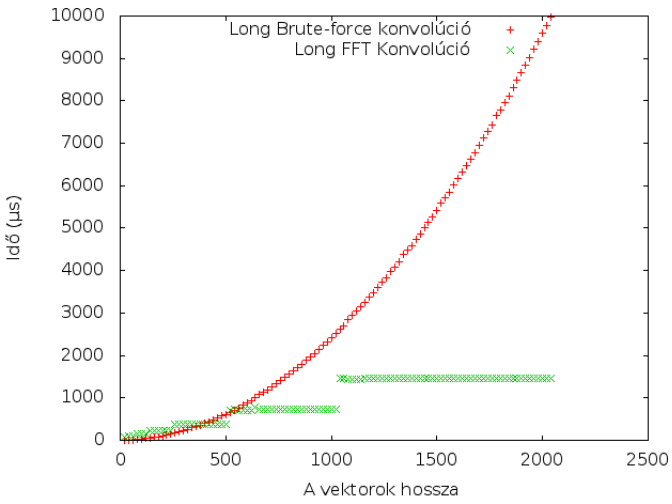
$$v[n] = \sum_{i=0}^{n} a[i]b[n-i]$$

értéket számolja ki minden $n = 0, 1, \ldots, m$-re, addig a gyors konvolúció ugyanezt minden $n = 0, 1, \ldots, 2m$



5.1. ábra. Brute-force és gyors konvolúció összehasonlítása

értékre kiszámolja (emlékezzünk, a vektorok ki lettek egészítve m darab 0 értékű koordinátával is, továbbá az indexeket ciklikusan értelmezzük). Ezekre az értékekre ugyan a feladatban nincsen szükségünk, viszont a Zero Delay Convolution algoritmusban lesz. Amennybien a brute-force megoldással is szükséges kiszámolni ezeket az értékeket, már kisebb értékekre is érdemes a (változatlan futásidejű) gyors konvolúciós algoritmus használni, lásd az 5.2. ábrát.



5.2. ábra. Hosszú brute-force és gyors konvolúció összehasonlítása. (Vigyázat, az 5.1 ábrához képes nem a gyors konvolúció sebessége nőtt, hanem a brute-force konvolúció sebessége csökkent.)

A továbbiakban amikor gyors konvolúcióról beszélünk, egy olyan függvényt értünk, amely kis n-ekre brute-force, nagy n-ekre FFT-vel megvalósított konvolúciót használ, a fenti grafikonok szerint.

5.2. Brute-force konvolúció és gyors konvolúció az idő szerinti iterációt használó algoritmusban

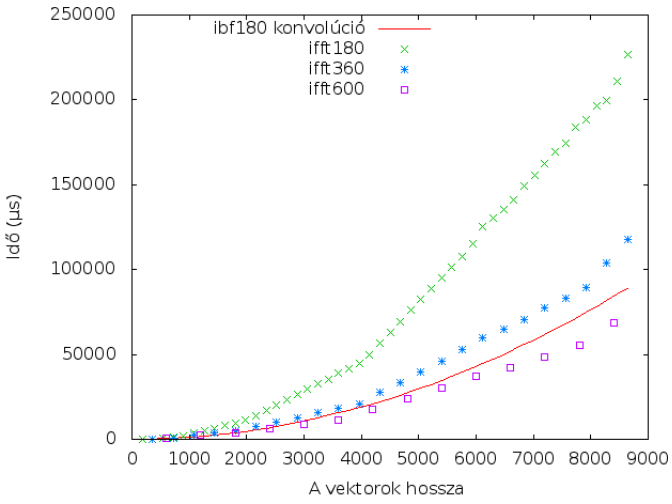
Emlékezzünk vissza, a 3.2. fejezetben a konvolúciókat FFT segítségével számoltuk, egyre növekvő méretben, mindig újraszámolva a korábbi eredményeket is (ezentúl: IFFT[1] konvolúcióval). Az, hogy ezzel együtt is gyorsabb tud-e lenni, mint a brute-force módszer, nagyon szorosan függ a d értéktől: mindig $\frac{d}{\delta}$ új elemre kell kiegészíteni a konvolúciót, tehát összesen $\frac{T}{d} = D$ darab, egyenletesen növekvő hosszúságú konvolúciót kell kiszámolnunk.

Meg kell jegyeznünk, hogy habár a brute-force módszerrel ugyanazokat a számításokat végezzük el lényegében az idő szerinti iterációs esetben, a számítások $\frac{d}{\delta}$ hosszú blokkokra osztása egy konkrét implementáció során a számítógépnek valamennyi többletmunkát jelent. Ez a futásidő-növekedés nem jelentős, de az összehasonlítás gyakorlati használhatósága érdekében innentől lehetőség szerint a megnövekedett futásidejű brute-force konvolúcióval (ezentúl: IBF konvolúció) fogunk számolni. Az egyre növekvő hosszúságú valamint az egyben végzett számítások közötti futásidő-igény különbség az 5.3. ábrán látható.

5.3. ábra. Darabokban számolt brute-force ($d = 16$) és egyben kiszámolt brute-force összehasonlítása

Az 5.4 ábra mutatja az előbb leírt IBF konvolúció, valamint az IFFT konvolúció összehasonlítását különböző d értékekre. Látható, hogy ugyan elképzelhető olyan d érték, amellyel bizonyos n hosszúságú vektorokra jobban teljesít, de ehhez nagyon nagy d érték szükséges. A példában látott $d = 600$ érték például abban az esetben fordulhat elő, ha az időt 1 másodperc hosszú intervallumokra diszkretizáljuk és az úthálózaton az egyes utcákon való minimális lehetséges áthaladási idő legalább 10 perc. Ez ugyan valamennyire életszerű, de nem mondható gyakorinak (egy ilyen méretű úthálózaton valószínűleg nincs szükség 1 másodperc pontosságú számolásokra) és az elért javulás sem igazán jelentős a brute-force megoldáshoz képest, valamint tudjuk, hogy a $\log(N)$-es szorzó miatt ez nagy n értékekre nemhogy növekedne, de csökken.

[1] Az IFFT jelölést a szakirodalomban sokszor az inverz (gyors) Fourier-transzformáció jelölésére szokták használni. Mivel mi ezt nem tesszük, ezért nem okoz félreértést ha a darabokban számolt konvolúcó jelölésére használjuk.



5.4. ábra. Darabokban számolt gyors konvolúció ($d = 180, 360, 600$) és darabokban számolt brute-force ($d = 180$) összehasonlítása

5.3. Brute-force konvolúció és Zero Delay Convolution összehasonlítása

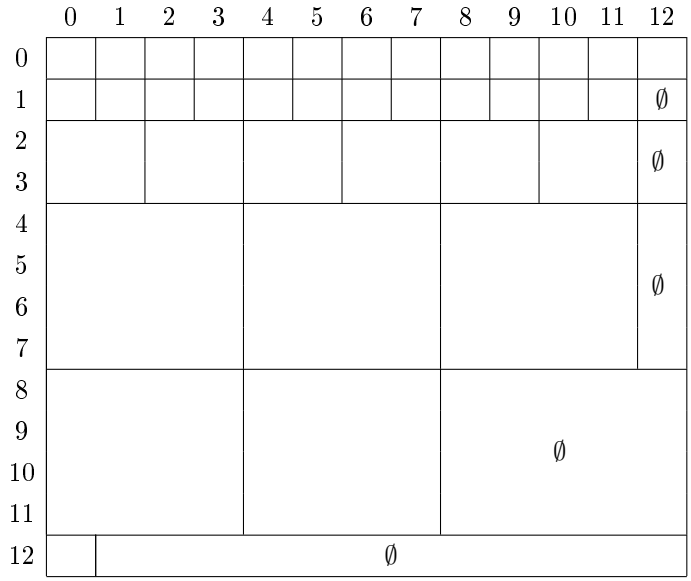
A tárgyalt módszerek közül a Zero Delay Convoltion (ezentúl: ZDC) mondható a legígéretesebbnek. Ez ugyanis minden értéket csak egyszer számol ki és nagyságrendben is jobb, mint a brute-force megoldás. A konkrét implementációja viszont sokkal bonyolultabb, mint az utóbbinak, így a sok járulékos költség miatt kis n értékekre lassabban fut, mint a brute-force megoldás.

Az algoritmusnak először blokkokra kell osztania a kiszámítandó konvolúciót. Ennek a módját bemutattuk a 3.3 fejezetben, viszont az ismertetett megoldás csak kettőhatványokra működik tökéletesen.

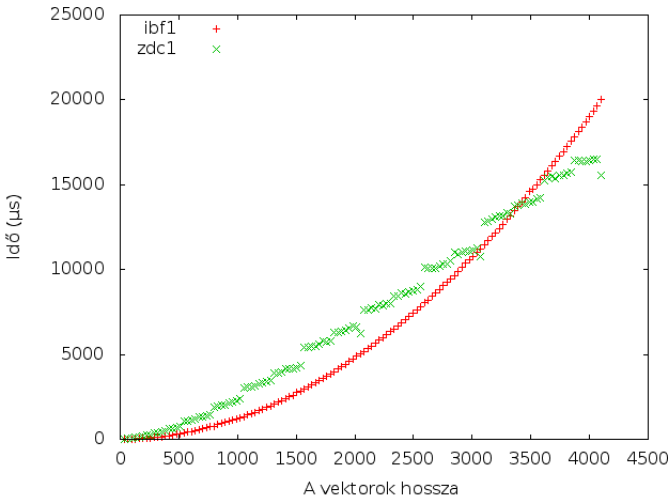
A mi implementációnk a program a futása során először megtervezi adott n-re a kiszámítandó konvolúciók elhelyezkedését, méretét, a számítás elvégzésének idejét (azaz, hogy mikor áll már rendelkezésre elég információ annak kiszámításához) valamint mérettől függően a módját (gyors konvolúció vagy brute-force konvolúció). Ezt a következőképpen teszi:

1. Létrehozza az első két sort.
2. A 3.3. fejezetben leírt módon létrehoz egyre növekvő méretű blokk-sorokat, fenntartva a megkötést, hogy azok kiszámíthatóak legyenek, mire az eredményükre szükség van. Amennyiben egy blokk eredményére nincs szükség, azt nem hozza létre. Mindezt addig folytatja, amíg az egyre növekvő méretű blokkok elférnek.
3. Amikor már nem fér el a következő, kétszeres méretű blokk, megpróbál még egy ugyanakkora, majd egyre csökkenő méretű blokksorokat létrehozni.
4. Ha nem maradt felosztatlan sor, a tervező algoritmus leáll.

Erre láthatunk egy példát 5.5. ábrán.



5.5. ábra. Két 13 hosszúságú vektor konvolúciójának blokkokra osztása. Az ∅-zal jelölt részek nem kerülnek kiszámításra.



5.6. ábra. Darabokban kiszámított brute-force (d=1) és ZDC (d=1) összehasonlítása

Külön említést érdemel, hogy az FFT-vel végzett konvolúcióknak egy részét már a terv készítése során kiszámítja: az ismert vektor egy szakaszának a DFT-jét elég egyszer kiszámítani, majd egy sor minden blokkjának feldolgozásakor azzal pontonként szorozni a megfelelő (másik) vektor DFT-jét.

Ezzel a módszerrel tetszőleges hosszúságú vektorokra tudjuk alkalmazni a ZDC algoritmust. Különböző méreteken vett összehasonlítása az IBF algoritmussal az 5.6. ábrán látható.

Látható, hogy ugyan kis értékekre lassabban fut, de nagyjából $n = 3500$ körül lehagyja a brute-force megoldást hatákonyságban, és a nagyságrend elemzésből tudjuk, hogy n növekedésével a különbség is növekedni fog.

Figyelmes szemlélő észreveheti, hogy ZDC-nek a grafikonon látott futásideje nem monoton növekvő. Ez sajnos nem a véletlen műve: valóban, a fent leírt kiegészítése az eredeti algoritmusnak nem optimális minden n-re. Ha ugyanis n valamivel, de nem sokkal egy kettőhatvány alá esik, akkor a fent leírt megoldás helyett érdemesebb lehet inkább kiegészíteni nullákkal a következő kettőhatványra és azzal számolni. A különbség kettőhatvány és nem-kettőhatvány között arányaiban nem olyan számottevő, mint amekkora ugrásokat például az 5.1. ábrán láthattunk.

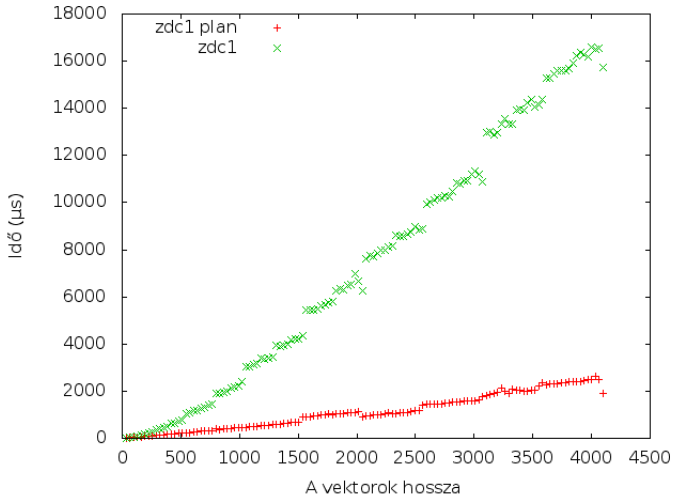
Azt, hogy mikortól érdemes inkább kiegészíteni a vektorok méretét nem triviális meghatározni, de azt sem, hogy milyen hosszúságig: gondoljuk bele, hogy nem csak 2^n alakra fut aránylag gyorsan az algoritmus, hanem például $2^n + 2^{(n-1)}$-re is: ekkor ugyanis elég a legnagyobb méretű blokkokból még egy sornyit hozzátennünk.

Egy másik optimalizációs lépés lehetne, hogy azokat a blokkokat, amelyek konvolúciójának például csak az első felére vagy az első pár elemére van szükségünk kisebb blokkokkal helyettesítjük. Ilyen lehet például 5.5. ábrán a 4. sor 8. eleménél kezdődő 4×4 méretű blokk. Az optimális stratégia itt sem nyilvánvaló: ekkor ugyanis azzal is számolni kell, hogy ezekhez a kisebb méretű blokkokhoz viszont nem tudjuk felhasználni a terv elkészítése során kiszámított, az ismert vektor egy szakaszához tartozó DFT-t.

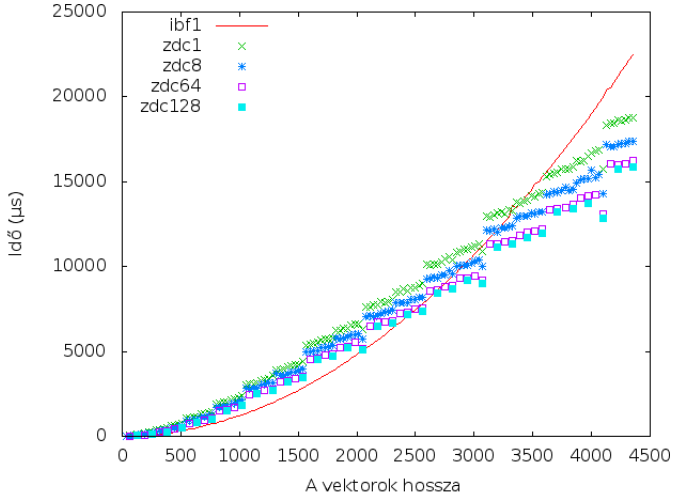
Végül érdemes meggondolni, hogy ha több azonos méretű konvolúciót számolunk ZDC-vel, akkor érdemes lehet különválasztani a tervezés és a számolás lépéseit, hiszen felesleges ugyanazt a felosztást minden új konvolúcióhoz kiszámítani. A tervezéshez szükséges időt különböző méretű vektorok esetén az 5.7. ábra mutatja (a tervezésbe itt nincs beleszámítva a korábban említett DFT-k kiszámítása).

A ZDC algoritmus hatékonyságán is javít a nagy d érték. Különböző esetek összehasonlítását az 5.8. ábra mutatja. Láthatjuk, hogy itt jóval kevésbé számít a d érték mint a korábbi esetben, de itt is gyorsít valamelyest az algoritmus futásidején.

Összefoglalásképpen tehát a tesztelt implementációval a ZDC algoritmus hatékonyságában ($d = 1$ esetén) nagyjából $n = 3500$ körül hagyja le a brute-force megoldást. Ez $\delta = 1$ (másodperc) mellett nagyjából egy óra, tehát könnyen elképzelhető életszerű helyzetekben ilyen, vagy ennél hosszabb konvolúciók kiszámítása. Az implementáció futásideje tovább javítható a fent említett módszerekkel valamint a d érték növelésével.



5.7. ábra. A ZDC algoritmus futásidejének tervezéssel eltöltött részének és a ZDC algoritmus futásidejének összehasonlítása



5.8. ábra. A ZDC futásidejének összehasonlítása különböző d értékekre. Látható, hogy a javulás d függvényében nem lineáris (a 3.3. fejezetben kiderül, hogy $\log^2(d)$ szerint változik).

Összefoglalás

A dolgozatban bevezettük a SOTA probléma alapfeladatát és leírtunk annak megoldására egy úgynevezett fokozatosan közelítő algoritmust (ennek futásideje $\mathcal{O}\left(Km\left(\frac{T}{\delta}\right)^2\right)$ nagyságrendű volt). Bizonyítottuk annak konvergenciáját és beláttunk két, a megoldás hibájának becslését segítő tételt.

Ezután bevezettük az idő szerinti iteráció módszerét és az ahhoz kapcsolódó, a feladatot a diszkrét valamint a folytonos modellben megoldó algoritmusokat, melyek futásideje jelentősen kisebb volt mint a fokozatos közelítéses algoritmusé ($\mathcal{O}\left(m\left(\frac{T}{\delta}\right)^2\right)$ nagyságrendű).

A konvolúciók kiszámításának hatékonysága érdekében ezután megvizsgáltuk két gyors Fourier-transzformáltat használó verzióját az algoritmusnak, melyek $\mathcal{O}\left(m\frac{T^2}{\delta d}\log\left(\frac{T}{\delta}\right)\right)$ valamint $\mathcal{O}\left(m\frac{T}{\delta}\log^2\left(\frac{T}{\delta}\right)\right)$ időben futottak.

A konvolúciók elvégzésének leírt módjait konkrét implementációkkal teszteltük, különböző hosszúságú vektorokon mérve azok sebességét.

A dolgozatban leírt algoritmusok hatékonyan megoldják a SOTA problémát. A 4. és az 5.3. fejezetekben felvetett lehetőségek mentén az algoritmus egy implementációja konkrét feladattól függően tovább bővíthető illetve gyorsítható lehet.

Mindenképpen említést érdemel viszont, hogy a leírt algoritmusok navigációs rendszerekben főképp az autóforgalomban való útvonaltervezésre alkalmasak, holott a SOTA alapfeladata más modellekben is megfogalmazható. Ilyen például tömegközlekedési járművek igénybevételének esete, ahol nem elég általános a modell ahhoz, hogy használható legyen.

Az említett esetben például az „élek”, azaz a járművek nem állnak rendelkezésre minden időpillanatban, hanem azokra várni kell. A várakozási idő eloszlását természetesen bele lehetne számolni a jármű menetidejének az eloszlásába (és ha ismert a menetrend, akkor ez az eloszlás annak megfelelően változhat az időben), viszont az utazónak ekkor rendelkezésére áll olyan információ, ami az eddig tárgyalt esetekben nem: megteheti ugyanis például, hogy egy megállóban állva az első érkező járműre száll fel. Ezzel azt használja ki, hogy már az útvonal kiválasztása előtt kap további információt egyes eloszlásokról, ami a mi modellünkben (a többi eloszlástól függetlenül) nem történhet meg.

Felhasznált irodalom

Fan, Yueyue és Yu Nie (2006a). „Optimal Routing for Maximizing the Travel Time Reliability". In: *Networks and Spatial Economics* 6.3-4, pp. 333–344. ISSN: 1566-113X.

Fan, Yueyue és Yu Nie (2006b). „Arriving-on-time problem: discrete algorithm that ensures convergence". In: *Transportation Research Record: Journal of the Transportation Research Board* 1964.-1, pp. 193–200.

Samaranayake, Samitha, Sebastien Blandin és Alex Bayen (2011). „A Tractable Class of Algorithms for Reliable Routing in Stochastic Networks". In: *Procedia - Social and Behavioral Sciences* 17. Papers selected for the 19th International Symposium on Transportation and Traffic Theory, pp. 341 –363. ISSN: 1877-0428.

Samaranayake, Samitha, Sebastien Blandin és Alex Bayen (2012). „Speedup Techniques for the Stochastic on-time Arrival Problem". In: *12th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*. Ed. by Daniel Delling és Leo Liberti. Vol. 25. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 83–96. ISBN: 978-3-939897-45-3.

Gardner, William G. (1995). „Efficient Convolution without Input-Output Delay". In: *Journal of the Audio Engineering Society* 43 (3), pp. 127–136.

Osgood, Prof. Brad (2007). *The Fourier Transform and its Applications*. Lecture Notes for EE 261. Stanford University, Electrical Engineering Department. URL: `http://see.stanford.edu/materials/lsoftaee261/book-fall-07.pdf`.

Blake, Anthony. *FFTS – The Fastest Fourier Transform in the South*. Version 0.7. URL: `http://anthonix.com/ffts/`.