Eötvös Loránd Tudományegyetem

Természettudományi Kar

Micskó Máté Benedek

Pontos algoritmusok NP-nehéz feladatokra

Szakdolgozat

Matematika BSc

Témavezető:

Szabó Dániel Péter

Operációkutatási tanszék

Budapest, 2025

Köszönetnyilvánítás

Nagyon hálás vagyok Szabó Dániel Péternek, a témavezetőmnek a témaötletért, a félév során nyújtott segítségéért és idejéért, amikkel lehetővé tette a szakdolgozatom létrejöttét.

Tartalomjegyzék

1. Bevezetés **4**

1.1. Pontos algoritmusok 4

1.2. Data Reduction 5

2. Measure & Conquer **6**

2.1. Branch & Reduce algoritmusok 7

2.2. Random lokális keresés 9

2.2.1. Programok 10

3. Directed Multiway Cut **16**

3.1. Problémakör 16

3.2. Mingyu Xiao Algoritmusa 17

3.3. Javított algoritmus 21

4. Alkalmazás **25**

4.1. Minimális dominláló halmaz 25

5. Összefoglalás **28**

1. Bevezetés

1.1. Pontos algoritmusok

A pontos algoritmusok célja, hogy pontos megoldást találjanak NP-nehéz problémákra a lehető legrövidebb legrosszabb-esetbeli futásidővel. Ezzel a területtel a hatvanas, hetvenes években kezdtek el foglalkozni (Held és Karp 1962; Tarjan és Trojanowski 1977), és az utóbbi években egyre több figyelmet kapott, és kap a mai napig. Több okot is fel lehet sorolni a pontos algoritmusok népszerűségére:

- Vannak gyakorlati alkalmazások, ahol szükséges a pontos megoldás megtalálása.
- Sok problémát nehéz közelíteni, vagy a lehetséges közelítés nem elég jó. Például a maxmimális független csúcshalmaz számát nehéz $O(n^{1-\varepsilon})$ pontossággal becsülni, tetszőleges $\varepsilon > 0$ konstansra, ha P≠NP.
- Az exponenciális futásidő alapját csökkenteni, például elérni $O(2^n)$ helyett $O(1.9^n)$ futásidőt, növeli az adott időn belül megoldható probléma méretét egy multiplikatív konstanssal. Míg gyorsabb gépek használatával csak (kis) additív konstanst lehet elérni.
- A pontos algoritmusok felépítése és analízise segít jobban megérteni az NP-nehéz problémákat. Létrehoz új és érdekes kombinatorikai és algebrai kihívásokat.

Az egyik legfontosabb technika pontos algoritmusok felépítésére a Branch & Reduce – elágazás és csökkentés –, ami Davis és Putman 1960-as cikkére ([5]) vezethető vissza . Ez a módszer bevezet csökkentési szabályokat, amikkel leegyszerűsíti az elágazáskor keletkező részproblémákat, amiket aztán rekurzívan megold. Ugyanakkor nem használ alsó becslést, mint a Branch & Bound. Utóbbiról a továbbiakban nem lesz szó, mert az az elméleti optimum helyett inkább a tapasztalati problémákra (konkrét esetekben) ad kis futásidőt.

1.2. Data Reduction

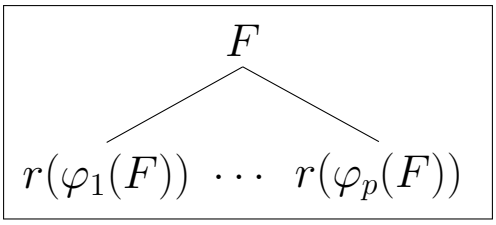
Az NP-nehéz feladatok pontos megoldása nagy számítási feladat. Ezért sok egyszerűsítést és trükköt, azaz adatcsökkentést vezettek be, amik hatékonyabbá teszik a megoldó algoritmusokat. Az adatcsökkentő szabályok polinom idejű műveletek, amik lecsökkentenek egy problémát úgy, hogy a megmaradó feladat optimális megoldása könnyen kiterjeszthető az eredeti feladat optimális megoldására. Ezek az adatcsökkentési szabályok különösen hasznosnak bizonyultak elágazási algoritmusoknál, ahol jó csökkentések gyorsan megoldható problémákhoz vezetnek. Ilyen módszer a Measure & Conquer is, amit sok más adatcsökkentési szabállyal párhuzamosan használva, a már javított hatékonyságú algoritmuson is lehet javítani.

2. Measure & Conquer

A sokat használt Branch & Reduce algoritmus, mint fent utaltam rá, az eddigi egyik legjobb eszköz az NP-nehéz feladatok pontos megoldására. Azonban kijelenthetjük, hogy az ilyen rekurzív algoritmusok analízise még mindig távol áll attól, hogy szoros alsó határt adjon a legrosszabb futásidőre. Éppen ezért, ennek javítására jött létre a Measure & Conquer, ami a standardhoz képest új mértéket ad a különböző részproblémáknak; ezzel kihasználva az algoritmus sajátosságait egy jobb alsó becslés érdekében.

Nem standard mérték. Az NP-nehéz feladatokra adott leggyorsabb algoritmusok általában bonyolultak; sok-sok nem triviális elágazást és szabályt tartalmaznak, amiket mély esetszétválasztással hoztak létre. Azonban az analízisük annál egyszerűbb. Van egy (standard) mérték a részproblémák méretére (pl.: gráfok csúcsainak vagy éleinek száma). Ez a mérték ad alsó határt a műveleteknek minden elágazási lépésnél.

A Measure & Conquer ennek a mértéknek a megválasztására fókuszál. Hiszen, ahogy később látni fogjuk, egy jobb mérték képes csökkenteni a futásidőt, akár nagy mértékben is. Ezt használja fel Mingyu Xiao a Directed Multiway Cut probléma megoldásában ([17]), és ennek az algoritmusnak a javításával fogok jobb időkorlátot adni ugyanerre a problémára. Megmutatva, hogy nem csak egy jó algoritmussal, és ahhoz egy jó mértékkel lehet jobb eredményt elérni, hanem olykor egy algoritmuson kicsit változtatva és a mértéken kicsit optimalizálva is. Összehasonlításképpen az eddigi eredmény futásideje $O(1.9967^n)$, míg a javítotté $O(1.9959^n)$. Ez is utal arra, amit [8]-ben megfogalmaztak, hogy még egy okosan választott mérték sem ad biztosan közel optimális analízist ; van helye a további javításoknak.



2.1. ábra. Az F gyökérből induló elágazás.

2.1. Branch & Reduce algoritmusok ([14] alapján)

Egy tipikus Branch & Reduce algoritmus egy F_0 kezdeti problémára a következőképpen jár el. Először végrehajt egy polinom idejű csökkentést $F_0 \to r(F_0)$, ahol $F \coloneqq r(F_0)$ megoldását tekintve ekvivalens F_0-al. Ha ez nem elég F_0 megoldhatóságának eldöntésére, akkor felosztja F-et részproblémákra φ_i szerint, a 2.1 ábrán látható módon. A φ_i elágazásokra igaz, hogy F megoldható ekvivalens azzal, hogy létezik i, amire $F_i \coloneqq \varphi_i(F)$ megoldható. Az így keletkező T fának a levelei olyan állapotok, amiből a megoldás polinomiális időben kiszámolható, tehát a fa méretéből adódik a futásidő. Az ilyen algoritmusok analízise általában ugyanazzal a mértékkel történik, mint amivel aztán a futásidőt megadják (pl.: a gráf csúcsainak száma). Legyen ez a mérték μ és a probléma mérete $\mu(F) = n$. Ekkor T leveleinek a számát, $l(T)$-t, elég fölülről becsülni, mert a futásidő $O^*(l(T))$; a levelek számától függ. Legyen $\Phi(F, F_i) = \mu(F) - \mu(F_i)$. Ekkor $\Phi(F) = (\Phi(F, F_i))_{i\in[p]}$ a részproblémák méretváltozásainak vektora, vagyis az *elágazási vektor*. Ezekre az elágazásokra igaz, hogy a keletkező ágakban összesen nem csökken a levelek száma

$$l(F) \leq \sum_{i=1}^{p} l(F_i).$$

Ahol $l(F)$ az F-ből kiinduló részfa leveleinek a száma. Később használni fogom az $l(\mu(F))$ jelölést, ami egy $\mu(F)$ méretű probléma leveleinek a számát jelöli. Továbbá legyen $\tau(\Phi(F))$ a

$$\sum_{i=1}^{p} x^{-\Phi(F)_i} = 1$$

egyenlet egyetlen pozitív gyöke, ezt nevezzük *elágazási faktornak*. Ez a τ függvény alkalmazható tetszőleges elágazási vektorra. Ekkor a T fában lévő összes elágazásra kapott τ értékek közül a legnagyobbra:

$$\tau_{max}(\mu, T) \coloneqq \max_{F\in T}(\tau(\Phi(F))),$$

teljesül a [14] cikk 8.2-es lemmája alapján a következő felsőbecslés a levelek számára:

$$l(T) \leq \tau_{max}(\mu, T)^n.$$

Összefoglalva tehát egy Φ távolságfüggvény definiálásával a φ elágazásokra, megkapható egy felső korlát a futásidőre. Ami nem más, mint a T fa által adott $\sum_{i=1}^{p} x^{-\Phi(F)_i} = 1$ egyenletrendszerek legnagyobb gyöke, amit jelöljön λ. Szemléletesen, minél nagyobb $\Phi(F, F_i)$, annál jobban csökkenti a φ_i elágazás a probléma komplexitását. A cél tehát egy olyan Φ választása, aminek a lehető legkisebb értéke is nagy. Ezt formalizálva $\Phi(F) = (\Phi(F, F_i))_{i\in[p]}$ vektor dominálja $\Phi'(F)$ elágazási vektort, ha $\Phi(F) \leq \Phi'(F)$, vagyis $\Phi(F)$ komponensenként nem nagyobb, mint $\Phi'(F)$. Ekkor $\Phi(F)$ legalább akkora futásidőt eredményez, mint $\Phi'(F)$. Tehát a futásidő vizsgálásakor elég megfigyelni egy domináló vektorhalmazt. Hasonló okokból, amikor kicserélünk egy elágazási vektort, egy őt dominálo vektorra, akkor egy durva felső becslést kapunk a futásidőre. Ezeknek a tulajdonságoknak fontos szerepe lesz a továbbiakban.

Mint fent említettem, általában a Branch & Reduce algoritmusokban a μ mértéknek egyszerűen azt választják, amivel aztán kifejezik a futásidőt. De bármilyen, akár bonyolult, μ' mérték választható, amire teljesül $\mu' \leq f(\mu)$ valamilyen ismert f függvényre. Ezáltal elérhető egy $O^*(\lambda^{\mu'(F)}) = O^*(\lambda^{f(\mu)(F)})$ időkorlát (ahol F a kiindulási probléma), amiben $f(\mu)$ a kívánt formában van. Vagyis μ jó meghatározása rendkívül fontos. A könnyebb megértés érdekében itt van egy példa. Nézzük a minimális domináló csúcshalmaz problémát, ahol a legkisebb k egész számot kell meghatározni, amire létezik k darab csúcs egy gráfban úgy, hogy a maradék csúcsok mindegyikének legyen szomszédja a k kiválasztott csúcs között. Ennek az NP-nehéz problémának egy sztenderd analízisében $G = (V, E)$ esetén $\mu(G) = |V|$. Nyilván a kívánt futásidő alakja $O^*(2^{\alpha|V|}) = O^*(2^{\alpha\cdot\mu(G)})$. De adható tetszőleges μ' mérték, amire $\mu'(G) = f(\mu(G))$, hiszen ekkor $O^*(2^{\mu'(G)}) = O^*(2^{f(\mu(G))}) = O^*(2^{\alpha\cdot\mu(G)})$.

A λ kiszámítása. Már szó volt a jó algoritmusok és a jó mértékek fontosságáról, azonban ugyancsak fontos egy gyors és optimális, vagy közel optimális módszer a λ kiszámítására, hiszen ez határozza meg az algoritmus futásidejét.

Tegyük fel, hogy adottak a $\Phi(F_1), \Phi(F_2), \ldots, \Phi(F_t)$ különböző elágazási vektorok az algoritmus összes elágazási pontjához és a hozzá tartozó μ. A kezdeti probléma mértéke legyen k. Ekkor tetszőleges λ kielégítő megoldás egy felső korlátot ad a futásidőre (bár nem feltétlenül jót). Egy λ kielégítő megoldás, ha a következő

egyenlőtlenségeket teljesíti:

$$1 \geq \sum_{i=1}^{p_j} \lambda^{-\Phi(F_j)}, \quad \forall j = 1, 2, \ldots, t. \tag{2.1}$$

Tehát nem kell az egyenlőséget megkövetelni, az enyhített feltételeket teljesítő λ már felső becslést ad. Ez egy nagyon fontos tulajdonság, innentől ezt az egyenlőtlenséget fogom vizsgálni. Rögzített kitevők mellett ez egy kvázi-konvex programozási feladat [6], ezért létezik egy algoritmus, ami ezt kihasználva hasonló problémákat old meg [7]. Azonban a μ mérték, amit a későbbi algoritmusokban súlyozások fognak meghatározni, ugyancsak optimalizálandó bizonyos feltételek mellett. Ezért ilyen oldalról nehéz megközelíteni a problémát, de nem lehetetlen ([7]). Emellett a jó megközelítés mellett - ami a kvázikonvex megoldók miatt kissé időigényes - elképzelhető egy teljes randomizálás. Ezt a módszert fogom bemutatni és elemezni a következő részben.

2.2. Random lokális keresés

Minden NP-nehéz feladatra van egy triviális algoritmus . Ezért egy bizonyos λ és $\mu(F) = n$ választással egy helyes megoldást kapunk (ami kielégíti az összes egyenlőtlenséget). A módszer lényege, hogy alkalmas paraméterekből és λ-ból kiindulva, véletlenszerűen változtatva λ-t és μ-t (vagyis a benne lévő súlyokat) a $[-d, d]$ intervallumból egyenletesen választott értékkel, kapunk egy új λ-át. Ha ez kisebb, akkor megtartjuk, és feljegyezzük a hozzá tartozó paramétereket is. Ezt ismételve, ha sokáig nem érünk el javítást, csökkentjük d értékét. Ezt a lépést addig iterálva, amíg d elég kicsi nem lesz, eljuthatunk egy lokális minimumhoz . Az algoritmus alapjának pszeudokódja az 1 algoritmus . Mivel ez egy lokális keresés, ezért előfordulhat, hogy a kapott megoldás egyáltalán nem optimális. A módszer paramétereit állítva megvizsgáltam az egyes esetek hatékonyságát.

Formalizálás. Tegyük fel, hogy a μ mérték m paramétert használ. Azaz tetszőleges F_j állapotra $\mu(F_j) = p_1(F_j) + p_2(F_j) + \cdots + p_m(F_j)$. Ebből $\Phi(F_j)$ – vagyis a mértékek különbsége – is hasonlóan fölírható. Legyen adott egy $\mathcal{F}$ feltételrendszer, amiben tetszőleges feltételek lehetnek a p paraméterekre. A feladat tehát a következő. Minimalizáljuk λ-át, és határozzuk meg a $p_j,\ j = 1, \ldots, m$ paramétereket úgy, hogy a paraméterek teljesítsék $\mathcal{F}$-et és teljesülnek a 2.1 egyenletek. Mivel a mértéknek kisebbnek kell lennie, mint a sztenderd mérték, ezért feltehetjük, hogy a gráfbeli feladatok esetében minden p_j értéke egy csúcsra nézve legfeljebb 1.

2.2.1. Programok

Minden programot pythonban írtam, ami itt megtalálható: GitHub - notebook. Minden itt említett programot a későbbiekben leírt problémákra, és az ott használt algoritmusokkal futtattam. A kapott ábrák is ezen futtatások alapján készültek. A véletlenszerűség miatt, ahelyett, hogy egy programot nagyon sok ideig futtatnék, inkább lefuttatom röviden, de sokszor, és nem mindig a kiinduló értékből, hanem az előző futtatást folytatva, az ott kapott értékből. Azaz a befejezett állapotból nagy lépésközzel próbál továbbhaladni a program, ezzel több lehetőséget végigpróbálva.

1. Algorithm Alap algoritmus

```
1:  function PERTURB(parameters, d)
2:      for i in parameters do
3:          i = i + random(−d, d)
4:      end for
5:      return parameters
6:  end function
7:  function RANDOMSEARCH(parameters, d, eps, delta, maxcnt)
8:      while d < eps do
9:          newparameters := PERTURB(parameters, d)
10:         if newparameters teljesíti F-et then
11:             parameters = newparameters
12:             cnt = 0
13:         else
14:             cnt+= 1
15:         end if
16:         if cnt > maxcnt then
17:             d = d · delta
18:             cnt = 0
19:         end if
20:     end while
21:     return parameters
22: end function
```

0. Program. Minden problémánál adott egy triviális megoldás, ami egy triviális exponenciális algoritmusból adódik, amiben minden lehetőséget végigpróbálunk. Az alapötlet, ahogy fönt írtam, mindennek az iterációnkénti randomizálása. Ez a módszer azonban egy problémába ütközik. A kiindulási értékek, amiket a triviális megoldásból kapunk, az $\mathcal{F}$ feltételek felső határán vannak, hiszen a standard mértéket használják. Vagyis előfordulhat - ha nem tudunk jelentősen jobb megoldást elérni a triviálisnál -, hogy a program nem tud elindulni, mert az esetek nagy részében van olyan paraméter, ami véletlenül növekedne, főleg a λ, ami az egyenletekben hatványalapul szolgál és a keresett futásidőt adja. Erre lenne egy megoldás, ha $\mathcal{F}$-et enyhítjük és megengedünk

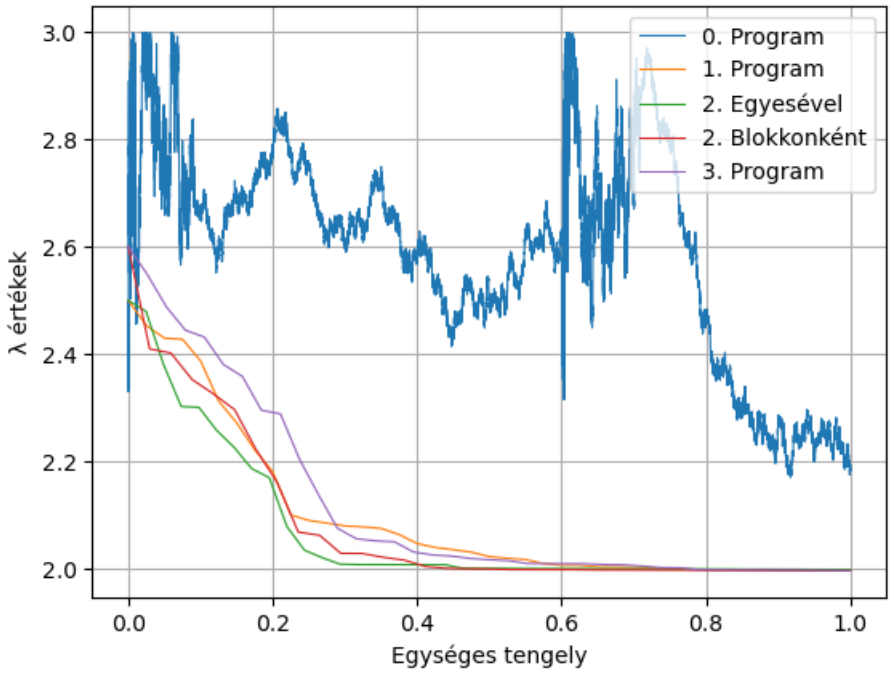
egynél nagyobb értékeket a paramétereknek (csúcsonként). Mivel azonban általában λ-nak az értéke csak kicsit változik, ez a program a triviálisnál rosszabb megoldást adna. Ahogy a randomizáció során megengedjük, hogy λ túlnőjön a triviális értéken, szinte biztos, hogy ezt meg is teszi, hiszen itt könnyebb találni olyan paramétereket, amik ezzel a λ-val helyes megoldást adnak. Tehát a program érdektelen lépéseket tesz a triviális határon túl, és nem talál jó megoldást (1).

1. Program. Erre megoldás, ha λ-át nem engedjük a triviális határértékén túlra, vagy ami még jobb, csak csökkentést engedünk meg (az 1 kód 3 sorában ha $i == \lambda$, akkor $i = i + random(-d, 0)$). Ez a program már jelentős javításra képes. Mivel λ csak csökkenhet, ha a λ-t a triviális értékről indítjuk, akkor előfordulhat, hogy nehezen indul el, hiszen a paraméterek még egyáltalán nincsenek beállítva. Ezért jobb nagyobb értékről indítani λ-át kihasználva a randomizáció elágazását. Viszont az enyhített feltételek melletti program itt nem ad jobb eredményt. Ennek az oka az lehet, hogy sokszor a már meglévő eredményből indulunk ki, vagyis ha egyszer elszakad a program a küszöbértékektől, már nem számít, hogy onnan indul.

2. Program. A következő ötlet külön-külön változtatni a paraméterek értékeit. Az első opció minden paramétert a λ nélkül változtatni; ugyanazzal a módszerrel, mint az első programban. Megpróbálni javítani az egyes paramétereket, és csak utána csökkenteni λ-t. Amikor az egyik paraméter elakad, vesszük a következőt. Amikor mindegyik elakad, kisebb lépésközzel újra elindítjuk a folyamatot. Amikor ez a külső lépésköz is elér egy kicsi értéket, megállunk. Ez az ötlet önmagában nem állja meg a helyét. Mivel mindig van összefüggés a paraméterek között, nem tud igazán hatékonnyá válni. Hasonlóan, ha egyszerre kettő paramétert változtat a program, akkor sem ér el jó eredményt. Azonban, főleg sok paraméteres esetben, a hasonló tulajdonságú paramétereket tudja optimalizálni, és ezzel elérni kisebb λ-t. Viszont előfordulhat, hogy az egyik ilyen paramétertömb optimalizálásával a program elér egy olyan lokális minimumot, ahol a többi paramétert már nem lehet változtatni, vagy nem lehet eléggé. Ezért érdemes bevezetni egy szabály enyhítést a sorrendekre vonatkozóan. Vagy ezt is lehet randomizálni. Fontos azonban, hogy a paraméterek kezdőértékei ne a megengedett felső határról induljanak. Ez gond lehet abban az esetben, ha nehéz jó paramétereket találni. Mindegyik esetben az alap kódnak (1) a PERTURB része változik. A 2. sorban ahelyett, hogy minden paraméteren végigmennénk, csak a kívánt specializáció szerint megyünk, és a RANDOMSEARCH függvényben (9. sor) a többféle PERTURB között váltogatunk a megfelelő feltételek szerint.

3. Program. A kész programban ahelyett, hogy korlátoznám a paraméterek változását, módosítom az intervallumot, amiből egyenletesen választottam a paraméterek megváltozásának értékét két iteráció között. Vagyis nem $(-d, d)$ közül sorsolok érté-

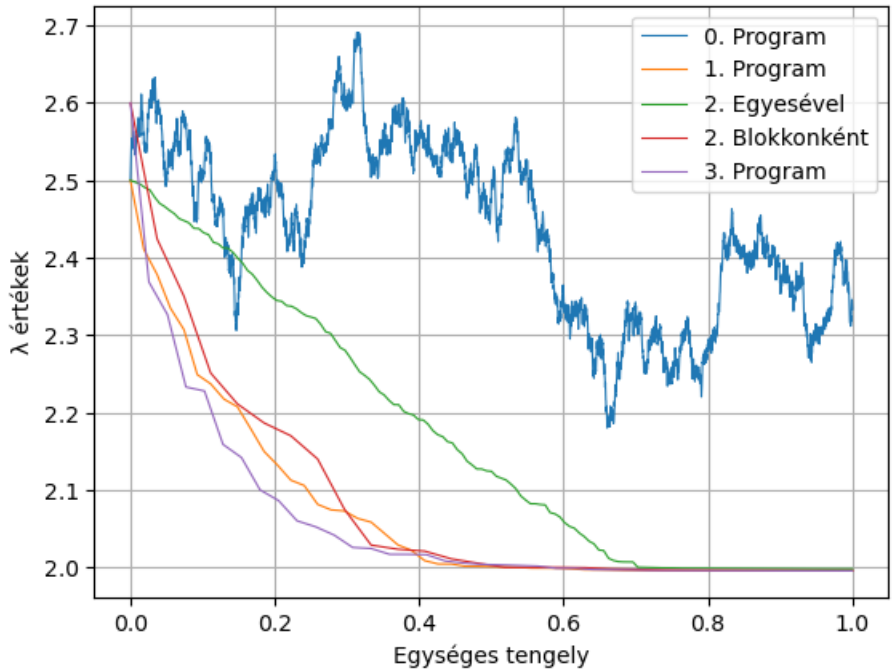
ket, hanem (d_1, d_2) közül. Ezeket az értékeket d-ből és kezdetben 0 várható értékű és d szórású normális eloszlásból számolom ki. Majd menet közben változtatom a várható értéket aszerint, hogy milyen irányba léptünk az előző iterációkban az egyes paraméterekkel. Ez a program bizonyult a legpontosabbnak, igaz hasonló eredményeket ért el, mint az 1. program. Ezen kívül az itt leírt konkrét program előtt futtattam egy hiperparaméter-optimalizálást, amivel beállítottam λ kezdőértékét és az összes többi változót: $d, eps, delta, maxcnt$. Ehhez pszeudokód a 2. Algoritmus. Már ez az előzetes futtatás is kiad egy megoldást (2.3 ábra), ami az általam kipróbált néhány problémára pár másodperc alatt megközelítette, vagy elérte a cikkekben nagyjából tízezrelékig megadott értékeket, amik a [7] algoritmust használták, ami az iterációnkénti kvázikonvex program megoldás miatt lassabb. A 2.2 és 2.3 ábrákon összehasonlítottam a leírt programokat. Látható, hogy a 0 kivételével mindegyik közel jó eredményt ad. Ezek közül kettőt még egy másik problémára is összehasonlítottam, itt már hosszabb futtatásokkal (ld.: 2.4 és 2.5).



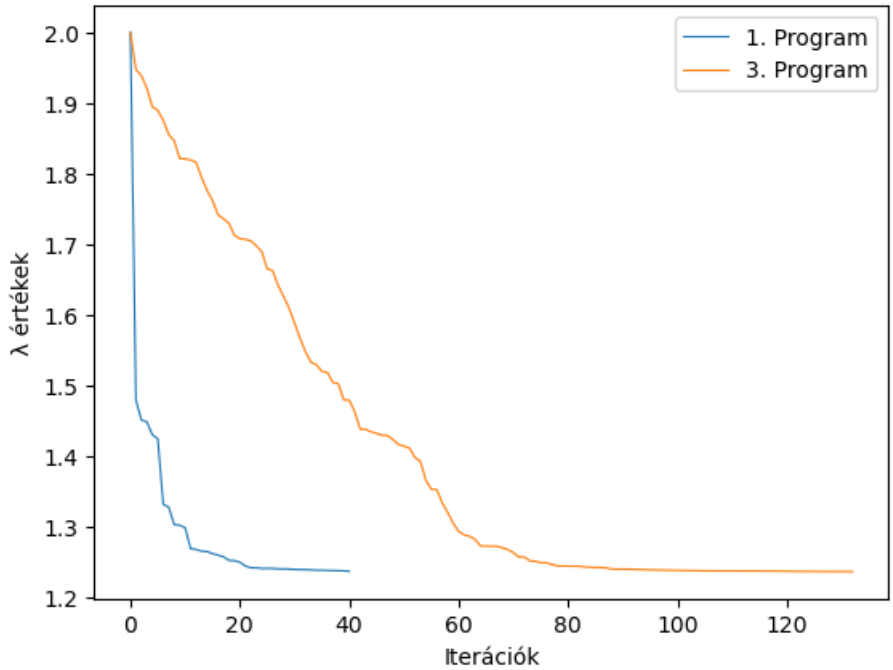
2.2. ábra. A programok eredménye a 3.3 feladatra iterációban skálázott x tengellyel ábrázolva.

2. Algorithm Kész algoritmus

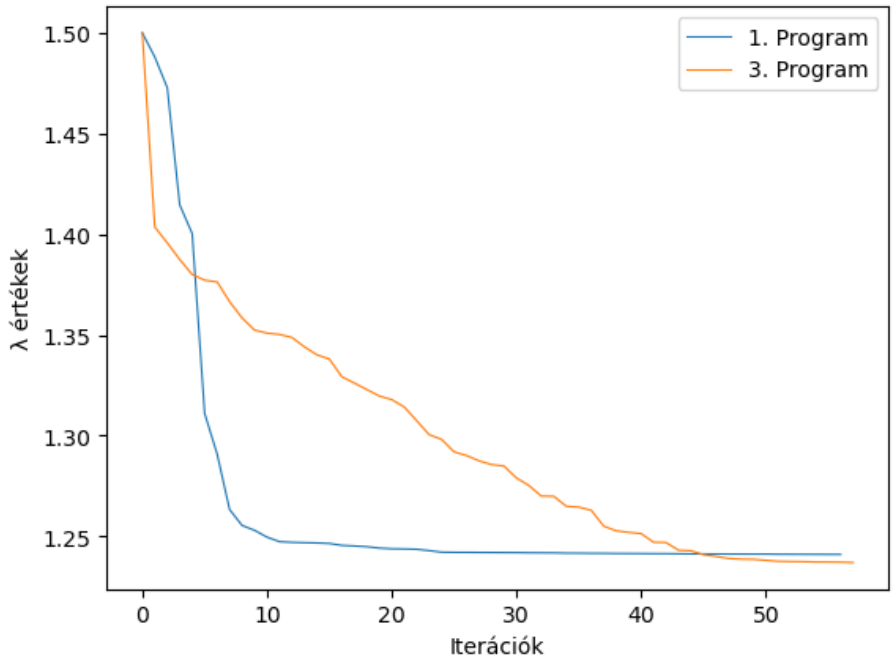
```
1: function PERTURB(parameters, d, balance)
2:     λ = λ + random(parameters)
3:     for i in parameters \ {λ} do
4:         dif[i] := random(−d + rnorm(balance[i], d), d + rnorm(balance[i], d))
5:         i = i + dif
6:     end for
7:     return parameters, dif
8: end function
9: function RANDOMSEARCH(parameters, balance, hyperpars)
10:    while d < eps do
11:        newparameters := PERTURB(parameters, d, balance)[0]
12:        if newparameters teljesíti F-et then
13:            parameters = newparameters
14:            cnt = 0
15:            balance+=PERTURB(parameters, d, balance)[1]
16:        else
17:            cnt+= 1
18:        end if
19:        if cnt > maxcnt then
20:            d = d · delta
21:            cnt = 0
22:        end if
23:    end while
24:    return parameters
25: end function
26: function PRETUNE(hyperpars, parameters, balance)
27:    for combination in product(hyperpars) do
28:        newhyper =RANDOMSEARCH(parameters, balance, combination)
29:        if λ in newhyper ≤ λ in hyperpars then
30:            hyperpars := newhyper
31:        end if
32:    end for
33:    return hyperpars
34: end function
```



2.3. ábra. A programok eredménye a 3.3 feladatra az optimális paraméterekkel, iterációban skálázott x tengellyel ábrázolva.



2.4. ábra. Teljes futtatások. Az első program fél percig, a harmadik egy percig futott ugyanannyi ismétléssel a 4.1 problémára. A harmadik program egy kicsit jobb eredményt ért el.



2.5. ábra. Az előző futtatás $\lambda = 1.5$ értékről indítva. Itt a 3. program volt gyorsabb.

3. Directed Multiway Cut

3.1. Problémakör

A vágások a gráfelméletben széles körben elterjedtek az alkalmazhatóságuknak köszönhetően. A gráf két pontja közötti minimális vágás megtalálására Ford és Fulkerson adott polinomiális idejű algoritmust. Ez a sokat használt max-flow min-cut - maximális folyam minimális vágás - algoritmus adott okot az általánosításra. A probléma szépségének ellenére a teljesen általános esetet nézve, az évek során kialakult többféle megközelítés mindegyikéről belátták, hogy NP-nehéz. Sőt, némelyikhez még közelítő algoritmust is nehéz adni. A feladat komplexitásából kifolyólag, mivel gyors algoritmusra nincs remény, a cél az exponenciális futásidő leredukálása, és mint látni fogjuk, ehhez a Measure & Conquer kapóra jön.

Ilyen általánosítás a minimális Multicut probléma, ahol adott l darab tetszőleges számú pontpár $\{s_i, t_i\}$ egy irányítatlan gráfban $G(V, E)$ és egy nemnegatív élsúlyozás. A feladat meghatározni egy minimális élhalmazt, amit elvéve a gráfból, a megmaradó gráfban nem marad út s_i és t_i között $i = 1, \ldots, l$-re. Ehhez hasonló a k-Multicut, ahol elegendő (rögzített k-ra) egy olyan minimális súlyú vágást találni, ami legalább k pontpárt választ szét az összes l-ből. A Multicut egyik fontos speciális esetében egy ponthalmaz adott, és olyan vágást keresünk, ami ezek közül bármely két pontot elválasztja. Leggyakrabban ezt a problémát hívják Multiway Cut-nak, azonban ebben a dolgozatban ennek egy másik változatára fogok hivatkozni ezen a néven. Ugyanis nézhetjük azt a problémát is, amikor élek helyett csúcsokat keresünk, és azok eltávolításával akarjuk elválasztani a kijelölt csúcsokat - terminálokat. Mindkét esetben $k \geq 3$ terminálra a feladat NP-nehéz. Érdemes megjegyezni, hogy a csúcsok törlésénél a terminálok nem lehetnek szomszédosak, hiszen ekkor a feladat nem megoldható. Fontos megjegyezni, hogy ebben a feladatban nem engedjük meg a terminálok törlését. Természetesen az itt említett feladatoknak nézhetjük az irányított változatát. Ez azonban még nagyobb kihívást jelent; a Directed Multiway Cut már $k \geq 2$ terminálra NP-nehéz.

Ebben a fejezetben a *Directed Multiway Cut* problémáról lesz szó, ami tehát a következő. Adott egy $G = (V, E)$ irányított gráf és egy $T \subseteq V$ terminálhalmaz. A cél egy minimális $S \subseteq V \setminus T$ halmaz megtalálása, amit ha kitörlünk a gráfból, a megmaradó gráfban nem lesz olyan irányított út, ami egy terminálból egy másik terminálba megy. Erre a problémára sokáig nem volt a triviálisnál jobb algoritmus, de 2024-ben Mingyu Xiao adott egy algoritmust, ami $O(1.9967^{|V|})$ idő alatt megoldást ad. A következőkben ezt az algoritmust vizsgálva, mutatok egy javítást, amivel kicsivel jobb futásidő elérhető. A jobb futásidőt új súlyok bevezetésével érem el. Ez persze felveti azt a kérdést, hogy ilyen vagy ehhez hasonló adatcsökkentési szabályokkal, mint a *Measure and Conoquer*, meddig csökkenthető egy pontos, NP-nehéz feladatot megoldó algoritmus futásideje, illetve milyen már kitalált algoritmusok adhatnak jobb futásidőt.

Directed Multiway Cut
Input: $G = (V, E)$ irányított gráf, valamint egy $T = \{t_1, t_2, \ldots, t_l\}$ terminálhalmaz, ahol $T \subseteq V$, és egy egész szám k.
Output: Létezik-e olyan $S \subseteq V \setminus T$ csúcsokból álló halmaz, hogy $|S| \leq k$ és az így kapott $G - S$ gráfban nincs út bármely két terminál között T-ben?

Jelölések

A feladatból kifolyólag $G(V, E)$ jelöljön irányított gráfot. Minden él E-ben egy V elemeiből álló rendezett pár. Legyen a csúcsok száma $|V| = n$. Egy $v \in V$ csúcs esetén a kiszomszédokat jelölje $N^+(v) \coloneqq \{u \in V : vu \in E\}$ és a beszomszédokat ennek megfelelően $N^-(v)$. Egy $V' \subset V$ csúcshalmaz esetén a jelölés legyen $N^+(V') = \bigcup_{v \in V} N^+(v) \setminus V'$ és $N^-(V') = \bigcup_{v \in V} N^-(v) \setminus V'$. Legyen továbbá $U \subset V$ esetén $N^{\pm}_U(V') = N^{\pm}(V') \cap U$.

3.2. Mingyu Xiao Algoritmusa

Először felszínesen leírom a [17]-ban használt algoritmust.

Keretrenszer. Az algoritmus során végig fenntartunk egy A részhalmazát a csúcsoknak, amire a következő alaptulajdonságok érvényesülnek:

- A-ban potosan egy terminál van, jelölje ezt t;

- t-ből minden másik csúcs elérhető A-ban;
- az A-ban lévő csúcsok nem lehetnek benne az S megoldáshalmazban.

Az A halmaz meghatároz két másik halmazt a következő módon: $B = N^+(A)$ a kiszomszédok halmaza és $R = G \setminus (A \cup B)$ a megmaradó csúcsok halmaza. Az algoritmus a legtöbb lépésben vagy kitöröl egy csúcsot a gráfból, betéve azt a megoldáshalmazba, vagy "megbélyegzi", mint nem megoldás. Az algoritmus lényege, hogy ily módon a csúcsokat R-ből B-be vigye, majd B-ből A-ba, és miután $N_R^+(B) = \emptyset$, vagy kitörli $A \cup B$-t a gráfból, ha $N^-(A \cup B) = \emptyset$, vagy megfordítja a gráf éleit, ha $N^-(A \cup B) \neq \emptyset$. Ez a standard súlyozással nem lenne jobb a triviális $O(2^n)$ futásidőnél. Ezért a Measure & Conquer módszerrel a csúcsokhoz a tulajdonságaiktól függően, különböző súlyokat rendelünk. Ezeket, az egyszerűség kedvéért fekete, fehér, kék és piros színekkel fogjuk jelölni. Illetve a színeken felül az algoritmus bizonyos lépésekben megjelöl csúcsokat, ezzel ugyancsak módosítva azok súlyát. Így a csúcsok a következő értékekkel a 3.6 táblázat alapján kapnak súlyokat: $\alpha = 0.958234, \beta = 0.943555, \gamma = 0.901789$. Nagyon fontos a színek jelentése. Feketére azokat a csúcsokat színezzük, amiket kizártunk a megoldáshalmazból; az A-ban és a T-ben lévő csúcsok mind feketék; a B-ben lévő csúcsok lehetnek kék és piros színűek aszerint, hogy van *általánosított kiszomszédjuk*, vagy nincs.

Jelölje w a gráfban lévő összes csúcs súlyának összegét, $w \leq n$. Továbbá jelölje $l(w)$ egy w súlyú gráf által a keletkezhető legtöbb levél számát. Minden egyes elágazási művelethez egy rekurzív összefüggést fogunk felépíteni ezekkel az értékekkel, amikből aztán kiszámolható az alágazási faktor. Ha az algoritmusban előforduló összes elágazási faktor közül a legnagyobb λ, akkor a keresési fa mérete legfeljebb $O(\lambda^w)$.

	fekete	piros	kék	fehér
jelölt	0	γ	γ	β
jelöletlen	0	β	α	1

3.6. ábra. Az eredeti algoritmus színeihez tartozó értékek.

Általánosított kiszomszéd. Egy u csúcs általánosított kiszomszédja v-nek, ha létezik egy $u \to v$ út, amiben u-n és v-n kívül minden csúcs fekete. A v általánosított kiszomszédainak jele $GN^+(v)$. Ha egy V halmazra használjuk, akkor ismét $GN^+(V) = \bigcup_{v \in V} GN^+(v)$ értjük. Illetve gyakran fog előjönni ugyanennek egy halmazra való megszorítása $GN_R^+(v) = R \cap GN^+(v)$.

Lépések. Mikor egy csúcsot kizárunk a megoldáshalmazból, automatikusan feketére színezzük. Az algoritmus minden művelete után, ha az ellenkezőjét nem állítjuk, $A \cup B$-re alkalmazzuk a *Színezési Szabályt*: Minden A-ba kerülő csúcsot feketére színezünk; pirosra színezünk egy $v \in B$ csúcsot, ha $GN_R^+(v) = \emptyset$ és kékre, ha $GN_R^+(v) \neq \emptyset$. Mielőtt végrehajtanánk bármelyik fő lépést, a következő csökkentési szabályokat alkalmazzuk.

- Ha A üres, vegyünk be egy tetszőleges $t \in T$ terminált A-ba.
- Ha van egy fekete csúcs B-ben, azt azonnal áttesszük A-ba.
- Ha egy csúcsból nem érhető el egyik terminál sem, vagy egy csúcs nem érhető el egyik terminálból sem, akkor töröljük a gráfból.
- Ha $N^+(A \cup B) = N^-(A \cup B) = \emptyset$, akkor töröljük $A \cup B$-t a gráfból.

Kilenc lépése van az algoritmusnak. Először A csak egy terminálpontot tartalmaz, $B = N^+(t)$ és $R = V \setminus (A \cup B)$. Az algoritmus addig hajtja végre a lépéseket, amíg $k < 0$ vagy az üres gráfhoz jutunk. Ha nincs megjelölt csúcs $B \cup R$-ben, akkor végrehajtjuk az 1-4. lépéseket. Ellenkező esetben az 5-9. lépéseket. Most bevezetem a lépéseket, és odaírom az eredeti súlyozással kapott elágazási faktorokat az összehasonlíthatóság kedvéért.

1. Lépés. *Ha van egy $v \in B$, csúcs, amire $|GN_R^+(v)| \geq 3$, akkor elágazunk kétfelé: vagy kitöröljük v-t a gráfból, és k-t csökkentjük* 1*-el, vagy feketére színezzük v-t.*

Az elágazás értéke nem nagyobb, mint 1.9736.

2. Lépés. *Ha van egy $v \in B$, csúcs, amire $|GN_R^+(v)| = 1$, akkor feltesszük, hogy $GN_R^+(v) = \{u\}$ és elágazunk kétfelé: vagy kitöröljük u-t a gráfból és k-t csökkentjük* 1*-el, vagy feketére színezzük u-t.*

Az elágazás értéke nem nagyobb, mint 1.9900.

3. Lépés. *Ha van egy $v \in B$, csúcs, amire $|GN_R^+(v)| = 2$, akkor feltesszük, hogy $GN_R^+(v) = \{u_1, u_2\}$ és elágazunk háromfelé: vagy feketére színezzük u_1-et, vagy kitöröljük u_1-et a gráfból, feketére színezzük u_2-t és csökkentjük k-t* 1*-el, vagy kitöröljük u_1-et és u_2-t is, és csökkentjük k értékét kettővel.*

Az elágazás értéke nem nagyobb, mint 1.9967.

4. Lépés. *Ha* $GN_R^+(B) = \emptyset$ *és* $N^-(A \cup B) \neq \emptyset$*, akkor* B *összes csúcsát fehérre színezzük, megjelöljük, megfordítjuk a gráf összes élét és frissítjük* B*-t, úgy, hogy* $B = N^+(A)$ *legyen az új gráfban.*

5. Lépés. *Ha van egy megjelölt csúcs* $v \in B$*, amire* $|GN_R^+(v)| \leq 1$ *teljesül, akkor* v*-t tegyük át* A*-ba.*

6. Lépés. *Ha van egy megjelölt* $v \in B$ *csúcs, amire* $|GN_R^+(v)| = 2$*, akkor tegyük fel, hogy* $GN_R^+(v) = \{u_1, u_2\}$ *és elágazunk kétfelé: vagy átesszük* v*-t* A*-ba, vagy kitöröljük* v*-t, csökkentjük eggyel* k*-t és* u_1*-et,* u_2*-t feketére színezzük.*

Ennek az elágazási értéke nem nagyobb, mint 1.5166.

7. Lépés. *Ha van egy megjelölt* $v \in B$ *csúcs, amire* $|GN_R^+(v)| = 3$*, akkor tegyük fel, hogy* $GN_R^+(v) = \{u_1, u_2, u_3\}$ *és elágazunk három alrészre: vagy átesszük* v*-t* A*-ba, vagy kitöröljük* v*-t, csökkentjük eggyel* k*-t és* u_1*-et feketére színezzük, vagy kitöröljük* v*-t és* u_1*-et a gráfból, csökkentjük* k*-t kettővel és* u_2*-t,* u_3*-at feketére színezzük.*

Ennek az elágazási értéke nem nagyobb, mint 1.8453.

8. Lépés. *Ha van egy megjelölt* $v \in B$ *csúcs, amire* $|GN_R^+(v)| = 4$*, akkor tegyük fel, hogy* $GN_R^+(v) = \{u_1, u_2, u_3, u_4\}$ *és elágazunk négy alrészre: vagy átesszük* v*-t* A*-ba, vagy kitöröljük* v*-t, csökkentjük eggyel* k*-t és* u_1*-et feketére színezzük, vagy kitöröljük* v*-t és* u_1*-et a gráfból, csökkentjük* k*-t kettővel és* u_2*-t feketére színezzük, vagy kitöröljük* v*-t,* u_1*-et és* u_2*-t a gráfból, csökkentjük* k*-t hárommal és feketére színezzük* u_3*-at és* u_4*-et.*

Ennek az elágazási értéke nem nagyobb, mint 1.9967.

9. Lépés. *Ha van egy megjelölt* $v \in B$ *csúcs, amire* $|GN_R^+(v)| \geq 5$*, akkor elágazunk kétfelé: vagy kitöröljük* v*-t és csökkentjük eggyel* k*-t, vagy átteszük* v*-t* A*-ba.*

Ennek az elágazási értéke nem nagyobb, mint 1.9967.

Ezeknek a lépéseknek a bizonyítását nem írom le, mert bizonyításuk nem fontos a javításom szempontjából. Elég annyit megfigyelni, hogy ha van megjelölt csúcs $B \cup R$-ben, akkor mind az 5-9. lépés ezek számát csökkenti. Vagyis miután elfogynak a jelölt csúcsok $B \cup R$-ben visszatér az algoritmus az 1-4. lépésekhez. Tehát a két része az algoritmusnak "különálló", és külön-külön javítható. Az is megfigyelhető, hogy a lassú lépések a 3., 8., 9. Ezeket figyelembe véve, és azt, hogy az 1., 2., 3. lépések egymást kizáró esetekben alkalmazandók, megpróbálhatjuk gyorsítani a 3. lépést úgy, hogy amikor végrehajtjuk, kicsit nagyobbat csökkentünk w-n, viszont

amikor az 1. és 2. lépéseket hajtjuk végre, ellensúlyozzuk ezeket a csökkentéseket egy kis extra munkával (azaz kisebbre véve w csökkenését). Ez az ötlet hasonló az amortizált analízis ötletéhez, ahol - bár legrosszabb esetben lassú az algoritmusunk - beláthatjuk, hogy az algoritmus nagy részében könnyű lépéseket teszünk, és ezek a könnyű lépések ellensúlyozzák a nehezet.

3.3. Javított algoritmus

Ahogy már említettem, az algoritmus első felében a harmadik lépésnek volt a legnagyobb az elágazási faktora, ezért ezen szeretnék csökkenteni. Ehhez három részre bontom ezt a lépést és bevezetek két új változót $\nu = 0.016377, \mu = 0.004020$, a többi érték pedig $\alpha = 0.958218, \beta = 0.944115, \gamma = 0.902332$ kicsit változik. Illetve az átláthatóság kedvéért bevezetek egy új $C \subset R$ halmazt úgy, hogy $C = GN_R^+(B)$. Ekkor definiálható az $R' = V \setminus (A \cup B \cup C)$ megmaradó csúcsok halmaza. Továbbá felteszem, hogy a **3.1.**, **3.2.**, **3.3.** lépések végrehajtása előtt minden $v \in B$ csúcsra, amire $|GN_R^+(v)| = \{u_1, u_2\}$, teljesül, hogy v súlya α, u_1 és u_2 súlya pedig 1, ahogy azt az eredeti algoritmusban elvártuk. Ezt később belátom, hogy továbbra is igaz marad. A harmadik lépését az algoritmusnak tehát lecserélem a következő három lépésre.

10. Tétel. *Az Irányított Multiway Cut probléma megoldható $O(1.9959^n)$ időben.*

3.1. Lépés. *Ha van egy $v \in B$ csúcs, amire $|GN_R^+(v)| = 2$ teljesül, ahol tegyük fel, hogy $GN_R^+(v) = \{u_1, u_2\}$, és $|GN_{R'}^+(u_1)| = 0$, akkor három alrészre osztjuk a feladatot: u_1-et feketére színezzük, és u_2 súlyát csökkentjük μ-vel; u_1-et kitöröljük a gráfból, és u_2-t feketére színezzük; u_1-et és u_2-t is kitöröljük a gráfból.*

Bizonyítás. Ennek a lépésnek a helyessége következik abból, hogy az eredeti algoritmus lépése helyes volt, és ez a lépés szűkített feltételek mellett, de ugyanazokra az elágazásokra bomlik. A súlyokat tekintve csak az első ágban van változás az eredeti lépéshez képest, ahol a feketére színezés mellett egy újabb μ-vel csökken az összsúly. Így a következő relációt írhatjuk fel:

$$l(w) \leq l(w - (1 + \mu)) + l(w - 2) + l(w - (2 + \alpha - \beta)),$$

így az elágazás faktora nem nagyobb, mint 1.9950. •

3.2. Lépés. *Ha van egy $v \in B$ csúcs, amire $|GN_R^+(v)| = 2$ teljesül, ahol tegyük fel, hogy $GN_R^+(v) = \{u_1, u_2\}$, $|GN_{R'}^+(u_1)| = 1$ és $GN_{R'}^+(u_2) = \{z\}$, akkor három alrészre*

osztjuk a feladatot: u_1-et feketére színezzük; u_1-et kitöröljük a gráfból, és u_2-t feketére színezzük, és z értéket csökkentjük μ-vel; u_1-et és u_2-t is kitöröljük a gráfból.

Bizonyítás. Ennek a helyességét ugyanúgy bizonyíthatjuk, mint az előzőt. A súlyokat tekintve most a második ágban lesz nagyobb a csökkenés az eredetihez képest. Mégpedig a fekete színezés és törlés 2-je helyett $2 + \mu$-vel fog csökkenni az összsúly, hiszen z egy fehér csúcs volt. Tehát a következő egyenlőtlenséget kapjuk:

$$l(w) \leq l(w-1) + l(w-(2+\mu)) + l(w-(2+\alpha-\beta)),$$

amiből az elágazási faktor nem nagyobb, mint 1.9959. •

3.3. Lépés. *Ha van egy $v \in B$ csúcs, amire $|GN_R^+(v)| = 2$ teljesül, ahol tegyük fel, hogy $GN_R^+(v) = \{u_1, u_2\}$, $|GN_{R'}^+(u_1)| \geq 2$, akkor három alrészre osztjuk a feladatot: u_1-et feketére színezzük, és v súlyát csökkentjük ν-vel; u_1-et kitöröljük a gráfból, és u_2-t feketére színezzük; u_1-et és u_2-t is kitöröljük a gráfból.*

Bizonyítás. Ez a lépés is helyes a fentiek miatt. A súlyokat tekintve az első ág változik az eredetihez képest. Ebben az ágban a csökkenés ν-vel nő, hiszen az eddig kék v csúcsot még csökkentjük ν-vel és $\nu < \alpha$ miatt v súlya pozitív marad. Az egyenlőtlenség

$$l(w) \leq l(w-(1+\nu)) + l(w-2) + l(w-(2+\alpha-\beta))$$

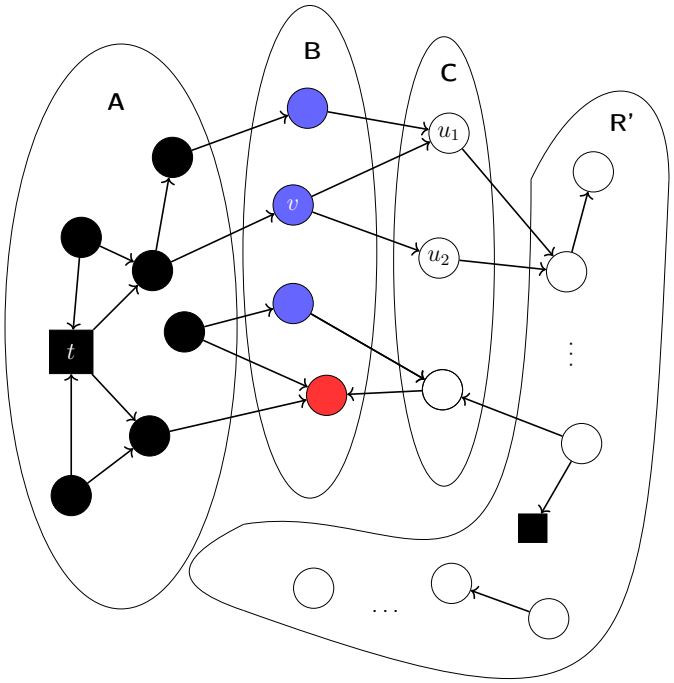
lesz, így az elágazási faktor nem több, mint 1.9894. •

Látható, hogy ez a három lépés helyettesíti az eredeti algoritmus 3. lépését, hiszen a v csúcs szomszédaira, ha $GN_R^+(v) = \{u_1, u_2\}$, akkor vagy $|GN_{R'}^+(u_1)| = 1$ és $|GN_{R'}^+(u_2)| = 1$ teljesül, vagy $|GN_{R'}^+(u_1)| \neq 1$, mivel u_1 és u_2 között nem teszünk különbséget, így $|GN_{R'}^+(u_1)| = 0$ vagy $|GN_{R'}^+(u_1)| \geq 2$.

Viszont ahhoz, hogy ez a módszer tényleg javítsa a futásidőt, feltettem a következő lemmát.

11. Lemma. *3.1, 3.2, 3.3 lépések végrehajtása előtt a $v \in B$ kiválasztott csúcsra, amire $|GN_R^+(v)| = \{u_1, u_2\}$, teljesül, hogy v súlya α; u_1 és u_2 súlyai pedig* 1.

Bizonyítás. Ennek a belátásához felhasználom azt, hogy az 1, 2, 3.1, 3.2 és 3.3 lépések tetszőleges sorrendben végezhetőek, szabadon választott csúcsokon. Ennek az oka, hogy mielőtt átvált az algoritmus az 5-9. lépésekre - mivel minden lépés legalább egy csúcsot eliminál - a B halmaz elemei mind pirosak lesznek a lépések



3.7. ábra. A gráf ábrázolása a színezett csúcsokkal. Ha ezen az ábrán a v csúcsra lép az algoritmus, akkor a 3.2. lépést hajtja végre.

sorrendjétől függetlenül. Nincs kitüntetett szerepe egyik lépésnek sem. Vagyis feltehetjük, hogy a 3.1 és a 3.2 lépések után az algoritmus mindig a 2 lépést, a 3.3 lépés után mindig az 1 lépést hajtja végre. Ez láthatóan mindig lehetséges, hiszen pont így változnak v kiszomszédai. Továbbá azt is feltehetjük, hogy az 1 és 2 lépéseket éppen a megváltozott súlyú csúcsokra alkalmazzuk. Ezzel elérve, hogy a μ-vel, vagy ν-vel történő változtatások azonnal eltűnjenek a gráfból. Részletesebben, tegyük fel, hogy az algoritmus során valamikor v-re végrehajtjuk valamelyik 3. lépést. Ekkor $|GN_R^+(v)| = 2$, és vagy a 3.1. és 3.2. lépések során egy szomszédja marad, aminek az értéke 1 helyett $1-\mu$ lesz, vagy a 3.3. lépés miatt v értéke α helyett $\alpha-\nu$ lesz, és legalább 3 szomszédja lesz. Az első eset után a 2. lépéssel v szomszédját vagy feketére színezzük, vagy töröljük a gráfból, de mindkét esetben 0 értéke lesz a csúcsnak. A második esetben az 1. lépéssel most magát v-t nullázzuk, hiszen vagy feketére színezzük, vagy töröljük. Azaz tényleg feltehetjük, hogy mielőtt az algoritmus valamelyik 3. lépést tenné, nem talál μ-vel, vagy ν-vel csökkentett csúcsot. •

Ezzel a trükkel tehát elértük, mivel $\mu, \nu > 0$, hogy a $|GN_R^+(v)| = 2$ esetet gyorsabban oldja meg az algoritmus. Ugyanakkor az 1 és 2 lépések a leírtak alapján a következőképpen romlanak:

1. Lépés. *Az elágazás mint fent (1)*

Bizonyítás. Mikor v-t töröljük, vagy feketére színezzük, annak az értéke legalább $\alpha - \nu$ az eredeti α helyett, vagyis a következőképpen módosul a súlyokra felírható változás:

$$l(w) \leq l(w - (\alpha - \nu)) + l(w - (3 - \nu - 2\alpha)),$$

aminek az értéke 1.9957 •

2. Lépés. *Az elágazás, mint fent (2)*

Bizonyítás. Mikor u_1-et töröljük, vagy színezzük feketére, annak az értéke legrosszabb esetben $1 - \mu$ az eredeti 1 helyett, vagyis a következőképpen módosul a súlyokra felírható összefüggés:

$$l(w) \leq l(w - (1 - \mu)) + l(w - (1 - \mu + \alpha - \beta)),$$

aminek az értéke 1.9959 •

Tehát beláttuk, hogy a megadott paraméterekkel az algoritmus első fele gyorsítható, és itt érdemes megemlíteni, hogy az eredeti cikkben megadott értékekkel az algoritmus 8 és 9 lépésének is az elágazási faktora a felső határon volt, viszont az új értékek ezen is javítottak, igaz, ugyancsak a felső határon vannak a faktorok (ld: 3.9 ábra).

	fekete	piros	kék	fehér
jelölt	0	γ	γ	β
jelöletlen	0	β	α	1

$\xi = \mu, \nu$ módosítás	fekete	piros	kék	fehér
jelölt	0	$\gamma - \xi$	$\gamma - \xi$	$\beta - \xi$
jelöletlen	0	$\beta - \xi$	$\alpha - \xi$	$1 - \xi$

3.8. ábra. A javított algoritmus csúcsainak értékei.

6. Lépés elágazási faktora: 1.5162
7. Lépés elágazási faktora: 1.8447
8. Lépés elágazási faktora: 1.9959
9. Lépés elágazási faktora: 1.9959

3.9. ábra. A nem változtatott lépések elágazási faktorai.

4. Alkalmazás

A Measure and Conquer fontos része az algoritmusban szereplő elágazási vektorokból kiszámolni az elágazási faktorokat, hiszen ezek adják a futásidőt. Vagyis fontos a λ meghatározásának a módja is. A 2.2 fejezetben adott módszert ezért megnéztem egy másik feladatra is, méghozzá pont a [8]-ben szereplő MSC algoritmusra. Ezzel az algoritmussal a probléma $O^*(2^{0,610n}) = O^*(1.5263^n)$ időben megoldható polinomiális térben.

4.1. Minimális dominál halmaz

Ez egy sokak által vizsgált , és széles körben használt probléma, amire [8] óta születtek közelítő és gyakorlati megoldások is. Újabb pontos algoritmus, hasonlóan a Measure and Conquer módszer használatával, csak rá 2 évre [16]-ben Bodlaendernek sikerült, aki egy $O(1.4969^n)$ idejű megoldást adott.

Az MDS probléma a következő. Adott egy $G(V, E)$ gráf. Az $U \subset V$ egy dominál halmaz, ha minden $v \in V$ csúcs vagy elem U-nak, vagy szomszédos egy U-beli csúccsal. A feladat megtalálni egy minimális U dominál részhalmazt. Ehhez a problémához hasonló az MSC, azaz minimális halmazfedés probléma. Itt adott egy U univerzum és egy $\mathcal{S}$ halmaz U részhalmazaival. $\mathcal{C} \subset \mathcal{S}$ egy halmazfedés, ha $\bigcup_{S \in \mathcal{C}} S = U$, ahol feltehető, hogy $\bigcup_{S \in \mathcal{S}} S = U$. A feladat egy tartalmazásra minimális $\mathcal{C}$ halmazfedést megtalálni. Az MDS könnyen redukálható egy MSC-re. Vegyünk a gráfnak megfelelő $|V| = n$ elemű univerzumot, ahol az elemek a csúcsokat reprezentálják és minden csúcshoz azt a részhalmazt, ami önmagából és a szomszédaiból áll. Ennek a megoldására adott [8] egy algoritmust, aminek az elágazási vektorai a következők:

$$l(w) \leq l(w - \Delta w_{OUT}(t)) + l(w - \Delta w_{IN}(t)) \tag{4.2}$$

ahol t egy éppen vizsgált $S \in \mathcal{S}$ halmaz lehetséges felépítései a következőképpen;

r_i jelölje S-ben az i gyakoriságú elemek számát, vagyis azoknak az elemeknek a számát, amik i elemében megtalálhatóak $\mathcal{S}$-nek, és $r_{\geq i}$ a legalább i gyakoriságú elemek összegét. Ekkor $t = (|S|, r_2, \ldots, r_6, r_{\geq 7})$, mert a cikkben belátták, hogy elegendő vizsgálni $3 \leq |S| \leq 7$ eseteket, és azon felül nem fontos megkülönböztetni minden esetet, mert ezek dominálják a többit. Definiáljuk 4.2 hiányzó értékeit:

$$
\begin{aligned}
&- \Delta w_{OUT}(t) = \alpha_{|S|} + \sum_{i=2}^{6} r_i \Delta\beta_i + \Delta w'_{|S|,r_2}, \\
&- \Delta w_{IN}(t) = \alpha_{|S|} + \sum_{i=2}^{6} r_i \beta_i + r_{\geq 7} + \Delta\alpha_{|S|} \left(\sum_{i=2}^{6} (i-1) r_i + 6 \cdot r_{\geq 7} \right).
\end{aligned}
$$

Az $\alpha_2, \alpha_3, \alpha_4, \alpha_5, \beta_2, \beta_3, \beta_4, \beta_5$ értékeket szeretnénk meghatározni. $\alpha_1 = \beta_1 = 0, \alpha_6 = \alpha_7 = \beta_6 = \beta_7 = 1$ (és minden további 1) rögzítjük. Ezeket a súlyokat

$$
\mu(\mathcal{S}) = \sum_{i \geq 1} \alpha_i n_i + \sum_{j \geq 1} \beta_j m_j
$$

használjuk, ahol n_i az i elemszámú $S \in \mathcal{S}$ halmazok száma, m_j pedig a j gyakoriságú $u \in U$ elemek száma. Itt látható, hogy $0 \leq \alpha_i \leq 1, 0 \leq \beta_j \leq 1$ esetén, minden i, j-re, a μ mérték kisebb az eredeti MDS feladat csúcsszámának kétszeresénél, hiszen $|U| = |\mathcal{S}| = n$. Vagyis ezzel a mértékkel kapott MSC futásidő négyzete egy jó MDS futásidő. Meghatározva a maradék értékeket $\Delta\beta_i = \beta_i - \beta_{i-1}$ és

$$
\Delta w'_{|S|,r_2} = \begin{cases}
0 & \text{ha } r_2 = 0; \\
\beta_2 + \alpha_2 & \text{ha } r_2 = 1; \\
\beta_2 + \alpha_3 & \text{ha } r_2 = 2; \\
\beta_2 + \alpha_2 + \alpha_3 & \text{ha } r_2 = 3, |S| = 3; \\
\beta_2 + \alpha_4 & \text{ha } r_2 \geq 3, |S| \geq 4;
\end{cases}
$$

ahol feltettük, hogy $\alpha_i \leq \alpha_{i+1}$, $\beta_i \leq \beta_{i+1}$ és $\Delta\alpha_i \geq \Delta\alpha_{i+1}$ minden $i \geq 2$ esetben megkapjuk az optimalizálandó feladatot.

Mindezeket a feltételeket figyelembe véve lefuttattam a 3 programot, és hosszas futtatás után sikerült szinte ugyanazt a cikkben kapott $\lambda \leq 1.2352...$ értéket megkapni: $\lambda \leq 1.235208... < 2^{0.305}$ a 4.10 táblázatban szereplő súlyokkal.

i	α_i	β_i
2	0.378537	0.398000
3	0.757048	0.765424
4	0.910186	0.926858
5	0.976292	0.984472

4.10. ábra. Az α_i és β_i súlyok értékei $i = 2, \ldots, 5$ esetén 3 futtatása alapján.

5. Összefoglalás

Az egyre gyorsabb számítógépeknek köszönhetően - azon túl, hogy egyszerű algoritmusokkal képes jó eredményeket elérni - a Measure & Conquer egy rendkívül jó adatcsökkentési módszer, mert megfelelő számolási módszerekkel gyorsan talál új eredményt. Igaz, meg kell említeni, hogy ezek a javítások is végesek. Hosszas futtatással a 3.2 feladatra a 3. programmal csak minimális javítást sikerült elérnem a [17] eredményéhez képest; a $\alpha = 0.958234, \beta = 0.943555, \gamma = 0.901789$ értékekkel $\lambda = 1.996632$ jó megoldás, viszont a $\alpha = 0.958232, \beta = 0.943557, \gamma = 0.901787$ értékekkel a $\lambda = 1.996631$ is jó megoldás. Ezen túl, az elért új eredmény (10) bizonyítja, hogy bonyolultságát tekintve egyszerű módosításokkal is el lehet érni új eredményeket a módszernek köszönhetően.

Irodalomjegyzék

[1] Adi Avidor and Michael Langberg. The multi-multiway cut problem. *Theoretical Computer Science*, 377(1):35–42, 2007.

[2] Amotz Bar-Noy, Toni Böhnlein, David Peleg, Yingli Ran, and Dror Rawitz. Sparse graphic degree sequences have planar realizations. In *Proceedings of the MFCS 2024*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.

[3] Kristóf Bérczi, Tamás Király, and Daniel P Szabo. Multiway cuts with a choice of representatives. *arXiv preprint arXiv:2407.03877*, 2024.

[4] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiway cuts (extended abstract). In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '92, page 241–251, New York, NY, USA, 1992. Association for Computing Machinery.

[5] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, July 1960.

[6] David Eppstein. Quasiconvex programming, 2004.

[7] David Eppstein. Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. *ACM Trans. Algorithms*, 2(4):492–509, October 2006.

[8] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5), August 2009.

[9] Serge Gaspers and Mathieu Liedloff. A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs. In Fedor V. Fomin, editor, *Graph-Theoretic Concepts in Computer Science*, pages 78–89, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[10] Serge Gaspers and Gregory B. Sorkin. Separate, measure and conquer: Faster polynomial-space algorithms for max 2-csp and counting dominating sets. *ACM Trans. Algorithms*, 13(4), December 2017.

[11] Ernestine Großmann, Kenneth Langedal, and Christian Schulz. A comprehensive survey of data reduction rules for the maximum weighted independent set problem, 2024.

[12] Hua Jiang and Zhifei Zheng. An exact algorithm for the minimum dominating set problem. In *IJCAI*, pages 5604–5612, 2023.

[13] Jarod Kintz. Amortized analysis. 2018.

[14] O. Kullmann. New methods for 3-sat decision and worst-case analysis. *Theoretical Computer Science*, 223(1):1–72, 1999.

[15] Zhifeng Sun. Multicut survey. Technical report, Technical report, 2008. URL http://www. ccs. neu. edu/home/austin/papers . . . , 2008.

[16] Johan M.M. van Rooij and Hans L. Bodlaender. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17):2147–2164, 2011.

[17] Mingyu Xiao. Solving Directed Multiway Cut Faster Than 2^n. In Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman, editors, *32nd Annual European Symposium on Algorithms (ESA 2024)*, volume 308 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 104:1–104:13, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[18] Patric R.J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1):197–207, 2002. Special Issue devoted to the 6th Twente Workshop on Graphs and Combinatorial Optimization.

Alulírott Micskó Máté Benedek nyilatkozom, hogy szakdolgozatom elkészítése során az alább felsorolt feladatok elvégzésére a megadott MI alapú eszközöket alkalmaztam:

Feladat	**Eszköz**	**Hely**	**Megjegyzés**
Python kód generálás	gpt-4-turbo	Notebook	
Ábra készítés	gpt-4-turbo	1,3.9,4.10	Csak formázáshoz

A felsoroltakon túl más MI alapú eszközt nem használtam.