Eötvös Loránd Tudományegyetem

Természettudományi Kar

Tossenberger Tamás

Autoenkóder alapú generatív modellek javítása

MSc Matematikus Szakdolgozat

Témavezető:

Varga Dániel

MTA Rényi Alfréd Matematikai Kutatóintézet

Belső konzulens:

Lukács András

Számítógéptudományi Tanszék

Budapest, 2017

Köszönetnyilvánítás

Ezúton is szeretném megköszönni témavezetőmnek, Varga Dánielnek, valamint Kiss Melinda Flórának a közös munkát. Továbbá mindenkinek, aki a dolgozat elkészítésében támogatott.

Tartalomjegyzék

Bevezetés

A dolgozatban egy gyakorlati, deep learning eredetű probléma megoldását tárgyaljuk, amely [1] alapján vetődött fel. A probléma megoldásához szükségünk van egy gyors online algoritmusra, amely egy közel-minimális súlyú közel-teljes párosítást talál élsúlyozott teljes páros gráfokban. Ennek a problémának a megoldásához bemutatunk egy maximális súlyú párosítást kereső online algoritmust [7], és ennek elméleti hátterét [9], [10], [11].

Mutatunk továbbá egy algoritmust [15], amelynek segítségével a teljes gráfokat ritkítani tudjuk oly módon, hogy se a maximális párosítás, se pedig a minimális súlyú maximális párosítás ne változzon sokat. Ez utóbbihoz kihasználtuk, hogy a megoldandó gyakorlati problémában a páros gráf csúcsaihoz rendelhető egy-egy pont egy metrikus térben, hogy egy él súlya pontosan a két végpontjának a metrikus térbeli távolsága.

Valamint bemutatjuk az irányított Even-Shiloach algoritmus egy tetszőleges valós élsúlyokra vett általánosítását, amely a dolgozat eredménye. Ezen kívül bemutatjuk, hogy [7] algoritmusa hogyan módosítható az előbbi algoritmus felhasználásával, és hogy így egy közel-minimális súlyú közel-teljes párosítást kereső algoritmust nyerhetünk. Ennek gyorsaságát, és közelítésének mértékét empirikusan kiértékeljük.

Ehhez a [4] deep learning könyvtárból elágazott [5] könyvtárban implementáltuk a prezentált súlyozott és súlyozatlan párosítási algoritmusokat teszteléshez. A teszteredmények a dolgozat végén szerepelnek, amelyek [5] kódjaival reprodukálhatóak.

A dolgozatkészítés keretében, szintén autoenkóder alapú generatív modellek javítására, készült egy másik projekt is, amely a látens pontfelhő eloszlását közelíti normális eloszlások összegével, majd ezen eloszlás sűrűségfüggvényéből származó veszteségfüggvénnyel tanítja a hálót, ezzel létrehozva egy *unsupervised learning* algoritmust. Az ehhez megírt kódok, és egy 5 oldalas technikai összefoglaló megtalálható [6] hivatkozáson, azonban a dolgozatnak nem ez a projekt a fókusza, így tovább nem itt tárgyaljuk. Érdekes lehet azonban ezt a projektet is tovább kutatni a technikai összefoglalóban bemutatott ötletekkel, valamint a [3]-as cikkre alkalmazni, mint klaszterező eljárás.

1. fejezet

Noise as target módszer a deep learning-ben

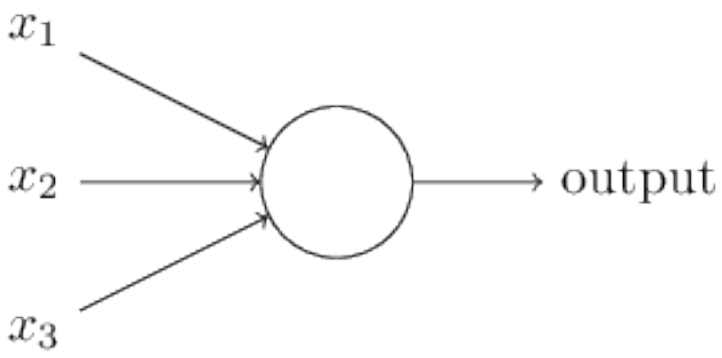
Ebben a fejezetben tisztázzuk a deep learning alapjainak a dolgozat számára legfontosabb részeit. Az első részben [12] és [13] online könyvek alapján íródtak. Ebben a szekcióban felhasználjuk [12] néhány ábráját is. Átfogóbb bevezetőhöz ajánljuk az előbbi két online könyv olvasását, a dolgozatban azonban csak a legszükségesebb fogalmakat vezetjük be ahhoz, hogy egy gyakorlatban felmerülő deep learning problémának egy kombinatorikus optimalizálási részfeladatát azonosíthassuk, majd megoldhassuk. A fejezet további célja, hogy a problémafelvetés és motiváció a deep learning háttérrel nem rendelkező olvasó számára is érthető legyen.

1.1. Deep learning bevezető

1.1.1. Mesterséges neuronok, és aktivációs függvények

A neurális hálók alapelemei a mesterséges neuronok. Ezek közül a legegyszerűbb az úgynevezett perceptron.

Legyen $x_1, x_2, \ldots, x_n \in \mathbb{R}$, $w_1, w_2, \ldots, w_n \in \mathbb{R}$ és $b \in \mathbb{R}$, ekkor egy perceptron mesterséges neuron az $x_1, x_2, \ldots, x_n$ inputokra a $w_1, w_2, \ldots, w_n$ súlyokkal és b „bias”-szal 0 outputot ad, ha $\sum_{i=1}^{n} w_i x_i + b \leq 0$, és 1 outputot ad, ha $\sum_{i=1}^{n} w_i x_i + b > 0$. Tehát $w = (w_1, w_2, \ldots, w_n) \in \mathbb{R}^n$ súlyvektorral és $b \in \mathbb{R}$ „bias”-szal rendelkező perceptront a



$p_{(w,b)} : \mathbb{R}^n \mapsto \{0,1\}$ leképezés írja le, ahol $\forall x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ inputra

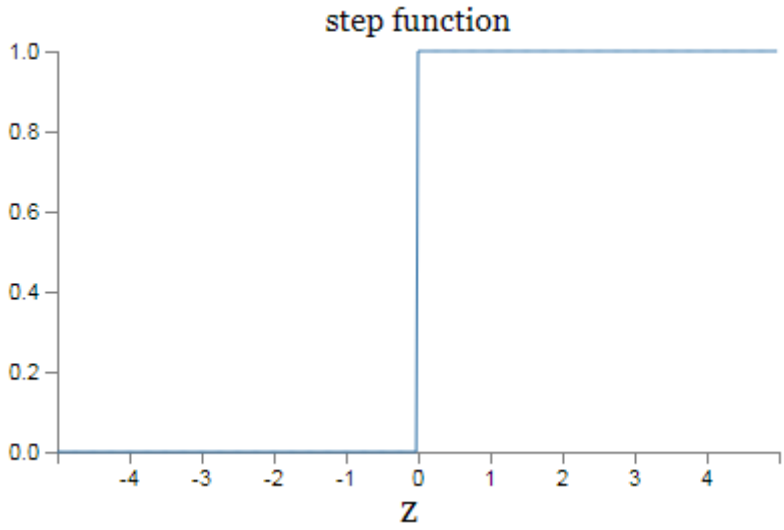
$$p_{(w,b)}(x) = \begin{cases} 0, \text{ha } \langle w, x \rangle + b \leq 0 \\ 1, \text{ha } \langle w, x \rangle + b > 0 \end{cases}$$

Könnyen látható, hogy $w = (-2, -2)$ súlyokra és $b = 3$ bias-ra $p_{(w,b)}(0,0) = 1$, $p_{(w,b)}(0,1) = 1$, $p_{(w,b)}(1,0) = 1$ és $p_{(w,b)}(1,1) = 0$. Tehát ilyen súly és bias választással a perceptron a *NAND* logikai kaput szimulálja; és mivel ez egy univerzális logikai kapu, ezért tetszőleges egyéb logikai kaput is szimulálni tudunk perceptronok összefűzésével.

A perceptronnak azonban megvan az a számunkra negatív tulajdonsága, hogy a súlyok vagy a bias kis megváltoztatásával az output nagyot változhat. Ezt szeretnénk elkerülni úgy, hogy olyan mesterséges neuront akarunk leírni, amelynek outputja az egyes súlyoknak és a bias-nak is folytonos (sőt, lehetőleg sima) függvénye. Vegyük észre, hogy $\forall w, x \in \mathbb{R}^n, b \in \mathbb{R} : p_{(w,b)}(x) = s\left(\langle w, x + b \rangle\right)$, ahol $s : \mathbb{R} \mapsto \mathbb{R}$, $\forall y \in \mathbb{R} : s(y) = \begin{cases} 0, \text{ha } y \leq 0 \\ 1, \text{ha } y > 0 \end{cases}$ lépcsős függvény.

Ezzel szemben, ha s lépcsős függvény helyett $\sigma(y) = \frac{1}{1+e^{-y}}$ sima függvény használatával az $sig_{(w,b)}(x) = \sigma\left(\langle w, x \rangle + b\right)$ úgy nevezett szigmoid-neuront kapjuk. Ekkor viszont az output valóban sima függvénye az egyes súlyoknak és a bias-nak. Az előbbi azért fontos számunkra, mivel a mesterséges neuronokból felépített neurális hálót az egyes neuronok súlyainak és bias-ainak hangolásával tanítjuk.

A σ függvényt úgy is felfoghatjuk, mint s simított változatát, láthatjuk, például, hogy $-\infty$-ben és $+\infty$-ben vett limeszei megegyeznek. A σ aktivációs függvénynek továbbá nagy előnye, hogy deriváltjai könnyen számolhatóak. Ezen kívül sok más aktivációs függvény kerül használatra a gyakorlatban; jelenleg leggyakrabban a $ReLU(y) = \begin{cases} 0, \text{ha } y < 0 \\ y, \text{ha } y \geq 0 \end{cases}$



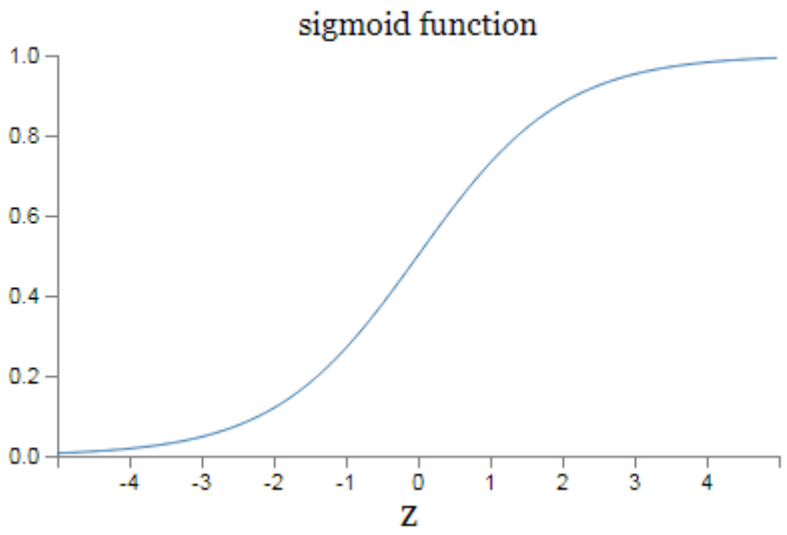
függvény használandó, amely népszerűségének legfőbb oka gyakorlati jellegű, a rá épített hálózatok sikerének köszönhető.

1.1.2. Neurális hálók

Mint már említettük, a neurális hálók alapelemei a mesterséges neuronok. Egy neurális háló rétegek egymásutánjából áll, ahol minden réteg neuronok összessége. Az első réteget az input rétegnek hívjuk, ahol a rétegbeli neuronok inputjait a tanítóadat adja meg. Az utolsó réteg pedig az output réteg. A köztes rétegeket rejtett rétegeknek hívjuk.

Egy rejtett rétegben lévő neuronok az előző réteg neuronjainak outputjaiból kapják az inputjaikat, és az outputjukat a következő rétegnek adják át. (Természetesen nem szükséges, hogy egy neuron az outputját a következő réteg minden neuronjának átadja.)

Az előbb definiált neurális hálókat *feedforward neurális hálóknak* hívjuk, mivel egy neuron outputját csak a következő réteg neuronjai használják fel. Ezen kívül még más neurális hálók is léteznek, némelyek ezek közül jobban hasonlítanak a biológiai neurális hálókhoz. Ilyenek például a *rekurrens neurális hálók*, melyek neuron gráfjában előfordulhatnak körök, ezeknél minden neuron egy lépésen át tüzel, ezzel aktiválva a belőle leszármazó neuronokat, amelyek ezután szintén egy lépésen át tüzelnek, és így tovább. A rekurrens neurális hálók használatosak például gépi szöveg- és hangfeldolgozásra. Képfeldolgozásra azonban feedforward neurális hálókat szokás használni, így mi a továbbiakban csak ezekre szorítkozunk, és a neurális háló kifejezés alatt mindig feedforward neurális hálót fogunk érteni.



1.1.3. Veszteségfüggvények

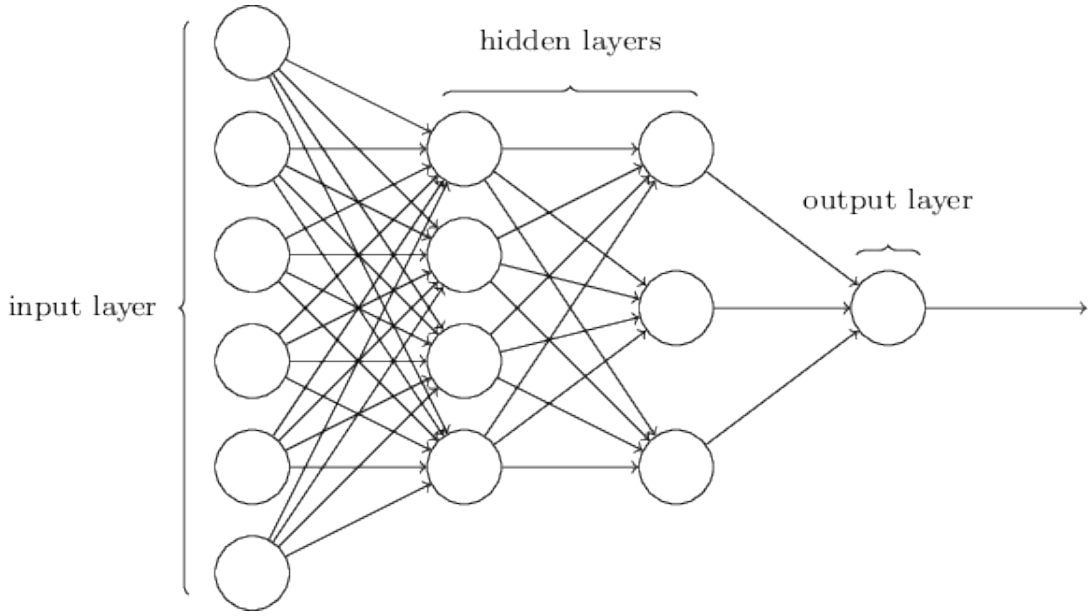
A neurális hálók tanítása alatt azt értjük, hogy az egyes súlyokat és bias-okat (vagy egyéb kalibrálandó változóit a neurális hálónak) próbáljuk úgy kalibrálni, hogy egy veszteségfüggvényt minimalizáljunk.

Ezt a minimalizálást sztochasztikus gradiens módszerrel tesszük meg, és a gradiensek gyors kiszámolására *backpropagation* algoritmussal történik. Ezt a dolgozat nem részletezi. A [12]-es és [13]-as hivatkozásokon részletes leírás található a *backpropagation* algoritmusról, és a sztochasztikus gradiens módszerről.

1.1.4. Konvolúciós hálók

Mint azt már említettük, a neurális háló szomszédos rétegei által alkotott páros gráf (amelyben komponensek csúcsai az egyik, illetve másik rétegben lévő neuronok; él pedig akkor fut két csúcs közt, ha a későbbi réteg csúcsa felhasználja a korábbi réteg csúcsának outputját inputként) tetszőleges lehet. Elsőre úgy tűnhet, hogy a teljes páros gráf minden esetben a legjobb választás, mivel máskülönben a háló megtanulhatná, hogy a nem használt élekhez (közel-) 0 súlyt rendel. Azonban érdemes ezt a terhet levenni a tanítási algoritmusról, amennyiben az adatok szerkezetét ismerve valamilyen jóval ritkább páros gráfokat tudunk használni az általánosság lényeges romlása nélkül.

Manapság képek klasszifikálásában, illetve célzott manipulálásában a *machine learning* algoritmusok gyakran elérik az emberi teljesítőképességet [17]. Ennek fő oka, hogy a

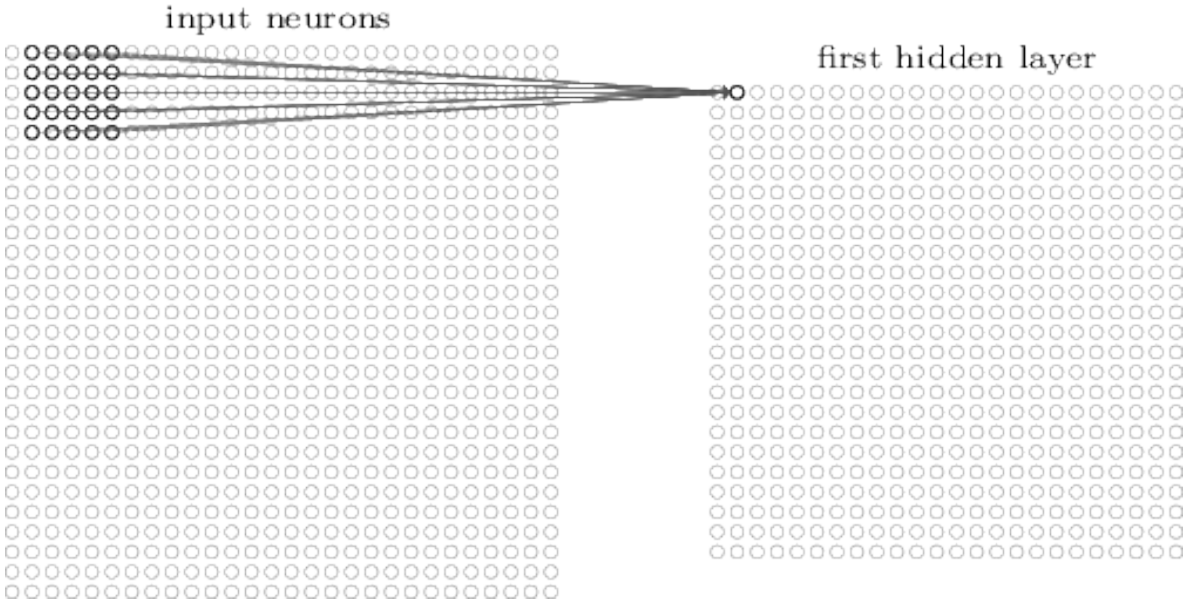


konvolúciós hálókkal jól megfogható a vizuális adatok szerkezete.

Példának álljon a tanító adathalmaz 28×28-as fekete-fehér képekből, így a 784 neuronból álló inputréteg neuronjainak inputjai legyen az egyes pixelek sötétségének mérőszáma. Ekkor, ha az input réteg és az azt követő réteg közt teljes páros gráfot vennénk, azzal a gráf szempontjából mind a 784 input neuronnak szimmetrikus lenne az egymáshoz való viszonya. Természetes képek esetén elvárható például, hogy az egymáshoz közeli pixelek kapcsolata szorosabb legyen mint az egymástól távoliaké. Ezért az input réteg neuronjait egy 28×28-as ponthalmazként ábrázoljuk, és minden kis $k \times k$-as négyzethez, valamely kis rögzített pozitív egészre, legyen most az ábrán látható módon $k = 5$, rendelünk egy neuront a következő rétegből.

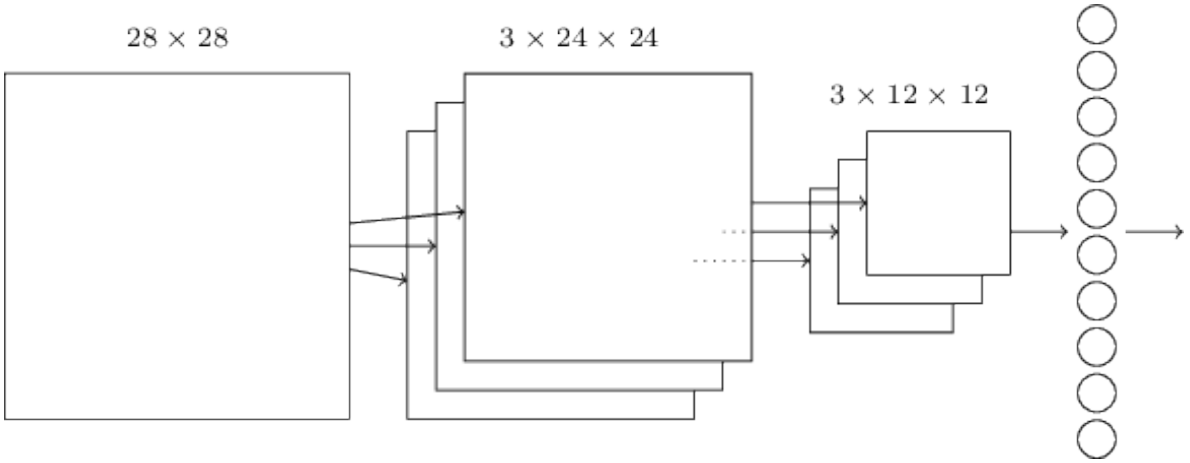
Így tehát esetünkben kapunk egy 24×24-es ponthalmazt mint a következő réteg neuronjai. Azonban azt a megszorítást is megtesszük, hogy az egyes súlyok és bias-ok amik az egyes $k \times k$-as kis négyzetek egyes neuronjaihoz tartoznak, közösek legyenek az összes $k \times k$-as kis négyzetre. Azaz csak k^2 súlyt és 1 bias-t tárolunk és kalibrálunk a két réteg között, ezzel lényegesen lecsökkentve a változók számát.

Például, ha vesszük a $\begin{bmatrix} 1 & 1 & 1 \\ -2 & -2 & -2 \\ 1 & 1 & 1 \end{bmatrix}$ súlymátrixot, akkor ezzel súlyozva egy fekete-fehér kép egyes 3×3-as kis négyzeteit, kimagaslóan nagy illetve kis értékeket kapunk a vízszintes fekete, illetve fehér élekre. Így tehát azt reméljük, hogy ezen súlymátrixok megtanulásával a leglényegesebb feature-jeit fogjuk meg a képekből álló adathalmaznak.



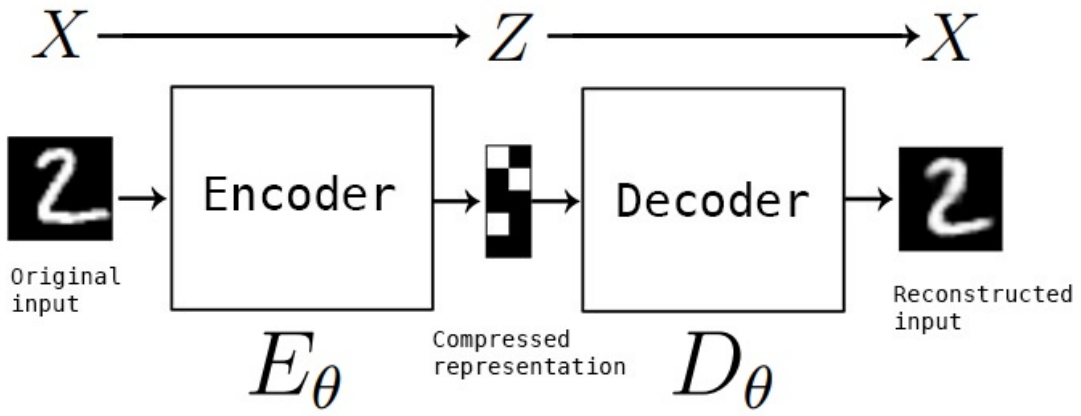
A további rétegek pedig ezen súlyozással vett értékeknek keresi meg a feature-jeit, így létrehozva egy hierarchikus rendszert.

Általában több súlymátrixot is tárolunk (ezzel megtöbbszörözve a következő réteg neuronszámát), és konvolúciós réteg után gyakran használunk úgy nevezett *maxpooling* réteget, amely egyszerűen minden kis $l \times l$-es négyzethez rendel a következő rétegben egy neuront, amelynek inputja a négyzetben lévő outputok maximuma. Az emögötti intuíció, hogy a feature-ök megjelenése lokálisan kicsit eltérhet, például, ha egy írott számjegyet 1 pixellel feljebb tolunk, attól még az új ábra ugyanazt a számjegyet ábrázolja. Így általában konvolúciós rétegek és maxpooling rétegek egymásutánjával építünk fel egy konvolúciós hálót.



1.1.5. Autoenkóderek

Az autóenkóderek olyan neurális hálók, amelyeknek az első fele egy *enkóder* részből, a második fele pedig egy *dekóder* részből áll. A hálózat input tere és output tere azonosak. Vagyis egy autoenkóder függvényként úgy írható le, mint egy $f_\theta : X \mapsto X$ függvény, valamint $E_\theta : X \mapsto Z$ az enkóder függvénye, $D_\theta : Z \mapsto X$ a dekóder rész függvénye, és $f_\theta = D_\theta \circ E_\theta$. ($\theta$ a kalibrálandó változók, tehát a súlyok és bias-ok, állapota a teljes Θ állapottérből.)



Általában $X = \mathbb{R}^n$ és $Z = \mathbb{R}^k$, ahol $k << n$. Továbbá a tanítási veszteségfüggvényünk, melyet a tanítás során minimalizálni kívánunk, hogy az egyes $i \in I$ inputokra „hasonlítson" a háló $f_\theta(i)$ outputja. Például hasonlósági mértékként az ℓ_2 normával dolgozva $\sum_{i \in I} \|i - f_\theta(i)\|^2$ kifejezést kívánjuk minimalizálni. Gyakran előfordul, hogy az előbbi összeg súlyozott változatát érdemes felírnunk, és esetleg ezeket a súlyokat dinamikusan változtatnunk, ha például szeretnénk a háló számára nehezen rekonstruálható képeket is jobban rekonstruálni.

Így tehát az autoenkóderekre úgy is gondolhatunk, mint az $I \subset X$ inputok Z kis dimenziós térben való veszteséges tömörítésére. A tanítással pedig szeretnénk minél jobb, azaz kisebb átlagos veszteségű, tömörítést előállítani.

Legyen $I' \subset X$ az összes lehetséges input (tehát például az összes 28×28-as fekete-fehér kép amely egy számjegyet ábrázol). Ekkor azt reméljük, hogy az autoenkóder az $I \subset I'$ inputok alapján megtanult enkódere egy „szép" beágyazást nyújt az I' összes lehetséges input halmazának a Z térbe. Ezért Z térre szokás *látens térként* hivatkozni,

$E_\theta(I)$ pontjaira pedig *látens pontokként.*

1.2. Noise as target

Láttuk, hogy egy autoenkóder $f_\theta : X \mapsto X$ függvényként írható le, ahol $E_\theta : X \mapsto Z$ az enkóder függvénye, $D_\theta : Z \mapsto X$ a dekóder rész függvénye, és $f_\theta = D_\theta \circ E_\theta$.

A noise as target módszernél, melyet az [1]-es cikk vezetett be, Z-t az $\mathbb{R}^n$-be standard módon beágyazott $n-1$ dimenziós gömbnek vesszük. Generálunk továbbá n darab target pontot ezen a gömbön, ahol n az inputok száma.

Jelölje az inputok halmazát I, és legyen az inputoknak megfelelő látens pontok halmaza $L = E_\theta(I)$. Továbbá jelölje a target pontok halmazát T. Ekkor szeretnénk a hálót úgy tanítani, hogy:

1. Az egyes $i \in I$ inputokra „hasonlítsanak" az egyes $f_\theta(i)$ outputok.

2. Az L látens pontok legyenek minél közelebb az T pontokhoz.

Az első veszteségfüggvény tehát a szokásos autoenkóder-hibafüggvény minimalizálását jelenti. A második veszteségfüggvény pedig az, hogy a $G = (L, T, E)$ súlyozott teljes páros gráfban, ahol egy $e = lt, l \in L, t \in T$ él súlya $c(e) = \|l - t\|$, határozzuk meg egy M minimális súlyú maximális párosítást, majd pedig veszteségfüggvényként minimalizáljuk ezen M párosítás súlyát. Tanítási veszteségfüggvényként pedig ezen két veszteségfüggvény valamely az adott esetben megfelelő lineáris kombinációját vesszük.

A második veszteségfüggvényben az M párosítás meghatározásához sajnos a magyar módszer alkalmazása rendkívül lassú lenne a gyakorlatban, ezért (általában 50-200-as méretű) batch-ekre bontjuk mind az L-beli, mind a T-beli pontokat. Ezután pedig az egyes batch-eken minimalizáljuk a második veszteségfüggvényt a magyar módszer alkalmazásával.

A módszer sikerességének információelméleti indoklása megtalálható a [2]-es hivatkozás alatt.

1.2.1. Megjegyzés. Autoenkóderek betanításánál azt tapasztaljuk, hogy az input halmazt az enkóder néhány kis kiterjedésű helyre képezi le, a látens tér nagy része pedig teljesen kihasználatlan marad. Így ha ezen kis területekről, ahol a látens pontok sűrűsége nagy, veszünk egy tetszőleges pontot, akkor a dekóder általában egy szép outputot

generál. Azonban kis sűrűségű helyeken ennek az ellenkezője igaz. Célunk, hogy az autoenkóder dekódere úgynevezett generatív modellként működjön. Informálisan ez alatt azt értjük, hogy a látens tér véletlenszerűen választott pontjától azt várjuk, hogy a dekódert alkalmazva rá az I halmaz egy pontjának közelébe jutunk. Valamelyest formálisabban, a dekódert akkor nevezzük generatív modellnek, ha a látens térbeli egyszerű prior eloszlású valószínűségi változóra a dekódert alkalmazva (közelítőleg) megkapjuk azt a valószínűségeloszlást, amiből az adathalmazunk vétetett. (Esetünkben a prior eloszlás a látens térbeli egységgömbön vett egyenletes eloszlás.) Ennek érdekében szeretnénk egy kompakt felületen „egyenletesen" széthúzni a látens pontfelhőt, remélve, hogy így a kompakt felület minden pontjára szép outputot generál a betanult dekóder. Ez a NAT módszer heurisztikája.

1.3. Problémafelvetés

Amint azt az előző szekcióban láttuk, a NAT háló tanítása során szükségünk van egy páros gráfban minimális súlyú teljes párosítás keresésére. Habár elvben ez megoldható lenne a magyar módszer alkalmazásával, a probléma, hogy a magyar módszer (gyorsított változatának) futásideje $O(n^3)$, ahol n a tanítóadatok száma. A gyakorlatban azonban mivel n legalább tízezres, de gyakran még nagyobb nagyságrendet vesz fel (máskülönben kevés adat lévén a háló nem lenne képes megfelelő szintű általánosításra), így a magyar módszer futtatása rendkívül időigényes folyamat lenne, ezért a NAT módszer megalkotói az egyes tanítási batch-eken (melyek mérete csupán 50-200) futtatták a magyar módszert.

Habár így az algoritmus valóban kellően gyors, az így talált teljes párosítás súlya igencsak messze lehet a minimálistól. Ezt a problémát szeretnénk kiküszöbölni azzal, hogy kidolgozunk egy olyan párosítási algoritmust, amely szintén reális futásidőben, de a minimális súlyú teljes párosítás súlyát jobban megközelítő (közel-) teljes párosítást talál. Ehhez szeretnénk elrugaszkodni a batch-enkénti párosításkereséstől, és globális módszert kialakítani.

További követelményünk, hogy az algoritmus *online* legyen, azaz képes legyen az adatok egyesével való bevételével a párosítást folyamatosan bővíteni és módosítani. Szeretnénk továbbá, hogy a párosítás online felépítése során minél kevesebbet változzon a párosítás, ezzel elérve, hogy az epoch-onkénti tanítás közben a látens pontokra ható vonzerők ne változzanak meg gyakran.

1.4. A probléma megoldásának magas szintű áttekintése

Vegyük azt a $G = (C, S; E)$ súlyozott teljes páros gráfot, amelynek C komponense a látens pontok aktuális helyei, S komponens pedig a target pontok (rögzített) helyei, és az élsúlyok az egyes ℓ_2 távolságok. Így tehát ezen páros gráfban szeretnénk (közel-) minimális súlyú (közel-) teljes párosítást találni, ráadásul online módon.

Először is az élek számát drasztikusan csökkentjük azzal, hogy a teljes gráf helyett ennek csak azon $G' = (C, S; E')$ részgráfjával dolgozunk, amelyben minden látens ponthoz tartozó C-beli csúcs csak a legközelebbi $\approx \log(n)$ darab S-beli csúccsal van összekötve. Mivel a közeli szomszédok közt fut él, azt reméljük, hogy ebben a részgráfban létezik közel-teljes párosítás, és ezek közt közel-minimális súlyú. Az *ANNOY* algoritmussal [15] ezen ritka gráfot nagyon gyorsan felépíthetjük, és élsúlyozását kiszámíthatjuk. Ennél a módszernél tehát kihasználtuk, hogy a gráf csúcsainak megfeleltethetőek egy metrikus tér elemei, ahol az élsúly az él végpontjainak egymástól vett távolsága.

Ezek után bemutatunk egy online párosító algoritmust, amely súlyozatlan páros gráfban keres maximális párosítást. Megmutatjuk, hogy ha ezt az algoritmust a G' ritka gráfunkon alkalmazzuk, akkor ezen párosítás megkeresése csupán $O(n^{\frac{3}{2}} \log(n))$ időben történik.

Az előbbi algoritmus azonban az általunk vizsgált $n = 50000$-es gyakorlati példára még mindig túl lassú, így mutatunk egy trükköt, melynek segítségével az algoritmus rendkívül gyors lesz, azon az áron, hogy a maximális párosítás mérete ezzel csökken. Azonban megmutatjuk, hogy a gyakorlatban nem lényeges mértékű ez a csökkenés.

A felhasznált online párosítási algoritmus az Even-Shiloach algoritmus irányított gráfokra való általánosításán alapszik. Így bemutatjuk az Even-Shiloach algoritmust és ismert általánosításait, valamint a dolgozat eredményeként mutatunk egy módosítást az irányított Even-Shiloach algoritmusra, ami segítségével közel-minimális súlyú párosítást keresünk az előbbi párosítási algoritmusba beépítve.

2. fejezet

Gráfelméleti előkészületek

Ebben a fejezetben tárgyaljuk az *Even-Shiloach* algoritmust [10], és annak általánosításait [9], [11]. Az Even-Shiloach algoritmus általánosításai közt egy súlyozott élekre való általánosítást is bemutatunk, amely a dolgozat eredménye.

2.1. Even-Shiloach algoritmus és általánosításai

2.1.1. Tétel. *(Even-Shiloach, 1981) [10] Legyen $G = (V, E)$ egy (súlyozatlan) egyszerű gráf, jelölje $m = |E|$ az élek számát, és $n = |V|$ a csúcsok számát. Továbbá jelölje $\forall v \in V : rang(v) = d(s, v)$, azaz a v pont távolságát az s csúcstól, és legyen rögzítve az E élek egy tetszőleges sorrendje. Ekkor $O(nm)$ időben megoldható, hogy a G gráf éleit egyesével elhagyjuk a meghatározott sorrendben, és a folyamat közben minden $v \in V$ csúcsra frissítjük a $rang(v)$ értékeket.*

Így tehát bármely kapott köztes részgráfra $O(1)$ időben (ismételten) lekérdezhetjük bármelyik csúcs rangját.

Bizonyítás. Legyen H a dinamikus gráfunk amely kezdetben $H = G$, majd pedig G éleit elhagyjuk belőle a rögzített sorrendben.

Először is határozzuk meg minden csúcs rangját $H = G$-ben egy szélességi bejárással. Ennek futásideje $O(m)$. (Az s-ből nem elérhető pontok rangja legyen n.)

A dinamikus algoritmus a tételnek megfelelően minden csúcsnak fenntarjuk a rangját, valamint minden csúcsnak fenntartjuk a szüleinek, gyerekeinek, és a testvéreinek a halmazát. Azaz $szülők(v) = \{u \mid uv \in E \wedge rang(v) = rang(u) + 1\}$, $testvérek(v) = \{u \mid uv \in E \wedge rang(v) = rang(u) \wedge u \neq v\}$, és $gyerekek(v) = \{u \mid uv \in E \wedge rang(v) = rang(u) - 1\}$.

Vegyük észre, hogy $\forall v \in V$-re ezen három halmaz particionálja v szomszédainak halmazát.
Valóban: A halmazok nyilvánvalóan diszjunktak az elemeik rangjából adódóan. Definíció szerint a v szomszédainak halmaza tartalmazza mindhárom halmazt, így az uniójukat is. Továbbá bármely két $v_1v_2 \in E$ élre $rang(v_2) > rang(v_1) + 1$ nem lehetséges, mivel v_1v_2 él bizonyítékul szolgál arra, hogy $rang(v_2) \leq rang(v_1) + 1$. Azaz a három halmaz fedi v szomszédainak halmazát.

Könnyen látható, hogy kezdetben az élek halmazán iterálva $O(m)$ időben felépíthetjük ezt a három halmazt az élek csúcsainak rangjának lekérdezésével.

Nézzük meg, hogy hogyan frissíthetjük a H gráf tárolt struktúráját, ha H-ból elhagyunk egy élt, azaz legyen $H' = H \setminus uv$ ahol $uv \in E(H)$. Mivel H irányítatlan, feltehetjük, hogy $rang_H(v) \geq rang_H(u)$. Tehát az előbbiek alapján vagy $rang_H(v) = rang_H(u)$, vagy $rang_H(v) = rang_H(u) + 1$.

Vegyük észre, hogy éltörléssel semelyik pont rangja sem csökkenhet, azaz $\forall v_2 \in V$: $rang_H(v_2) \leq rang_{H'}(v_2)$. Továbbá, ha $v_1v_2 \in E(H')$, akkor $rang_{H'}(v_2) \leq rang_{H'}(v_1) + 1$. Tehát, ha $v_1v_2 \in E(H')$, $rang_H(v_2) = rang_H(v_1) + 1$, és $rang_{H'}(v_1) = rang_H(v_1)$, akkor $rang_{H'}(v_2) = rang_H'(v_1) + 1$. Definíció szerint $rang_H(s) = rang_{H'}(s) = 0$, azaz így s minden H'-beli gyerekének a rangja ugyanaz H'-ben és H-ban, és s helyett most ugyanaz elmondható s minden egyes H'-beli gyerekére, és így tovább.

1.) eset: Ha $rang_H(v) = rang_H(u)$. Ekkor az előbbi megállapítás alapján láthatjuk, hogy mivel minden csúcsnak ugyanazok a gyerekei H-ban mint H'-ben (tekintve, hogy testvéreket összekötő élt töröltünk), ezért egyik csúcs rangja sem változik.

2.) eset: Ha $rang_H(v) = rang_H(u) + 1$. Ekkor biztosan ugyanaz marad a rangja minden olyan csúcsnak, amely nincs v leszármazottai között. (v leszármazottai közé értjük önmagát is.) Sőt, v rangja pontosan akkor nő, ha $szülők(v) = \emptyset$. 2.1) aleset: Ha $rang_H(v) = rang_H(u) + 1$ és $szülők(v) \neq \emptyset$. Ekkor $rang_{H'}(v) = rang_H(v)$, és így értelemszerűen a többi csúcs rangja is ugyanaz marad.

2.2) aleset: Ha $rang_H(v) = rang_H(u) + 1$ és $szülők(v) = \emptyset$. Legyen $l = rang_H(v)$. Ekkor tudjuk, hogy v rangja legalább 1-gyel nőtt, így megjelöljük, mint gyanús csúcsot, és a rangjához tartozó értéket növeljük 1-gyel, azaz legyen $rang_{H'}(v) = l + 1$, továbbá így v testvéreiből szülők lesznek, és gyerekeiből testvérek, azaz legyen $szülők(v) = testvérek(v)$, $testvérek(v) = gyerekek(v)$ és $gyerekek(v) = \emptyset$. Fontos megjegyezni, hogy az utóbbi műveletet valójában úgy érdemes végrehajtani, hogy a szülők, testvérek és gyerekek pointereit permutáljuk, így ez $O(n)$ idő helyett $O(1)$ időben végrehajtható. Ennek megfelelően

v korábbi gyerekei (azaz mostani testvérei) szülőhalmazából áttesszük v-t a testvérhalmazába, és v korábbi testvérei (azaz mostani szülei) testvérhalmazából áttesszük v-t a gyerekhalmazba. Megjelöljük gyanúsnak v korábbi gyerekei (most testvérei) közül azokat amelyeknek nincs szülőjük (mivel a szülővel rendelkezőknek nem csökken tovább a rangja, de a szőlővel nem rendelkezőknek igen). Valamint, ha v-nek már van szülője, akkor töröljük a gyanússágát, megtaláltuk a rangját. Így minden gyanús csúcs az $l+1$-edik szinten van jelenleg, és csak a gyanús csúcsoknak változik a rangja a továbbiakban. Így haladunk tovább a gyanús csúcsok szintjét eggyel csökkentve, a halmazokat módosítva, és korábbi gyerekeik és ő maguk gyanússágát leellenőrizve. Így valóban a gyanús csúcsok mindig egy szinten lesznek, és a folyamat során ez a szint egyesével nő, amíg el nem jutunk az n-edik szinthez amikor megállhatunk, az ott lévő gyanús csúcsok s-ből nem elérhető pontok a gráfban.

Az világos, hogy ez egy véges algoritmus, de felmerül a kérdés, hogy a gyanús csúcsok rekurzív megtisztítása nem vesz-e túl sok időt igénybe. Jelöljük egy $uv \in E$ él rangját $rang(uv) = rang(u) + rang(v)$-vel. Ekkor könnyen látható, hogy bármely $uv \in E$ élre $rang(uv) \geq 0 + 1 = 1$ és $rang(uv) \leq n + n = 2n$. Azaz $m \leq \sum_{e \in E} rang(e) \leq 2nm$. Így mivel minden v csúcs szintnövelésével $|gyerekek(v)|$-vel nőtt $\sum_{e \in E} rang(e)$ értéke, és a szintnöveléssel (és gyerekei ellenőrzésével) mindig legfeljebb $c\,|gyerekek(v)|$ idő telt el valamely $c > 0$ (v-től nem függő) konstansra, ezért az algoritmus valóban $O(m + m + 2nm - m) = O(nm)$ időben fut. $\square$

2.1.2. Megjegyzés. Az előbbi bizonyítást végigkövetve könnyen látható, hogy ha csak a legfeljebb Δ rangú pontok rangját szeretnénk tudni lekérdezni (és a többire mondjuk kapjunk vissza $\Delta + 1$-et), akkor csak annyit kell módosítanunk az algoritmusunkon, hogy a gyanús pontokat nem az n-edik szintig, hanem a $\Delta+1$-edik szintig követjük. Valóban így mindvégig $m \leq \sum_{e \in E} rang(e) \leq 2(\Delta + 1)m$, azaz ugyanúgy mint előbb belátható, hogy az algoritmus futásideje $O(m+m+2(\Delta+1)m-m) = O(\Delta m)$. Továbbá láthatjuk, hogy habár az algoritmus alapból nem engedi meg élek beszúrását, olyan élet mégis beszúrhatunk amely nem csökkenti egyik csúcs aktuális rangját sem, azaz olyan $uv \notin E$ élet amelyre $rang_H(v) \leq rang_H(u) + 1$, és a beszúrással $H(E) = H(E) \cup uv$, és az u illetve v szomszédsági halmazainak esetleges módosításával $O(1)$ időben elvégeztük az él beszúrását. Így megkaptuk a következő tételt.

2.1.3. Tétel. *Legyen $G = (V, E)$ egy dinamikus egyszerű gráf, $s \in V$ kitüntetett csúcs. Ekkor az előbbi módosítással egy olyan algoritmust kapunk amelyben egy lépésben vagy töröljük G dinamikus gráf egy élét, vagy pedig beszúrunk egy élet amely az aktuális rangok egyikét sem változtatja meg, és egy tetszőleges csúcs rangját bármely lépésben után $O(1)$ időben lekérdezhetjük. Ennek az algoritmusnak a futásideje az előbbiek szerint $O(\Delta(m+i))$ ahol $m = |E|$ és i a beszúrt élek száma.*

2.1.4. Megjegyzés. Az Even-Shiloach algoritmust, pontosan annak egy általánosítását, arra fogjuk használni, hogy a folyamat során $v \in V$ pontokra megkapjunk egy sv legrövidebb utat, ha $rang(v) \leq \Delta$. Ez valóban megoldható $O(rang(v)) = O(\Delta)$ időben, mivel az utat felépíthetjük visszafelé haladva v-től mindig egy tetszőleges gyerekét választva a legutóbb bevett csúcsnak. Ezen út hossza $rang(v)$ lesz, ami így definíció szerint minimális hosszú sv út.

Mi azonban irányított gráfokkal fogunk dolgozni, és nem triviális, hogy az előbbi algoritmus hogyan módosítható az irányított esetben. Ekkor egy uv irányított él esetén csak azt tudjuk, hogy $rang(v) \leq rank(u) + 1$. Látni fogjuk azonban, hogy az az általánosítás működik, hogy minden csúcs szomszédait rangonként halmazokra bontjuk, és ezeket számontartjuk a folyamat során.

2.1.5. Tétel. *(irányított Even-Shiloach) [9], [11] Legyen $D = (V, A)$ egy (súlyozatlan) irányított gráf, jelölje $m = |A|$ az irányított élek számát, és $n = |V|$ a csúcsok számát. Továbbá jelölje $\forall v \in V : rang(v) = d(s, v)$, azaz a v pont távolságát az s csúcstól, és legyen rögzítve az A irányított élek egy tetszőleges sorrendje. Ekkor $O(nm)$ időben megoldható, hogy a D gráf éleit egyesével elhagyjuk a meghatározott sorrendben, és a folyamat közben minden $v \in V$ csúcsra frissítjük a $rang(v)$ értékeket.*

Bizonyítás. Hasonló bizonyítást adunk, mint az irányítatlan esetben. Legyen H a dinamikus irányított gráfunk, amely kezdetben $H = D$, majd pedig később D éleit az adott sorrendben elhagyjuk belőle. Végezzünk továbbá egy szélességi bejárást $H = D$ irányított gráfon, amellyel $O(m)$ időben meghatározzuk az aktuális rangokat. (Az s-ből irányított úton nem elérhető pontok rangja legyen n.)

Mint azt már említettük, az irányítatlan esettel ellentétben most egy $v \in V$ csúcsnak a szüleinek (ez alatt most azon $u \in V$ csúcsok halmazát értjük amelyekre $uv \in A$) a rangja $\{0, 1, 2, \ldots, n-1\}$ halmaz bármely eleme lehet. Éppen ezért most a folyamat során végig számontartunk minden egyes ponthoz n darab halmazt melyek az adott csúcs gyerekeit

particionálják rang szerint. Legyen tehát $\textit{szülők}(v,i) = \{u \in V \mid uv \in A \wedge \textit{rang}(u) = i\}$ minden $v \in V$ és $i \in \{0,1,2,\ldots,n-1\}$-re. Továbbá hasonlóan particionáljuk minden pont gyerekeit is, ezzel kapva $\textit{gyerekek}(u,i) = \{v \in V \mid uv \in A \wedge \textit{rang}(v) = i\}$ halmazokat minden $u \in V$ és $i \in \{0,1,2,\ldots,n-1\}$-re. Könnyen látható, hogy ezen halmazokat $H = D$ gráfra O(m) időben inicializálhatjuk A élein iterálva a végpontok rangjait lekérdezve.

Az irányítatlan esethez hasonlóan most is igaz, hogy minden csúcs rangja a folyamat során monoton nő. Ebből ugyanúgy láthatjuk, hogy $uv \in A$ élelhagyással (azaz legyen $H' = H \setminus uv$), ha egy csúcs rangja nem változik, akkor annak az 1-gyel nagyobb rangú gyerekeinek a rangja sem változik, és így induktíve semelyik olyan leszármazottjának sem változik a rangja amelyet olyan irányított úton érünk el a csúcsból, hogy az irányított út minden következő csúcsának 1-gyel nagyobb a rangja mint az övének. Hívjuk az ilyen irányított utakat rangnövekvő utaknak.

Tehát, ha uv élet töröljük, akkor mivel a definícióból adódóan a legrövidebb sw irányított út egyben rangnövekvő út is, és mivel $\textit{rang}_H(s) = \textit{rang}_{H'}(s) = 0$, ezért minden olyan $w \in V$-re amelyre $\exists sw$ rangnövekvő út amely nem tartalmazza uv irányított élet, a w csúcs rangja sem változik. Legyen tehát $l = \textit{rang}_H(v)$. Ekkor az előbbi alapján tudjuk, hogy az uv éltörléssel $\forall w \in W : \textit{rang}(w) \leq l-1$ csúcs rangja nem változik. v rangja pontosan akkor változik (azaz nő), ha $\textit{szülők}(v, l-1) = \emptyset$.

Tehát megint hasonló az eljárás. v pontot gyanúsnak jelöljük, ha $\textit{szülők}(v, \textit{rang}(v) - 1) = \textit{szülők}(v, l-1) = \emptyset$. Tehát pontosan akkor jelöltük gyanúsnak, ha a rangja megváltozik uv éltörléssel. Ezután növeljük $\textit{rang}(v)$ tárolt értékér 1-gyel, majd ennek megfelelően v szüleinek gyerekhalmazaiban v-t rakjuk át az 1-gyel nagyobb ranghoz tartozó gyerekhalmazba, és v gyerekeinek szülőhalmazaiban rakjuk át v-t az 1-gyel nagyobb rangoz tartozó szülőhalmazba. Ezzel tehát $\sum_{uv \in A} \textit{rang}(u) + \textit{rang}(v)$ (a folyamat során monoton növő változó) értékét $|\textit{szomszédok}(v)|$-vel növeltük, és a halmazok módosítása legfeljebb $c\,|\textit{szomszédok}(v)|$ időt vett igénybe valamely $c > 0$, a folyamat során nem változó értékre.

Ezek után megnézzük, hogy a gyanús csúcsok, esetünkben v, a módosítások után is gyanús-e, azaz tovább nő-e a rangja. Vagyis akkor lesz továbbra is gyanús, ha $\textit{szülők}(v, \textit{rang}(v) - 1) = \textit{szülők}(v, l) = \emptyset$. Mivel pontosan azon w csúcsoknak csökken a szintje amelyekre $\textit{szülők}(w, \textit{rang}(w) - 1) = \emptyset$, ezért mivel a módosítás csak az $l-1$-ről l-re növelte az előzőleg gyanús csúcsok, az ilyen w csúcsok az előzőleg gyanús csúcsok l rangú szomszédai közül kerülnek ki, így csak ezekről kell meghatároznunk, hogy gyanúsak-e, és a gyanúsak rangját 1-gyel növeljük. Egy $w \in V$ csúcs rangjának 1-gyel való növelése így

$|szomszédok(w)|$-vel növelte $\sum_{uv\in A} rang(u) + rang(v)$ értékét, és legfeljebb $c\,|szomszédok(v)|$ időt vett igénybe. Továbbá ezzel arányos időt vesz igénybe w gyanús csúcs a módosítás utáni $rang(w)$ rangú gyerekeinek gyanúságának vizsgálata is.

Így tovább szintenként lejjebb kerülnek a gyanús csúcsok, és ha az n-edik szinthez érünk, megállunk, ekkor tudjuk, hogy ezen csúcsok nem érhetőek el s-ből irányított úton. Ezzel befejeztük az uv irányított él törlését követő korrigációkat.

Mivel $m \leq \sum_{uv\in A} rang(u) + rang(v) \leq 2nm$, ezért az egész algoritmus futásideje valóban $O(m) + O(m) + O(nm) = O(nm)$. $\square$

2.1.6. Megjegyzés. A fenti algoritmus nem pontosan a [9] és [11]-beli algoritmus, az ezen cikkekben leírt algoritmusban a minden csúcshoz csak az előző szintbeli szüleit (tehát a csúcs szintjének tanúit) tároljuk. Ezzel szemben mi minden csúcshoz az összes szinten lévő szüleit tároljuk szintekre bontva. Így az algoritmus valamennyivel gyorsabb (de a matematikai futásideje változatlan), a memóriaigényt pedig megnöveli. Mivel így sem használ sok memóriát az algoritmus, ezért ez egy praktikus döntés, azonban legfőképpen azért prezentáltuk így az irányított Even-Shiloach algoritmust, mert az élsúlyozott általánosításunknál ez egy nagyon fontos lépés.

2.1.7. Megjegyzés. Mint az irányítatlan esetben, most is módosíthatjuk az előző tételt. Egy $uv \notin A$ irányított élet beszúrhatunk a dinamikus irányított gráfba, ha ez a beszúrás éppen nem csökkenti egyik $w \in V$ csúcs rangját sem. Azaz, ha $rang(v) \leq rang(u) + 1$. Valamint, ha csak a legfeljebb Δ rangú csúcsok rangját kívánjuk számontartani (a többire pedig a rangfüggvény adjon vissza $\Delta + 1$-et), akkor az algoritmust úgy módosítva, hogy a szélességi bejárást csak Δ szintig csináljuk (a többi csúcs rangja legyen $\Delta + 1$), és éltörlés után a gyanús csúcsok követését a $\Delta + 1$-edik szinten állítjuk meg (valamint a gyerek- illetve a szülőhalmazok számát is csökkenthetjük csúcsonként Δ darabra). Így mivel $m \leq \sum_{uv\in A} rang(u) + rang(v) \leq 2(\Delta + 1)m$, ezért az algoritmus futásideje $O(m) + O(m) + O(\Delta m) = O(\Delta m)$. Így kapjuk a következő fontos tételt.

2.1.8. Tétel. *[9], [11] Legyen $D = (V, A)$ egy dinamikus irányított gráf, $s \in V$ kitüntetett csúcs. Ekkor az előbbi módosítással egy olyan algoritmust kapunk amelyben egy lépésben vagy töröljük D dinamikus gráf egy élét, vagy pedig beszúrunk egy élet amely az aktuális rangok egyikét sem változtatja meg, és egy tetszőleges csúcs rangját bármely lépésben után $O(1)$ időben lekérdezhetjük. Ennek az algoritmusnak a futásideje az előbbiek szerint $O(\Delta(m + i))$ ahol $m = |A|$ és i a beszúrt élek száma.*

A dolgozat eredménye az irányított Even-Shiloach algoritmus további általánosítása, amelyet az élsúlyozott esetben a közelítőleg minimális súlyú maximális párosítás keresésére fogunk alkalmazni.

Láttuk, hogy az Even-Shiloach és az irányított Even-Shiloach algoritmusokat tudjuk alkalmazni az s kitüntetett csúcsból induló legfeljebb Δ hosszú utak megtalálására (vagy annak igazolására, hogy ilyen rövid út nem vezet s-ből az adott csúcsba). Később, a 2.2.1-es tételben, láthatjuk, hogy az irányított Even-Shiloach algoritmus (pontosabban a 2.1.8-beli általánosítása) hogyan alkalmazható adott csúcsokból való legrövidebb javító utak megtalálására. Ebben a tételben láthatjuk, hogy mennyit gyorsít a rövid javító utak ezen hatékony számontartása és lekérdezhetősége.

Az előbbi továbbfejlesztett változata, amikor súlyozott páros gráfban szeretnénk minél kisebb súlyú maximális párosítást találni. Ehhez ugyanúgy legrövidebb javító utakat fogunk keresni, azonban azok közül mindig azt választjuk, amelyikkel való augmentálása a párosításnak minimalizálja az új párosítás össz-élhosszát. Később bemutatjuk, hogy ez az általánosítása az irányított Even-Shiloach algoritmusnak gyors a gyakorlatban, az irányított Even-Shiloach algoritmusnál sem lényegesen lassabb, azonban a futásidejére adott elméleti becslésünk lényegesen gyengébb, nem tükrözi jól az algoritmus gyakorlatban mért futásidejét.

2.1.9. Megjegyzés. A következő algoritmus megértéséhez mind a 2.1.8-es tétel algoritmusának, mind pedig a 2.2.1-es tétel algoritmusának és bizonyításának ismerete szükséges.

2.1.10. Tétel. *Legyen $D = (V, A)$ egy dinamikus súlyozott irányított gráf, $s \in V$ kitüntetett csúcs. Ekkor létezik olyan algoritmus, amelyben egy lépésben vagy töröljük D dinamikus gráf egy élét, vagy pedig beszúrunk egy uv iránytott élet, amelyre $rang(u) \geq rang(v)$ az aktuális rangokkal, valamint számontartja a legfeljebb Δ rangú csúcsok rangját (a többire pedig $\Delta + 1$-et ad vissza). Továbbá egy $\leq \Delta$ rangú v csúcshoz $O(\Delta)$ időben megtalálhatjuk a legkisebb élsúlyú legrövidebb $s \to v$ utat. Ennek az algoritmusnak a futásideje $O(\Delta(m+i)n\log(n))$ ahol $m = |A|$, i a beszúrt élek száma, és $n = |V|$.*

2.1.11. Megjegyzés. Ezt az algoritmust úgy fogjuk közel-minimális súlyú párosítás keresésére alkalmazni a 2.2.1-es tétel algoritmusával, hogy egy $M \subseteq D$ (dinamikus), kezdetben $M = \emptyset$, párosítást számontartunk és növelünk az algoritmus szerint, és ez alapján határozzuk meg D élsúlyozását. Származzon kezdetben (tehát $M = \emptyset$-re) D élsúlyozása az irányítatlan súlyozásból. Mivel a 2.2.1-es tétel algoritmusa közben M növelése mindig

egy új él beszúrásával történik (mivel amikor egy javító út mentél augmentálunk, akkor az egy él megfordítása az él törléséből és az ellenkező irányú él beszúrásából áll), ezért megtehetjük, hogy az újonnan beszúrt M-beli éleket az irányítatlan élsúly negáltjával súlyozva rakjuk be D-be. Azaz az algoritmus során végig D-ben pontosan M élei vannak a hozzá tartozó irányítatlan él súlyának (-1)-szeresével, a többi él pedig az irányítatlan él súlyának 1-szeresével szerepel. Így tehát amikor a 2.2.1-es tétel algoritmusával találunk egy javító utat D-ben, akkor az út D-beli súlya pontosan annyi, amennyivel M párosítás (az irányítatlan súlyozásnak megfelelő) súlya nő az úttal való augmentálás következtében. Tehát az ötletünk, hogy az Even-Shiloach algoritmusnak ezen általánosításával mindig a 2.2.1-es tétel algoritmusa szerint adott $c \in C$ pontra nem csak ezen pontból induló legrövidebb javító utak egyikét vesszük, hanem ezek közül a legkisebb D-beli súlyút, majd e mentén augmentálunk. Ezen út gyors megkeresését éppen az algoritmusunk teszi lehetővé. Ily módon szeretnénk minél kisebb súlyú maximális párosítást találni.

Bizonyítás. Vegyük tehát D élsúlyait a tételben említett módon, a súlyfüggvényt jelölje w. Az irányított Even-Shiloach algoritmust fejlesztjük tovább a probléma megoldásához.

Korábban a *szülők*(v, l) halmazoknál nem volt lényeges az elemek sorrendje, azonban most bevezetünk az egyes csúcsokra egy távolságfüggvényt, és *szülők*(v, l) halmazok most az $(u, \textit{távolság}(u) + w(uv))$ párok rendezett halmazai lesznek, ahol az u csúcsok a v csúcs l rangú szülei, és a rendezés második tag alapján zajlik (kisebbtől a nagyobbig).

Legyen tehát $\textit{távolság}(v) = \min\{w(P) \mid P = s \to v \text{ minimális hosszú irányított út}\}$. Ekkor világos, hogy $\forall v \in V(D), v \neq s$-re:

$$\textit{távolság}(v) = \min\{\textit{távolság}(u) + w(uv) \mid uv \in A, \textit{rang}(u) = \textit{rang}(v) - 1\} \tag{2.1}$$

azaz *távolság*(v) a *szülők*$(v, \textit{rang}(v) - 1)$ rendezett halmaz első elemének második tagja lesz $\forall v \in V(D), v \neq s$ csúcsra (és *távolság*$(s) = 0$), így tehát a távolságfüggvényt nem kell külön számolni és számontartani. Az algoritmus során végig fenntartjuk ezen dinamikus távolságfüggvény értékét minden csúcsra.

Kezdetben az irányított Even-Shiloach algoritmusban szélességi bejárással határozzuk meg a *rang*, a *szülők*, és a *távolság* függvényeket. Most ezen annyit módosítunk, hogy egy szülő rendezett halmaz bővítésekor (az aktuális rendezett halmaz méretének logaritmusával arányos időben) a megfelelő helyre szúrjuk be az új elemet. Illetve a szélességi bejárást w szerint végezzük, azaz élválasztásnál mindig a legkisebbet választjuk, így a 2.1-es egyenlet alapján a szélességi bejárással a *távolság* függvényt is megkapjuk. Ehhez

persze kell az élsúlyok rendezése, ami $O(\log(m))$ időt vesz igénybe, így a szélességi bejárás ezen futtatása $O(m)\log(m)$ időt vesz igénybe. (Azonban erre a 2.2.1-es tétel algoritmusában nem is lesz szükségünk, mivel ott kezdetben minden s-től különböző csúcs rangja 1, és mind a $2n$ él súlya 0, azaz *távolság* $\equiv 0$, és a szülő halmazok is ismertek.)

Élbeszúrás esetén, ha uv a beszúrt él, akkor csak *szülők*$(v, rang(u))$ rendezett halmazba kell beszúrni egy elemet, ugyanis a *távolság* függvény 2.1-beli (*rang* szerinti) rekurzív alakjából adódóan, mivel a beszúrási feltétel szerint $rang(u) \geq rang(v)$, ezért a az élbeszúrás a *rang* függvényt nem változtatja meg, továbbá $rang(u) \neq rang(v) - 1$, azaz a *távolság* függvény sem változik az élbeszúrással.

Élelhagyás esetén viszont nehezebb dolgunk van. A *rang* függvény is ugyanígy változik, mint az irányított Even-Shiloach algoritmusban. Azonban a szülő halmazok változása azonban lényegesen bonyolultabb lehet. Az $uv \in A$ irányított él elhagyásával először törölnünk kell a v-hez tartozó számpárt a *szülők(v, rang(u)* halmazból. Ha $rang(u) \neq rang(v) - 1$, akkor ezzel meg is vagyunk, mivel ekkor a *távolságfüggvény* nem változik. Ha azonban $rang(u) \neq rang(v) - 1$ és u (az elhagyás előtt) éppen a *szülők*$(v, rang(v) - 1)$ első elemének első tagja, akkor *távolság*(v) megváltozhat az előbbi szülő halmaz módosítás következményében, és ezzel v-nek a $rang(v) + 1$ szinten lévő w gyerekeire megváltozhat a *szülők*$(w, rang(w) - 1)$ halmaz, pontosan akkor, ha *szülők*$(w, rang(w) - 1)$ első elemének első tagja éppen v. Így tehát beindul egy, a szinteken lefelé menő, láncreakció, hasonlóan mint a *rang* függvény módosításakor. Így ezt a láncreakciót párhuzamosan kezeljük a rangok növelésével, szintenként végezve őket. Vegyük észre, hogy definícióból adódóan minden szinten a rangnövelésre váró csúcsok részhalmazát alkotják azon pontok halmazának melyekre a *távolság* függvény módosul. Továbbá amíg egy csúcs rangja nem rögzült (a csúcs zuhanóban van), akkor nem tudjuk, hogy mi a rangja, így a szülőhalmazokból a zuhanás elején töröljük, és a zuhanás végén visszaírjuk a frissített ranggal és távolsággal.

Sajnos az világos, hogy így a rangmódosítások összesen $O(\Delta(m+i)\log(n))$ időt veszznek igénybe (ugyanúgy bizonyítva mint az irányított Even-Shiloach algoritmus esetén, csak még a rendezésből adódó $\log(n)$-es szorzót be kell vennünk), azonban a dolgozat keretében nem sikerült távolságok változtatásából eredendő módosítások futásidejére jó becslést adnunk, így maradunk a triviális becslésnél, miszerint minden távolságmódosítás mind az n csúcsra kihat, ezzel kapva az $O(n\Delta(m+i)\log(n))$-es nagyon durva felső becslést. Így tehát az össz-futásidőre valóban $O(n\Delta(m+i)\log(n))$ felső becslést kapunk, azonban később látjuk, hogy gyakorlatban az algoritmus közel olyan gyors, mint az irányított Even-Shiloach algoritmus, ezért azt sejtjük, hogy létezik sokkal jobb elméleti felső

becslés is. □

2.1.12. Megjegyzés. Eltárolhatjuk a *távolság* függvény logikája kapcsán azt a D-beli s gyökerű feszítőfenyőt, amelyben minden v szülője a $szülők(v, rang(v)-1)$ rendezett halmaz első elemének első tagja. Ezt a feszítőfenyőt a szülő halmazok módosításával párhuzamosan módosítjuk amikor szükséges. Ebből láthatjuk, hogy egy élelhagyáskor az elhagyott éllel a fenyőből leszakadt csúcsoknak a távolságával lehet csak gond. Ha tehát m élből 1-et hagynánk el véletlenszerűen, akkor az csak $\frac{n-1}{m}$ eséllyel lenne fenyőél, és még ha fenyőél is, akkor is egy fenyőél elhagyásával átlagosan

$$\approx \frac{\frac{n-1}{\Delta}\sum_{i=1}^{\Delta} i}{n-1} = O(\Delta) \tag{2.2}$$

fenyőélet hagynánk el (mivel a feszítőfenyőt a leveleiből építve láthatjuk, hogy ez az érték akkor lesz maximális ha a feszítőfenyő s-ből kiinduló diszjunkt irányított utak uniójából áll, amelyek közül egy kivételével mindegyik Δ hosszú). Vagyis véletlen éltörlések mellett a bármely gráf esetén $O(\Delta^2(m+i)\log(n))$ a várt futásidő. Ami már jóval közelebb van az irányított Even-Shiloach algoritmus futásidejének becsléséhet.

2.1.13. Megjegyzés. [9]-ben a cikk szerzői bemutatnak egy általánosítást az irányított Even-Shiloach algoritmusra súlyozott élek esetén, azonban az élsúlyok azesetben csak nemnegatív egészek lehetnek, és a maximális méretüktől lineárisan függ a futásidő. Ezt a bizonyítást nem részletezzük, mivel rögtön következik az Even-Shiloach algoritmus bizonyításából, ha az éleket megtöbbszörözzük. Sajnos esetünkben ez az egész élsúlyozásos általánosítás nem hasznos a probléma megoldása szempontjából.

2.1.14. Megjegyzés. Világos, hogy a 2.2.1-es tétel algoritmusában az élbeszúrásokkor valóban teljesült a fenti beszúrási feltétel (mivel vagy S-beli csúcsot kötöttünk össze 1 rangú C-beli csúccsal, vagy az út-augmentáció esetén az élmegfordításhoz szúrtunk be olyan éleket amelyek végpontjai 1-gyel kisebb rangúak, mint kezdőpontjai). Azt is könnyen láthatjuk, hogy olyan uv irányított élek beszúrását is megengedhetnénk amelyekre $rang(u) = rang(v) - 1$, azonban ekkor hasonló láncreakciót indítanánk be, mint egy éltörléskor.

A 2.2.1-ben tehát majd a megmutatott módon fogjuk felhasználni az irányított Even-Shiloach algoritmust maximális párosítás keresésére. Ezt módosítjuk azzal hogy szubrutinként az általunk bemutatott súlyozott irányított Even-Shiloach algoritmust használjuk,

és az azzal nyert, mindig az adott $c \in C$ pontból induló minimális súlyú legrövidebb javító út mentén augmentálva.

Fontos azonban, hogy miután kapunk így egy maximális párosítást, amelyről reméljük, hogy (közel-) minimális súlyú, azután a kitüntetett pontból tetszőleges $v \in S \setminus M$-beli csúcsba menő alternáló út mentén augmentálhatunk. Ez a párosítás nagyságát természetesen nem változtatja, azonban a párosítás súlyát változtathatja, mégpedig éppen *távolság*(v)-vel növeli. Ezért érdemes mohó algoritmussal mindig a legkisebb távolságú $S \setminus M$-beli csúcsok mentén augmentálni (és a struktúrát továbbra is fenntartani) mindaddig amíg a legkisebb távolságú $S \setminus M$-beli csúcs távolsága negatív. Vagy esetleg valamilyen konstans lépésszámig, vagy távolsághatárig augmentálunk. Várhatóan így nem sokkal több időt igénybe véve még kisebb súlyú maximális párosítást tudunk találni. Ehhez azonban érdemes egy rendezett halmazban tárolni az $S \setminus M$-beli csúcsok távolságait (a hozzátartozó csúcsokat is velük tárolva), és a halmazt megfelelően szerkeszteni az algoritmus során.

2.2. Minimális súlyú maximális párosítás keresése

Ebben a fejezetben bemutatjuk a [7]-es cikk online maximális súlyozatlan párosítást kereső algoritmusát, amely az 2.1.8-es tétel algoritmusát használja szubrutinként. Valamint megmutatjuk, hogy a [8]-as cikk alapján ez az algoritmus még jobban teljesít amennyiben a C csúcshalmazt egyenletes eloszlással véletlenszerűen vesszük be.

2.2.1. Tétel. *Legyen $G = (S, C; E)$ egy egyszerű páros gráf, ahol $n = |S| = |C|$ és $m = |E|$. Mutatunk egy online párosítási algoritmust amely egy dinamikus $H = (S, \emptyset; \emptyset) \subset G$ részgráfból kiindulva sorban hozzáveszi C csúcsait és a hozzá tartozó E-beli éleket, és minden csúcsbevétel után fenntart egy M maximális párosítást H-ban. Azt állítjuk, hogy online párosítási algoritmus $O(m\sqrt{n}\sqrt{\log(n)})$ időben fut, és összesen $O(n\log^2(n))$ élmódosítást (éltörlést vagy élhozzáadást) hajt végre M-ben.*

Mivel az algoritmus végén $H = G$ lesz, ezért M egy maximális párosítás lesz G-ben.

Az algoritmus minden új $c \in C$ csúcs (és a hozzá tartozó E-beli élek) H-hoz való hozzávétele után keres egy c-ből induló legrövidebb javító utat, és ha talál ilyet, e mentén augmentálja a párosítást. A tétel bizonyításban megmutatjuk, hogy ez a módszer valóban végig fenntart egy M maximális párosítást H-ban, összesen $O(n\log^2(n))$ élmódosítást hajt végre a dinamikus M-ben, és az algoritmus megfelelően implementálva $O(m\sqrt{n}\sqrt{\log(n)})$ időben fut.

2.2.2. Megjegyzés. Habár az algoritmus egyszerű és intuitív, egy javító út megtalálása akár $O(m)$ időt is igénybe vehet (szélességi bejárással), így az algoritmus futásidejére csak $O(nm)$-es naiv felső becslést kapnánk. Az algoritmus gyorsaságának bizonyítása egyrészt egy, a legrövidebb javító utak eloszlásáról szóló lemmán alapszik, másrészt a megfelelő Δ választás mellett a 2.1.8 tétel algoritmusának trükkös alkalmazásával a Δ-nál rövidebb javító utak gyors tárolásán és megtalálhatóságán.

2.2.3. Lemma. *Ha a fentiek szerint minden C-beli csúcs bevételével a legrövidebb ezen csúcsból induló javító út mentén augmentálunk (amennyiben ilyen létezik), akkor tetszőleges $h \in \mathbb{R}$-re fennáll, hogy az algoritmus során legfeljebb $\frac{4n\log(n)}{h}$ darab olyan javító út mentén augmentáltunk, melyek hossza $> h$.*

Ezen lemma bizonyítása a terjedelme miatt a dolgozatban nem szerepel. Az Olvasó megtalálhatja a teljes elemi bizányítást a [7]-es cikkben.

2.2.4. Lemma. *Ez az algoritmus valóban egy M maximális párosítást talál G-ben.*

Bizonyítás. Kezdetben $H = (S, \emptyset; \emptyset)$ gráfban $M = \emptyset$ valóban egy maximális párosítás H-ban. Innen teljes indukcióval belátjuk, hogy M mindvégig maximális párosítás lesz H-ban, és így az algoritmus végén amikor $H = G$, egy M maximális párosítást kapunk G-ben. Tehát azt kell belátnunk, hogy minden új C-beli csúcs (és annak E-beli éleinek) H-hoz való hozzávételével és M-nek az algoritmus szerinti módosításával, az új M is maximális párosítás az új H-ban.

Tegyük fel indirekt, hogy egy c csúcs hozzávétele után a módosított M, jelölje M' nem maximális párosítás (a módosított) H-ban, azaz $\exists M'' \subseteq H : |M''| > |M'|$. Mivel egyetlen csúcs hozzávételével a maximális párosítás legfeljebb 1-gyel nőhet, ezért $|M''| \leq |M|+1 \leq |M'|+1$, azaz $|M''| = |M'|+1 = |M|+1$. Továbbá mivel c hozzávételével szigorúan nagyobb párosítást kaptunk, ezért $c \in M''$. Így tehát $M \triangle M''$ néhány páros hosszú alternáló út, nehány páros hosszú alternáló kör, és egy páratlan hosszú alternáló út uniója, ahol az előbbiek szerint a páratlan hosszú alternáló út egyik végpontja c (máskülönben M nem lett volna maximális). Azaz ez a páratlan hosszú c-ből induló alternáló út egy c-ből induló javító út. De ekkor az algoritmus olyan M'-t talált volna amelyre $|M'| = |M| + 1$, vagyis $|M'| = |M''|$, amivel ellentmondásra jutottunk, ezzel igazolva a lemma állítását. $\square$

2.2.5. Megjegyzés. Az előző gondolatmenetből azt is rögtön megkapjuk, hogy ha egy újonnan bevett $c \in C$ csúcsból nem találtunk javító utat, akkor a teljes algoritmus során fennáll $c \notin M$.

2.2.6. Lemma. *Az algoritmus összesen $O(n\log^2(n))$ élmódosítást hajt végre a dinamikus M párosítási gráfban.*

Bizonyítás. Alkalmazzuk 2.2.3 lemmát $h = 2^i$-ra valamely i nemnegatív egészre. Ekkor tudjuk, hogy legfeljebb $\frac{4n\log(n)}{2^i}$ olyan javító út mentén augmentáltunk az algoritmusban amelyek $> 2^i$ hosszúak. Következésképpen legfeljebb $\frac{4n\log(n)}{2^i}$ olyan javító út mentén augmentáltunk amelyek hossza a $(2^i, 2^{i+1}]$ intervallumba esik. Tehát mivel minden javító út hossza a $(0; 2n-1]$ intervallumba esik, ezért az algoritmus során használt javító utak összhossza $\leq \sum_{i=0}^{k} 2^{i+1}\frac{4n\log(n)}{2^i}$, ahol $k = \lceil \log_2(2n-1)\rceil - 1$. Azaz $\sum_{i=0}^{k} 2^{i+1}\frac{4n\log(n)}{2^i} = k8n\log(n) = O(n\log^2(n))$, és mivel a használt javító utak összhossza nyilván felső becslése az M-ben végzett élmódosítások számának, ezért az M-beli élmódosítások száma valóban $O(n\log^2(n))$. □

2.2.7. Lemma. *Megfelelő implementálás mellett az algoritmus összesített futásideje $O(m\sqrt{n}\sqrt{\log(n)})$.*

Bizonyítás. Mint azt már említettük, az algoritmus hatékony implementálásához 2.1.8 algoritmusát fogjuk felhasználni szubrutinként. Ehhez definiálunk G (pontosabban H) alapján egy D dinamikus irányított gráfot, amelyben az irányított Even-Shiloach algoritmussal gyorsan kezelhetjük az érkező $c \in C$ pontokból induló rövid javító utakat.

Legyen tehát D dinamikus irányított gráf kezdetben $D = (S \cup C \cup \{t\}, A)$, ahol t egy új csúcs ($t \notin S, C$), és A a t-ből S-be futó, valamint a t-ből C-be futó irányított élek halmaza. Minden egyes $c \in C$ csúcs hozzávételével adjuk D gráfhoz az E halmaz c csúcsot tartalmazó éleit S-ből c-be való irányítással. Ezután hagyjuk el a tc irányított élet. Ezt elvégezve az aktuális c csúcsra, az algoritmus szerint keressünk c-ből egy legrövidebb P javító utat (ezen keresés implementálását később tárgyaljuk), és ha találtunk ilyet, akkor P mentén augmentáltuk M-et, fordítsuk meg D-ben P éleit, majd pedig töröljük a ts irányított élet D-ből, ahol $s \in S$ a P javító út másik végpontját jelöli.

Könnyen látható, hogy így D irányított gráf minden $c \in C$ pont hozzávétele, és az azt magával vonó lépések elvégzése után a következő: $V(D) = S \cup C \cup \{t\}$, és $A(D)$ pedig $E(H) \setminus E(M)$-nek az éleit tartalmazza $S \to C$ irányítással, M éleit tartalmazza $C \to S$ irányítással, és még azon irányított éleket tartalmazza amelyek t-ből mennek $V(H) \setminus V(M)$-be.

Az előbbi alapján tehát, ha az újonnan hozzávett $c \in C$ csúcsra elvégeztük az javító út keresése előtti lépéseket, akkor tetszőleges c-ből induló P alternáló útnak megfelel egy P' irányított út t-ből c-be, ahol P' a P éleinek megfelelő D-beli irányított élekből áll, valamint egy ts irányított élből, amelyre $s \in S \setminus V(M)$, valamint tetszőleges $P' = t \to c$ irányított útból az első irányított él, azaz ts irányított él (ahol $s \in S \setminus V(M)$), elhagyásával egy $s \to c$ irányított utat kapunk, amelynek irányítását elhagyva a kapott $P = s \to c$ út egy alternáló út. Tehát a c-ből való legrövidebb javító út keresése H-ban ekvivalens a legrövidebb $t \to c$ irányított út keresésével.

Így tehát a 2.1.8-es tétel algoritmusát szeretnénk alkalmazni a D dinamikus gráfra t kitüntetett csúccsal, és később a futásidőre optimalizálva megválasztott Δ-val. Azaz ilyen módon amikor c-ből keresünk legrövidebb javító utat, akkor $O(1)$ időben megállapíthatjuk, hogy létezik-e $\leq \Delta - 1$ hosszú c-ből induló javító út (*rang*(c) lekérdezésével). Ha létezik ilyen, tehát $rang(c) \leq \Delta$, akkor c-ből indulva mindig eggyel kisebb rangú tetszőleges szülőt választva az 1 rangú s pont eléréséig, egy $c \to s$ legrövidebb javító utat kaptunk $O(\Delta)$ időben. Ha pedig nem létezik ilyen rövid javító út, azaz $rang(c) = \Delta + 1$, akkor szélességi bejárással $O(m)$ időben megtalálhatjuk a legrövidebb c-ből induló javító utat, vagy bizonyítékot kapunk arra, hogy nincs c-ből induló javító út H-ban.

Ahhoz viszont, hogy 2.1.8 algoritmusát valóban alkalmazhassuk, meg kell bizonyosodnunk arról, hogy a dinamikus D irányított gráfba beszúrt élek sosem csökkentették egyik csúcs rangját sem. Az világos, hogy a $c \in C$ csúcs D-be való bevétele után a sc irányított élek (ahol $s \in S$) beszúrása nem okoz problémát, mivel $rang(c) = 1$. Egyedül a javító útnak megfelelő $s \to c$ irányított úton történő élmegfordítás okozhat gondot. Ha azonban az élmegfordítást úgy végezzük, hogy az irányított út mentén sorra beszúrjuk az irányított út soron lévő irányított élének megfordítását majd töröljük az irányított élet, akkor láthatjuk, hogy az így beszúrt pq irányított élre mindig $rang(p) = rang(q) + 1$, így ezeket az éleket is beszúrhatjuk.

Az algoritmus során tehát egyrészt fenn kell tartani a 2.1.8-es tétel algoritmusának tárolt struktúráját összesen $O(\Delta m)$ időben, másrészt a $\leq \Delta - 1$ hosszú javító utak esetén a javító út megtalálása (és az annak mentén történő augmentálás) $O(\Delta)$ időben zajlik, az ennél hosszabb javító utaké pedig $O(m)$ időben. Így tehát a 2.2.3-es lemmát használva $h = \Delta - 1$-re, azt kapjuk, hogy az összesített futásidő $O\left(\Delta m + \left(n - \frac{4n\log(n)}{\Delta - 1}\right)\Delta + \frac{4n\log(n)}{\Delta - 1}m\right) = O\left((m+n)\Delta + \frac{mn\log(n)}{\Delta}\right)$, így azzal az ésszerű feltevéssel élve, hogy $n = O(m)$, a teljes futásidő $O\left(m\Delta + \frac{mn\log(n)}{\Delta}\right)$, vagyis ezt $\Delta = O(\sqrt{n}\sqrt{\log(n)})$ minimalizálja, és így a futásidő

valóban $O(m\sqrt{n}\sqrt{\log(n)})$. $\square$

Ezzel tehát befejeztük a 2.2.1-es tétel bizonyítását.

Ha azonban ugyanezen mohó algoritmus mellett a C-beli csúcsokat ráadásul egyenletes eloszlással véletlenszerűen vesszük be, akkor még jobb eredményt kapunk. Mégpedig a [8]-as cikk II.2-es tétele szerint a felhasznált augmentáló utak összhossza legalább $1 - \frac{1}{n^2}$ valószínűséggel $O(n\log(n))$, azaz az algoritmus legalább $1-\frac{1}{n^2}$ valószínűséggel $O(n\log(n))$ élmódosítást hajt végre. Mivel neurális háló tanítása esetén a batch-ek elemei egyenletes eloszlással véletlenszerűen kerülnek kiválasztásra, ezért (a soron lévő batch random permutálása után, ha szükséges) a batch elemeihez tartozó C-beli pontokat sorra véve az előbbi tétel feltétele teljesül.

Továbbá a [8]-as cikk III.1-es tétele szerint ha a G gráf olyan, hogy minden $c \in C$ csúcsából $\Theta(\log(n))$ (S-en egyenletes eloszlással, egymástól függetlenül) véletlen választott S-beli csúcshoz vezet él, akkor legalább $1 - O(\frac{1}{n})$ valószínűséggel legfeljebb $O(n)$ élmódosítást végzünk az algoritmus során. Persze ez a feltétel nem fog teljesülni a gyakorlati példánkon. Mivel az S-beli target pontokat véletlenszerűen (és egymástól függetlenül) választjuk meg a gömbfelületen, és majd látjuk, hogy minden $c \in C$ csúcsot valóban $\Theta(\log(n)$ S-beli csúccsal kötünk össze, így valóban igaz, hogy tetszőleges $c \in C$ csúcsnak $\Theta(n)$ éle van, melyek egyenletes eloszlással, egymástól függetlenül mennek egyes S-beli csúcsokba, azonban a különböző C-beli csúcsokra az egyes élválasztások közel sem lesznek függetlenek. Viszont később még egyes kiegészítő éleket is hozzá fogunk venni a gráfhoz, éppen ezen logika szerint, és mivel egy gráf élhalmazának bővítésével a bővítés előtti augmentáló utak megmaradnak, ezért ez esetben teljesül a tétel feltétele. Többek közt ez indokolja, hogy ha valamilyen kis Δ-nál nagyobb javító utakat már nem próbálunk megkeresni, akkor ezzel közel-maximális párosítás találunk.

2.3. Teljes párosítás random páros gráfban

Mint azt már említettük, célunk az, hogy egy $G = (C, S; E)$ élsúlyozott teljes páros gráfban szeretnénk egy minimális súlyú teljes párosítást találni, ahol $n = |C| = |S|$. Azt is előrevetítettük, hogy mivel a gyakorlatban erre a problémára a magyar módszert alkalmazva a futásidő túl hosszú lenne, ezért ehelyett egy közel-minimális súlyú közel-teljes párosítást szeretnénk találni.

Erdős és Rényi a [16]-os cikkben 1963-ban bebizonyította a következő tételt.

2.3.1. Tétel. *(Erdős-Rényi, 1963) Legyen $B_{n,m}$ egy véletlen páros gráf, amelynek mindkét csúcsosztálya n csúcsból áll, és $m = n(\log(n) + c_n)$ véletlen éle van. Ekkor*

$$\lim_{n\to\infty} \mathbb{P}(B_{n,m}\text{-ban létezik teljes párosítás}) = \begin{cases} 0, ha\ c_n \to -\infty \\ e^{-2e^{-c}}, ha\ c_n \to c \\ 1, ha\ c_n \to \infty \end{cases} \tag{2.3}$$

Az előbbi tétel alapján, valamint a [14]-es cikk eredményei alapján az a sejtésünk, hogy ha $G' = (C, S; E') \subset G$ élsúlyozott ritka páros gráfot úgy alkotjuk meg, hogy minden $c \in C$ csúcsot a hozzá legközelebb lévő $\approx \log(n)$ darab S-beli csúccsal kötjük össze, akkor ha C körülbelül egyenletesen oszlik el a gömbfelületen (S-re pedig ez definíció szerint teljesül), akkor a gráfban (nagy valószínűséggel) lesz közel-teljes párosítás.
Habár erre elméleti bizonyítékot nem adunk, a gyakorlati példán bemutatjuk, hogy ez valóban teljesül.

3. fejezet

Implementálás

Ebben a fejezetben megmutatjuk, hogy pontosan hogyan oldjuk meg a felvetett párosítási problémát. Továbbá bemutatjuk a páros gráf ritkításához, a futásidő lecsökkentése céljából használt *ANNOY* algoritmust [15], és annak elméletét, a *lokálitás-érzékeny hashelést.*

3.1. Közeli szomszédok keresése

$\mathbb{R}^n$-ben legyen adott egy $n-1$ dimenziós gömbfelületen N darab különböző pont (amelyeket a gömbön egyenletes eloszlással generáltunk), ezek halmazát jelölje S, valamint adott további N darab pont a gömbön, melyek eloszlása ismeretlen, és ezek halmazát jelölje C, illetve legyen $K << N$ rögzített pozitív egész. A következő problémát szeretnénk megoldani: adjunk egy gyors algoritmust amely minden C-beli pontra megadja a K hozzá legközelebbi S-beli pontot.

Láthatjuk, hogy ha C minden egyes $c \in C$ pontjára $\forall s \in S$-re kiszámoljuk $\|c - s\|$ távolságot, és c-hez számontartjuk a K darab legkisebb normát eredményező s-et, akkor ezzel az algoritmussal megoldottuk a problémát $O(N^2 n \log(K))$ lépésben. Ez viszont a megoldandó gyakorlati problémára túl lassú lenne, sokkal gyorsabb algoritmust szeretnénk találni.

Vegyük észre, hogy ha valahogy felosztjuk a teret (vagy legalábbis a gömbfelszínt) M partícióra úgy, hogy minden partícióba körülbelül $\frac{N}{M}$ pont jusson S-ből, akkor ha minden $c \in C$ pontra csak a partíciójában lévő körülbelül $\frac{N}{M}$ darab S-beli pontra vizsgáljuk a tőle vett távolságát, és ezekből vesszük a K legkisebbet, akkor így egy $O\left(N\frac{N}{M} n \log(K)\right)$ idejű algoritmust kapunk. Így tehát megfelelő M választással lényegesen gyorsabb algoritmust kapunk, azonban ezzel az algoritmussal két nagy probléma is van. Az egyik, hogy semmi

sem garantálja, hogy a $c \in C$ ponthoz K legközelebbi S-beli pont mindegyike ugyanabban a partícióban van mint c, mi csak a partíción belüli K legközelebbi pontot találjuk meg. A másik probléma, hogy egy tetszőleges $c \in C$ pontra gyorsan (esetünkben legfeljebb $O\left(\frac{N}{M} n \log(K)\right)$ időben) meg kell tudnunk mondani, hogy melyik partícióba tartozik.

Az első problémát nem oldjuk meg teljesen, ebből adódik az *ANNOY* (Approximate Nearest Neighbors Oh Yeah) algoritmus [15] közelítő jellege. Az algoritmus bemutatásához először a lokalitás-érzékeny hashelést tárgyaljuk, amely az algoritmus alapötlete, és egy hasonló példán elméleti számolásokat végzünk. Ezután áttérünk az ANNOY algoritmus rövid bemutatására, amelyet később felhasználunk a ritka páros gráf készítéshez.

3.1.1. Lokalitás-érzékeny hashelés

Legyen M egy metrikus tér, és $P \subset M$ (nem feltétlenül véges) ponthalmaz, valamint H véges halmaz. Ekkor a $h : P \mapsto H$ függvényt a P ponthalmaz lokalitás-érzékeny (probabilisztikus) hashfüggvényének hívjuk, ha $\exists r > 0 : \forall x, y \in P, d_M(x, y) < r : \mathbb{P}(h(x) = h(y)) > \frac{1}{|H|}$. Vagyis a kellően közeli pontok hashelt értékei $\frac{1}{|H|}$-nál nagyobb valószínűséggel megegyeznek.

Legyen most $M = \mathbb{R}^n$ és P ponthalmaz az $n-1$ dimenziós gömb standard beágyazása $\mathbb{R}^n$-be, és legyen k rögzített pozitív egész. Generáljunk k darab véletlen homogén irányított hipersíkot: $h_1, h_2, \ldots, h_k$ origón áthaladó irányított hipersíkokat, ahol véletlen alatt azt értjük, hogy ezen irányított hipersíkok normált normálvektorait, rendre $v_1, v_2, \ldots, v_k \in \mathbb{R}^n$ vektorokat, egyenletes eloszlással választottuk az origó középpontú 1 sugarú $n-1$ dimenziós gömbön.
Ekkor $p \in P$ pont postosan akkor van h_i irányított sík pozitív oldalán, ha $\langle p, v_i \rangle \geq 0$, és pontosan akkor van a negatív oldalán, ha $\langle p, v_i \rangle \leq 0$. Így minden $p \in P$ ponthoz definiáljuk úgy a $h : P \mapsto M$ függvényt, ahol M a k hosszú bináris sorozatok halmaza, hogy $h(p)$ bináris sorozat i-edik tagja legyen 1, ha $\langle p, v_i \rangle \geq 0$, és 0, ha $\langle p, v_i \rangle < 0$. Azt állítjuk, hogy az így definiált h valóban lokalitás-érzékeny (probabilisztikus) hashfüggvény.
Legyen $p, q \in P$ pontok, és jelölje $V(r)$ a P gömb azon részének felületét melyek a p ponttól legfeljebb r távolágra lévő pontok összessége (ez a definíció nyilván független p pont választásától). Annak a valószínűsége, hogy p és q pont hashelése az i-edik helyen eltér:

$$\mathbb{P}\left(\left(\langle p, v_i \rangle \geq 0 \wedge \langle q, v_i \rangle < 0\right) \vee \left(\langle p, v_i \rangle < 0 \wedge \langle q, v_i \rangle \geq 0\right)\right) =$$

$$= \mathbb{P}\left(\langle p, v_i\rangle \geq 0 \wedge \langle q, v_i\rangle < 0\right) + \mathbb{P}\left(\langle p, v_i\rangle < 0 \wedge \langle q, v_i\rangle \geq 0\right) = 2\frac{V(d(p,q))}{V} \quad (3.1)$$

ahol V a P gömb felülete. Tehát annak a valószínűsége, hogy p és q azonos hashértéket kap $\mathbb{P}(h(p) = h(q)) = 1 - \left(2\frac{V(d(p,q))}{V}\right)^k$. Tehát, mivel ha $r \to 0$, akkor $V(r) \to 0$, ezért ekkor $1-\left(2\frac{V(r)}{V}\right)^k \to 1$, így a fent definiált h valóban egy lokalitás-érzékeny hashfüggvény.

Annak érdekében, hogy közeli pontokat nagyobb valószínűséggel találjuk meg, alkalmazunk még egy trükköt, mégpedig, hogy L-szer végezzük el a fenti procedúrát, így kapva $h_1, h_2, \ldots, h_L$ hashfüggvényeket. Ha $\exists j \in \{1, 2, \ldots, L\} : h_j(p) = h_j(q)$ akkor azt mondjuk, hogy q közeli pontja p-nek. Jelöljük p közeli pontjainak halmazát $N(p) \subset P$-vel. Így tehát tetszőleges $q \in P$ pontra

$$\mathbb{P}(q \notin N(p)) = \mathbb{P}\left(\bigwedge_{j=1}^{L} h_j(p) \neq h_j(q)\right) = \prod_{j=1}^{L} \mathbb{P}(h_j(p) \neq h_j(q)) =$$

$$= \mathbb{P}(h_1(p) \neq h_j(q))^L = \left(2\frac{V(d(p,q))}{V}\right)^{kL} \quad (3.2)$$

Azaz, ha $d(p,q) \leq r$, akkor $\mathbb{P}(q \notin N(p)) \leq \left(2\frac{V(r)}{V}\right)^{kL}$, és így $\mathbb{P}(q \in N(p)) \geq 1-\left(2\frac{V(r)}{V}\right)^{kL}$, tehát kis r-re nagyon nagy a valószínűsége annak, hogy $q \in N(p)$.

Térjünk vissza a szekció bevezetésében definiált közeli szomszédkeresés problémára. Az előbbi algoritmusban a hashkódok kiszámolása egy $c \in C$ pontra $O(Lkn)$ időt vesz igénybe, majd a c-hez tartozó L darab térrészbe eső S-beli pontokból a K legközelebbi kiválasztása várható értékben legfeljebb $O(Ln\frac{N}{2^k})$ feltéve, hogy $k \leq n$. Tehát az egész algoritmus futásideje $O(NLkn + NLn\frac{N}{2^k})$, vagyis $k = \Theta(\log(N))$ választással $O(NLn\log(N))$. Persze ehhez szükséges $n = \Omega(\log(N))$; ezt a kellemetlenséget az ANNOY algoritmus kiküszöböli.

3.1.2. ANNOY algoritmus

Most bemutatjuk az [15]-ös hivatkozáson elérhető ANNOY algoritmust, és annak ötleteit. Az előbbi algoritmusban az új síkokkal két részre osztottuk az összes már meglévő térrészt, most azonban egy síkkal csak egyetlen térrészt kívánunk két részre osztani. Ezt úgy is megfogalmazhatjuk, hogy az S ponthalmazhoz építünk egy bináris fát amelyben az egyes "véletlen" irányított síkok a bináris fa csúcsai, továbbá a bináris fa egyes leveleihez azokat az S-beli pontokat rendeljük amelyek abban a térrészben vannak amelyet a gyökérből a levélhez vezető úton úgy kapunk meg, hogy a gyökérnél a teljes térből indulunk ki, majd,

ha az út éle bal-él a bináris fában, akkor az aktuális térrésznek és az irányított sík negatív félterének a metszete lesz az új térrész, ha pedig az út éle jobb-él, akkor az irányított sík pozitív félterének.

A "véletlen" irányított síkokat most úgy választjuk meg, hogy a két részre osztandó térrészből egyenletes eloszlással választunk két különböző S-beli pontot, és ezeknek vesszük a felezősíkját (és a sík irányítását a két pont kiválasztási sorrendje szerint). Továbbá egy térrészt nem osztunk tovább, ha az abban lévő pontok száma egy előre meghatározott kritikus érték alatt van.

Az ANNOY algoritmus egyik fontos trükkje, hogy a fenti lokalitás-érzékeny hasheléses algoritmussal ellentétben most egy $c \in C$ pontra nem csak azt nézzük meg a bináris fát végigkövetve, hogy melyik levélhez tartozó térrészbe esik (és melyek a levélhez tárolt, a térrészbe eső S-beli pontok), hanem c-nek a vizsgált síkoktól való előjeles távolságait is rendezve számontartjuk (amelyet a kiszámolt skaláris szorzatból úgyis megkapunk), majd, ha c kellően közel van egy elválasztó síkhoz, akkor annak a síknak a másik oldalán is továbbhaladunk (mivel ekkor jó eséllyel a sík másik oldalán is lesznek c-hez közeli pontok). Mivel a síkoktól való távolságokat rendezve tároljuk, ezért megadhatjuk, hogy minden egyes c pontra hány levélhez való térrészbe tartozó S-beli pontokat kívánjuk vizsgálni.

Itt is elsütjük azt a trükköt, hogy 1 fa helyett L darab fát építünk, és egy c csúcshoz a kapott térrészek S-beli részhalmazainak unióját vesszük. Majd pedig ezen S-beli pontok közül keressük meg a c-hez K legközelebbit a $\|c - s\|$ értékek kiszámolásával és a legkisebb K-hoz tartozó s pont nyilvántartásával.

3.2. Párosítási algoritmus alkalmazása, a háló tanítása

A [7]-es cikk párosítási algoritmusát tehát úgy módosítjuk, hogy a szubrutinként használt irányított Even-Shiloach algoritmust lecseréljük a dolgozatban leírt élsúlyozott változatra.

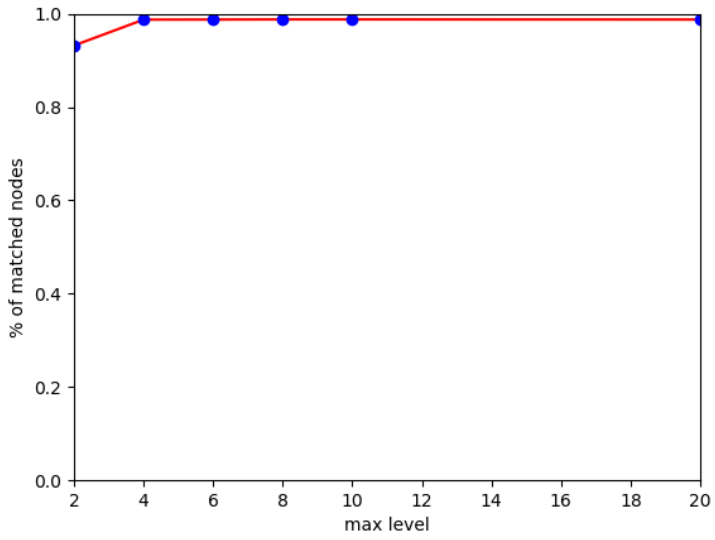
Egy epoch kezdetén tehát generálunk n darab target pontot, azaz legeneráljuk S halmazt. Ezután elvégezzük az aktuális látens pontokat, azaz C halmazt kiszámoljuk az aktuális enkóder használatával. Majd az S halmazra az ANNOY algoritmussal LSH-et végzünk, és ezt követően minden egyes $c \in C$ csúcsra meghatározzuk a $\lceil \log(n) \rceil$ (közelítőlegesen) legközelebbi S-beli szomszédját, és ezen közelségi gráf lesz $G' = (C, S, E') \subset G$ élsúlyozott ritka páros gráf.

Miután G' gráfot meghatároztuk, felépítünk egy új H irányított gráfot a módosított

párosítási algoritmusunkkal a korábban prezentált módon. Tehát Az egyes klienseket egyesével bevesszük a hozzájuk tartozó élekkel együtt (megfelelő irányítással), majd a bevett kliensből minimális súlyú legrövidebb alternáló utat keresünk, és ha találunk, akkor amentén augmentálunk. Eközben M párosítást is végig fenntartjuk.

A neurális hálót pedig batch-enként tanítjuk úgy hogy a veszteségfüggvény a rekonstrukciós veszteségből, és az M élsúlyok összegének $\frac{1}{n}$-szereséből tevődik össze.

Mint ígértük, megvizsgáljuk, hogy ha C-t is egyenletes eloszlással generáljuk a gömb felszínén, akkor mekkora lesz a párosítás mértéke. Pontosabban, megmutatjuk, hogy ha Δ értéke relatíve kicsi, és a párosítási algoritmusunkat még azzal is módosítjuk, hogy szélességi bejárással már ne keressen Δ-nál hosszabb legrövidebb alternáló utat, akkor is közel-teljes párosítást tudunk találni. (Δ-t az alábbi ábrán a *max level* változó jelöli a programkódból kifolyólag.)



Azonban ha C eloszlása koncentrált, akkor persze nem tudunk így közel-teljes párosítást találni (azonban a G'-beli maximális párosítást szintén meg tudjuk közelíteni kis Δ-val, a szélességi bejárásos legrövidebb alternáló út keresést elhagyva). Ezért a G' gráfot bővítjük, mégpedig minden egyes $c \in C$ klienshez hozzáveszünk további $\lceil \log(n) \rceil$ darab random élet. Azt tapasztaljuk, hogy ekkor szintén kis Δ-ra már közel-teljes párosítást kapunk. Azt tapasztaljuk, hogy már $\Delta = 4$ választás is esetünkben teljességgel kielégítő volt ezzel a módszerrel.

Tehát vegyünk az első epoch esetén még G'-höz minden $c \in C$ csúcsból $\lceil \log(n) \rceil$ darab

random élet, és vizsgáljuk meg az epoch végén a párosítás nagyságát. Ha ez elér egy bizonyos küszöböt (pl. 0.95), akkor a következő epoch-ban már minden $c \in C$ csúcshoz 1-gyel kevesebb random segéd élet veszünk fel az új G' mellé, és így tovább, míg elérünk oda, hogy nincs szükségünk segéd élekre, és tovább tanítjuk a hálót.

3.3. Kísérleti eredmények

Most bemutatjuk, hogy a fenti algoritmus súlyozott és súlyozatlan változata hogyan teljesít a gyakorlatban. Ehhez a magyar módszerrel fogjuk összehasonlítani őket. A gyakorlati példánkban olyan $G = (C, S, E)$ teljes páros gráffal dolgozunk, ahol $n = 50000$, azaz $|C| = |S| = 50000$, így ezen a gráfon szeretnénk az algoritmusokat összehasonlítani. Sajnos azonban ez nem fizibilis, mivel egyrészt mind a magyar módszernek, mind pedig a prezentált súlyozott párosítási algoritmusunknak túl nagy lenne a futásideje. Másrészt pedig, $n^2 = 2.5 \cdot 10^9$ élsúly tárolása is speciális hardware-t venne igénybe. Habár ez kiküszöbölhető az élsúlyok külső memóriában való tárolásával, vagy az élsúlyok újraszámlálásával, mindkét módszer lényegesen tovább lassítaná az algoritmusokat.
A tesztgráfokban minden esetben C-t és S-et egyenletes eloszlással generáljuk az $\mathbb{R}^{10}$-be standard módon beágyazott S^9 9-dimenziós gömbfelületről (10-dimenziós standard normális eloszlásból generált pontok normalizálásával), innen kapva az élsúlyokat a pontpárok euklideszi távolságának kiszámításával. A súlyozott- és súlyozatlan Even-Shiloach algoritmusoknál pedig minden esetben $\Delta = 4$-et állítottunk be. Az utóbbi paraméter növelésével megnövelhetnénk a talált párosítás méretét a futásidő rovására.

Mivel az prezentált párosítási algoritmusok súlyozott és súlyozatlan változata is az ANNOY módszerrel ritkított $G' = (C, S, E') \subset G$ gráfokon dolgozik, amelyekre szintén $|C| = |S| = n = 50000$, és minden $c \in C$ csúcsból $\lceil \log(n) \rceil = 11$ él megy a hozzá (az ANNOY módszerrel meghatározott, közelítőlegesen-) legközelebbi S-beli csúcsokba, ezért ez lesz az egyik tesztgráfunk.
A magyar módszer az előbbi gráfunkra viszont még mindig túl lassú lenne (becsült futásidő: 400 nap), ezért ehelyett még másik két tesztgráfon teszteljük mindhárom algoritmust. Az egyik tesztgráf $n = 500$-ra a teljes páros gráf, a másik pedig ugyanígy $n = 500$-ra az ANNOY algoritmussal ritkított páros gráf lesz.

Várhatóan a neurális háló tanítása közben lényegesen jobb eredményt fogunk elérni, ugyanis ekkor már az egyes $c \in C$ pontokra a $d(c, S)$ távolságok átlaga (tehát c távolsága a hozzá legközelebbi S-beli ponttól) lényegesen kisebb lesz.

Az eredményeket az alábbi táblázatok foglalják össze. A táblázatokban a súlyozatlan (irányított) Even-Shiloach algoritmuson alapuló párosítási algoritmusra *súlyozatlan ESM* néven, az általunk prezentált súlyozott (irányított) Even-Shiloach algoritmuson alapuló párosítási algoritmusra pedig *súlyozott ESM* néven hivatkozunk.

n=500, ritkított páros gráf			
Párosítási algoritmus	Magyar módszer	Súlyozatlan ESM	Súlyozott ESM
Párosított csúcsok aránya	0.998	0.984	0.982
Párosítás élhossz átlaga	0.6602	0.7457	0.6778
Párosítás összsúlya	330.1	366.9	332.8
Futási ideje (s)	20.88	0.3080	0.7214

n=500, teljes páros gráf			
Párosítási algoritmus	Magyar módszer	Súlyozatlan ESM	Súlyozott ESM
Párosított csúcsok aránya	1.0	1.0	1.0
Párosítás élhossz átlaga	0.6565	1.401	0.7092
Párosítás összsúlya	328.2	700.3	354.6
Futási ideje	1154.6	10.30	189.0

n=50000, ritkított páros gráf			
Párosítási algoritmus	Magyar módszer batch-enként	Súlyozatlan ESM	Súlyozott ESM
Párosított csúcsok aránya	1.0	0.9918	0.9897
Párosítás élhossz átlaga	0.7185	0.4539	0.3995
Párosítás összsúlya	35925	22507	19767
Futási ideje	21287	27.05	74.80

Mivel mint már említettük, hogy $n = 50000$ estén még a ritkított páros gráfra sem tudjuk futtatni a magyar módszert, ezért az legutóbbi táblázatban a magyar módszer helyett a NAT módszert bevezető [1]-es cikkben használt algoritmusát vizsgáltuk. Azaz a magyar módszer helyett a magyar módszer $b = 250$-es batch-enkénti változatát teszteltük. (Az eredeti cikk $b = 256$-es batch mérettel dolgozik, ezt az oszthatóság miatt 250-re kerekítettük.) Kihangsúlyozzuk, hogy természetesen ennél az algoritmusnál is csökken a párosítás összsúlya a tanítás alatt, azonban a kiszámolt nagy párosítási élhossz átlag is azt sejteti, hogy a NAT ezen algoritmus mellett a gömbfelületen szétszórja az egymáshoz hasonló klasztereket is.

Mindegyik algoritmust 2 magos, 4 szálas, CPU-n [18] futtattuk 1.60 GHz-es alapfrekvencián. A mérési eredmények reprodukálhatóak dolgozat keretében megírt [5]-beli *hungarian_method.py*, *NAT_unweighted_graph.py* és *NAT_weighted_graph.py* futtatásával (a *directed_unweighted_ES.py* és *directed_weighted_ES.py* fájlok mellett).

Irodalomjegyzék

[1] Piotr Bojanowski és Armand Joulin, *Unsupervised Learning by Predicting Noise*, 2017 (`https://arxiv.org/pdf/1704.05310.pdf`)

[2] Huszár Ferenc, *Unsupervised Learning by Predicting Noise: an Information Maximization View*, 2017 (`https://www.inference.vc/unsupervised-learning-by-predicting-noise-an-information-maximization-view-2/`)

[3] Ismeretlen szerzők, *Neural Clustering by Predicting and Copying Noise*, 2018 (`https://openreview.net/pdf?id=BJvVbCJCb`)

[4] `https://github.com/zsoltzombori/generative_modeling`

[5] `https://github.com/ttossenb/generative_modeling`

[6] `https://github.com/ttossenb/GenEnc`

[7] Aaron Bernstein, Jacob Holm és Eva Rotenberg, *Online Bipartite Matching with Amortized $O(\log^2(n))$ Replacements*, 2018 (`https://arxiv.org/pdf/1707.06063.pdf`)

[8] Kamalika Chaudhuri, Constantinos Daskalakis, Robert D. Kleinberg és Henry Lin, *Online Bipartite Perfect Matching With Augmentations*, 2009 (`https://ieeexplore.ieee.org/document/5062016`)

[9] Valeria King, Mikkel Thorup, Jie Wang (Szerk.), *Computing and Combinatorics*, 2001, fejezet: *A Space Saving Trick for Directed Dynamic Transitive Closure and Shortest Path Algorithms*

[10] Shimon Even, Yossi Shiloach, *An On-Line Edge-Deletion Problem*, 1981
(https://www.uni-trier.de/fileadmin/fb4/prof/INF/DEA/Uebungen_LVA-Ankuendigungen/ws07/KAuD/onl.pdf)

[11] Valerie King, *Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs*, 1999
(https://pdfs.semanticscholar.org/51d5/fbb94b775743e95f31f291c96f254e84d09a.pdf)

[12] Michael A. Nielsen *Neural Networks and Deep Learning*, 2015
(http://neuralnetworksanddeeplearning.com/)

[13] Ian Goodfellow, Yoshua Bengio és Aaron Courville, *Deep Learning*, 2016
(http://www.deeplearningbook.org/)

[14] Alan Frieze és Boris Pittel *Perfect mathings in random graphs with prescribed minimal degree*, 2002
(http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.3097&rep=rep1&type=pdf)

[15] Erik Bernhardsson, *Nearest neighbors and vector models – part 2 – algorithms and data structures*, 2015
(https://erikbern.com/2015/10/01/nearest-neighbors-and-vector-models-part-2-how-to-search-in-high-dimensional-spaces.html)

[16] Erdős Pál és Rényi Alfréd, *On Random Matrices*, 1964
(https://www.renyi.hu/~p_erdos/1964-14.pdf)

[17] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg és Li Fei-Fei *ImageNet Large Scale Visual Recognition Challenge*, 2015, 6.4-es szekció
(https://arxiv.org/pdf/1409.0575.pdf)

[18] https://ark.intel.com/content/www/us/en/ark/products/75459/intel-core-i5-4200u-processor-3m-cache-up-to-2-60-ghz.html