Maximális összsúlyú, szigetvilág tulajdonságú részgráf megtalálása előjeles, élsúlyozott gráfban
Diplomamunka

Fancsali Szabolcs Levente
alkalmazott matematikus hallgató
ELTE

Témavezető:
Vizvári Béla
docens
ELTE Operációkutatási Tanszék

Kivonat

Szakdolgozatomban egy, a gyakorlatban felmerülő, NP-teljes bonyolultságú problémára keresek a gyakorlatban jól használható algoritmust.

Bevezetésként definiálom az előjeles gráfok és a szigetvilág tulajdonságú előjeles gráfok fogalmát, majd kimondok és bizonyítok egy régi tételt, aminek következményét az algoritmusok használni fogják. Ezután ismertetem a gyakorlati problémát.

A témával előttem foglalkozók eredményeinek rövid összefoglalása után mutatom be az általam javasolt algoritmusokat. Ezek első csoportjába tartoznak az eddig használt algoritmusok Bruno Simeoni ötlete alapján javított változatai, második csoportjába pedig az első csoport fejlesztése közben született saját ötleteimen alapuló új, iterációs eljárások.

Tartalomjegyzék

1. fejezet

Bevezetés

1.1. Előjeles gráfok, szigetvilág gráfok

A későbbiekben ismertetendő gyakorlati probléma (egyik) matematikai modelljéhez szükséges az előjeles gráfok, és a szigetvilág tulajdonságú előjeles gráfok fogalma.

1. Definíció. *A $G(V, E)$ gráfot előjeles gráfnak nevezzük, ha az E élhalmazt P és N halmazok diszjunkt uniója alkotja. Ekkor a P-beli éleket „pozitív éleknek", az N-belieket „negatív éleknek" nevezzük.*

2. Definíció. *A $G(V, E = P \cup N)$ előjeles gráf szigetvilág tulajdonságú, ha a pozitív élek P halmaza által meghatározott $G^P(V, P)$ részgráf összefüggő komponensein belül az eredeti G gráfban nem halad N-beli („negatív") él.*

Egy előjeles gráf szigetvilág tulajdonságának az algoritmusok szempontjából kényelmes szükséges és elégséges feltételét fogalmazza meg a következő, régóta ismert tétel. A tétel kimondásához szükséges egy újabb definíció.

3. Definíció. *Egy előjeles gráfban „rossz" körnek nevezzük azokat a köröket, amelyek pontosan egy negatív élet tartalmaznak.*

Hogy ezeket a köröket miért nevezzük rosszaknak, az a tétel ismeretében magától értetődik.

1. Tétel. *Egy előjeles gráf pontosan akkor szigetvilág tulajdonságú, ha* NINCS *benne olyan kör, amelyben pontosan egy darab negatív él van („rossz kör").*

„Szigetvilág $\Rightarrow$ nincs rossz kör" irány bizonyítása . Egy szigetvilágban kétféle kör lehet:

- vagy teljes egészében egy $G^P(V,P)$-komponensen belül halad, ekkor csak pozitív élei vannak;
- vagy legalább kettő komponensbe belemetsz, ekkor van benne legalább kettő negatív él.

□

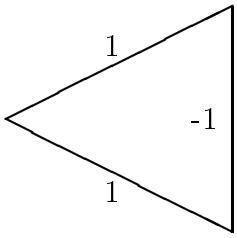
„Nem szigetvilág ⇒ van rossz kör" irány bizonyítása . Ha egy előjeles gráf nem szigetvilág, az azt jelenti, hogy $G^P(V,P)$ valamelyik komponensén belül az eredeti gráfban van negatív él. De ezen negatív él két végpontja között van pozitív élekből álló út. □

1. Következmény. *Ha egy tetszőleges előjeles gráfból oly módon hagyunk el éleket hogy minden rossz köréből legalább egy élet elhagyunk, akkor a maradék részgráf szigetvilág tulajdonságú lesz.*

1.2. Előjeles gráfok egy alkalmazása

Adva van valahány objektum, és ismertek az ezek közötti, valós számokkal kifejezhető kapcsolatok. Például valószínűségi változók és a közöttük lévő korrelációk. Úgy szeretnénk csoportosítani az objektumokat, hogy két, azonos csoportba tartozó elem között semmiféleképpen se legyen negatív számmal jellemzett kapcsolat, és két, különböző csoportba tartozó elem között semmiféleképpen se legyen pozitív számmal jellemezhető kapcsolat.

Természetesen ez általában nem oldható meg, a legegyszerűbb ellenpélda:



1. Állítás. *Ha az objektumok egy előjeles gráf pontjai, a negatív számmal jellemezhető kapcsolatok a negatív élek és a pozitív számmal jellemezhető kapcsolatok a pozitív élek, akkor a fenti tulajdonságú csoportosítás pontosan akkor valósítható meg, ha ez a gráf szigetvilág tulajdonságú.*

„Csoportosítható ⇒ szigetvilág" irány bizonyítása . Ha a és b pont ugyanabba a P-komponensbe tartozik, akkor a fenti csoportosításban feltétlenül

azonos csoportba kellett őket osztanunk, mert az őket összekötő P-beli út minden élére igaz, hogy az él két végpontjának azonos csoportba kell tartoznia. Ekkor viszont a csoportosítás tulajdonsága miatt nem lehet a és b között negatív él. □

„Szigetvilág $\Rightarrow$ csoportosítható" irány bizonyítása . A $G^P(V,P)$ részgráf összefüggő komponensei, mint csoportok (illetve egy csoport lehet több olyan komponens uniója is, amelyek között nem megy negatív él) teljesítik a csoportoktól elvárt tulajdonságokat. □

Ha tehát az objektumok előjeles gráfja nem szigetvilág, akkor a csoportosíthatóság érdekében „csalnunk" kell: el kell hagynunk néhány élt (azaz figyelmen kívül kell hagynunk néhány kapcsolatot) ahhoz, hogy az első tétel következménye nyomán szigetvilágot kapjunk, ami már csoportosítható. Cél, hogy minél „kevesebb"-et csaljunk. Ahhoz, hogy értelmezhessük, mi az a „kevesebb csalás", be kell vezetnünk a „csalás mértékét". Ehhez előbb ismerkedjünk meg egy gyakorlati problémával.

1.3. Egy tőzsdei probléma és modellje

Az előző részben megfogalmazott feladat gyakorlati értékét bemutatandó megemlítek egy valóságos problémát, amivel témavezetőm [1]-ben foglalkozott.

A Dow Jones index 30 részvényt tartalmaz. Meghatározott időszakban (ami lehet egy nap, vagy egy hét) figyeljük a részvényárfolyamok mozgását. Szeretnénk ezen megfigyeléseink alapján a részvényeket oly módon csoportosítani, hogy egy befektető reménykedhessen az azonos csoportba tartozó részvények árának hasonló viselkedésében.

Most tekintsünk el attól a (korántsem triviális) problémától, hogy hogyan is lehet megmérni két részvényárfolyam korrelációját, és tegyük fel, hogy már ismerünk minden számunkra fontos kapcsolatot bármely két árfolyam között. Ekkor a részvények közötti kapcsolatokat kényelmes módon egy élsúlyozott, előjeles gráffal adhatjuk meg, amiben azon részvények között lesz pozitív a kapcsolat, amelyeknek ára általában azonos irányba változik, és az éleket a megfelelő részvényárfolyamok közötti korrelációk abszolút értékével súlyozzuk.

A csoportosítást az előző fejezetben leírt módon szeretnénk elvégezni. A részvényárfolyamok előjeles gráfja tipikusan nem szigetvilág tulajdonságú, ezért itt is „csalnunk" kell, azaz figyelmen kívül kell hagynunk néhány kapcsolatot. Célunk, hogy a figyelmen kívül hagyott kapcsolatok minél kevesebbet problémát okozzanak az elvégzett csoportosítás után.

Ezt [1] szerzői úgy próbálják elérni, hogy az elhagyott élek összsúlyát (a figyelmen kívül hagyott korrelációk abszolút értékeinek összegét) szeretnék minimalizálni. Most már látható, hogy a „csalás" előző fejezetben említett mértékét érdemes az elhagyott élek összsúlyaként (a figyelmen kívül hagyott kapcsolatok abszolútértékeinek összegeként) értelmezni.

Tehát a tőzsdei problémában a feladat a következő: Adva van egy $G(V, P \cup N, W)$ előjeles, élsúlyozott gráf, ahol V a részvényeknek megfelelő pontok halmaza, P a pozitívan korrelált részvényeknek megfelelő pontokat összekötő éleknek, N a negatívan korrelált részvényeknek megfelelő pontokat összekötő éleknek a halmaza (a korrelálatlan részvényárfolyamokat reprezentáló pontokat nem köti össze semmilyen él sem), W pedig egy $W : P \cup N \to (0, 1]$, $\{u, v\} \mapsto |R(u, v)|$ függvény, ahol $R(u, v)$ jelöli az u és v részvényárfolyamok korrelációját. Keressük a $G(V, P \cup N, W)$ élsúlyozott előjeles gráfnak egy olyan $G'(V, P' \cup N', W')$ élsúlyozott részgráfját (azaz: $P' \subset P$, $N' \subset N$ és $\forall\{u, v\} \in P' \cup N' : W'(\{u, v\}) = W(\{u, v\})$), amely szigetvilág tulajdonságú, és a $(P \cup N) \setminus (P' \cup N')$-beli (azaz elhagyott) élek összsúlya a lehető legkisebb.

1.4. A feladatok

Az fenti tőzsdei probléma alapján megfogalmazható a szakdolgozat címében szereplő feladat:

Adva van egy tetszőleges $G(V, P \cup N, W)$ pozitívan élsúlyozott előjeles gráf (ahol $W : P \cup N \to \mathbb{R}_+$ a súlyfüggvény). Keressük benne azt a $G'(V, P' \cup N', W')$ részgráfot, amely szigetvilág tulajdonságú, és az éleinek összsúlya maximális a G szigetvilág tulajdonságú részgráfjai között.

Először vizsgáljuk meg a fenti feladat bonyolultságát! A következő tétel szintén [1]-ben található.

2. Tétel. *A maximális összsúlyú, szigetvilág tulajdonságú részgráf megtalálása NP-teljes feladat.*

Bizonyítás. Mint sok más esetben, az NP-teljességet itt is azzal bizonyítjuk, hogy egy már bizonyítottan NP-teljes problémát polinomiálisan visszavezetünk a most bizonyítandóan NP-teljes problémára. Esetünkben ez a közismerten NP-teljes probléma az úgynevezett „halmazfedési probléma" lesz.

A halmazfedési probléma a következő: Adva van egy n elemű H halmaz és k darab ($A_i \subset H$, $i = 1 \ldots k$) „fedendő" halmaz. A H halmaz minden eleméhez hozzá van rendelve egy súly (c_j, $j = 1 \ldots n$). Keressük azt az X halmazt, amely mindegyik „fedendő" halmazba belemetsz, és az elemeinek összsúlya

minimális az ilyen tulajdonságú halmazok között. Formálisan leírva:

$$min \sum_{j=1}^{n} c_j x_j$$

$$\sum_{j=1}^{n} A_{ij} x_j \ \geq 1 \ \forall i = 1..k$$

$$A_{ij}, x_j \in \{0, 1\}$$

ahol $\underline{x}$ az X halmaz karakterisztikus vektora, az $\underline{\underline{A}}$ mátrix sorai pedig a „fedendő" halmazok karakterisztikus vektorai. Tehát a halmazfedési probléma megadásához elég megmondani az $\underline{\underline{A}}$ mátrixot és a $\underline{c}$ súlyvektort.

1. Megjegyzés. *Az könnyen látható, hogy a mi problémánk hogyan vezethető vissza a fenti problémára: H a gráf élhalmaza, A_i-k a rossz körök élhalmazai, az elemek súlyai pedig az élek súlyai. (Persze a rossz köröket még meg is kellene találni.) A másik irányú visszavezetéshez be kellene látni, hogy tetszőleges halmazfedési problémához van olyan élsúlyozott előjeles gráf, amelyben a rossz körök élhalmazai pont a „fedendő" halmazok. Ennél azonban kicsit bonyolultabb a bizonyítás.*

Az $\underline{\underline{A}}$ mátrixszal és $\underline{c}$ súlyvektorral adott halmazfedési problémához konstruálunk egy $G(V, P \cup N, W)$ előjeles gráfot. Első lépésként megmondjuk, hogy mik lesznek a gráf pontjai: Legyen

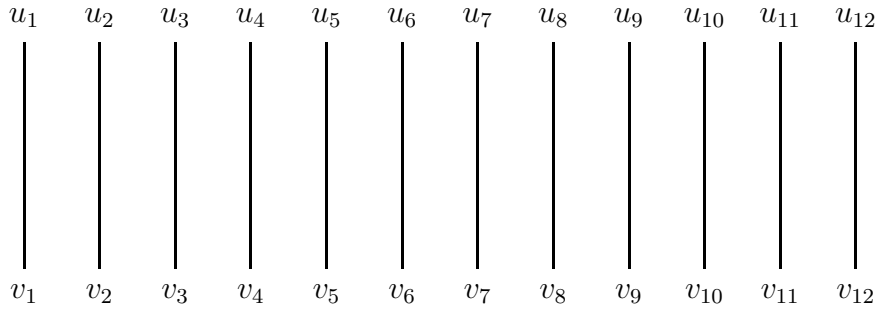
$$V := \{v_j | j = 1 \ldots n\} \cup \{u_j | j = 1 \ldots n\}.$$

Tehát a konstruálandó gráfnak kétféle pontja lesz: Az eredeti H halmaz minden elemének megfeleltetünk egy-egy pontot az u_j-kkel jelölt, illetve a v_j-kkel jelölt pontok közül is. (Az egyszerűség kedvéért a H halmaz elemeit az első n egész számmal indexelve a H j indexű pontjához rendeljük hozzá a v_j-t és az u_j-t.)

Azokat a pontokat, amelyeket ugyanazon H-beli pontnak feleltettünk meg, *pozitív* élekkel összekötjük egymással, és erre az össszekötő élre (amely élek tehát már egy-egyértelműen megfeleltethetők a H elemeinek) ráírjuk súlyként a megfelelő H-beli elem súlyát:

$$\{v_j, u_j\} \in P, W(\{v_j, u_j\}) := c_j \ \forall j = 1 \ldots n$$

Például 12 pontú H halmaz esetén:



Legyen

$$I(i) := \{j | A_{ij} = 1\} = \{j_1, j_2, \ldots j_{k_i}\}$$

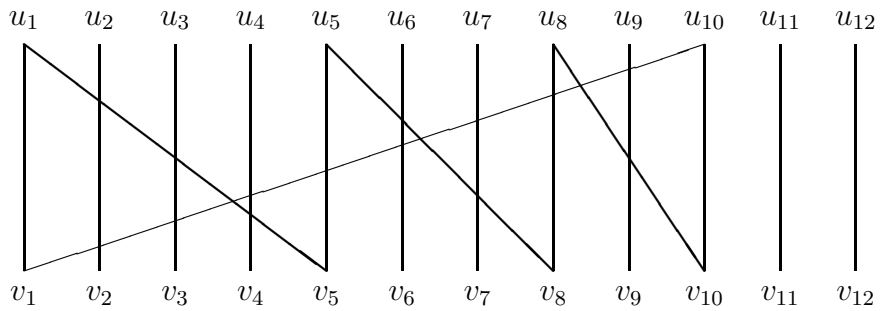
az i-edik fedendő halmaz elemeinek indexhalmaza. Feltehető, hogy $j_1 < j_2 < \ldots < j_{k_i}$. Minden $I(i)$ halmaz jelentsen újabb éleket:

$$\{u_{j_1}, v_{j_2}\}, \{u_{j_2}, v_{j_3}\}, \ldots \{u_{j_{k_i-1}}, v_{j_{k_i}}\} \in P,$$

$$\{u_{j_{k_i}}, v_{j_1}\} \in N.$$

Ezeknek az éleknek a súlya legyen olyan nagy, hogy elhagyásuk szóba se kerülhessen, például $(\sum_{j=1}^{n} c_j) * 10$. Tehát az $I(i)$ indexhalmaz elemeinek megfelelő H-beli elemeknek (azaz az i-edik fedendő halmaz elemeinek) megfelelő gráf-pontokat úgy kötjük össze, hogy az indexek szerint növekvő sorrendbe rakva az i-edik fedendő halmaz elemeit, minden elemre a megfelelő u_{j_i} pontot *pozitív* él köti össze a sorrendben következő (szintén az i-edik fedendő halmazbeli) elemnek megfelelő $v_{j_{i+1}}$ ponttal; és az i-edik fedendő halmaz sorrendben utolsó elemének megfelelő $u_{j_{k_i}}$ pontot *negatív* él köti össze az i-edik fedendő halmaz sorrendben legelső elemének megfelelő v_{j_1} ponttal.

Például a fenti példa H halmaza esetén legyen az egyik fedendő halmaz az $\{1, 5, 8, 10\}$ indexű pontok halmaza:



Látható, hogy minden $I(i)$ indexhalmazzal jellemzett fedendő halmaz *része* egy-egy (minden i-re különböző) rossz körnek, de ezen rossz körből csak

az $I(i)$-beli indexű éleket hagyhatjuk el, ha maximális összsúlyú részgráfot keresünk; és csak ezek lesznek a gráf rossz körei.

Ha megoldjuk a maximális összsúlyú szigetvilág részgráf problémát a fenti gráfra, azzal megoldottuk a tetszőlegesen adott $\underline{\underline{A}}, \underline{c}$ halmazfedési problémát is. □

A G' részgráfot két lépésben fogjuk megkeresni. Elsőként feltérképezzük az eredeti G gráfot, hogy „mennyire nem szigetvilág tulajdonságú", azaz megkeressük G-ben a rossz köröket.

Nyilván G-nek pontosan azok a részgráfjai lesznek szigetvilág tulajdonságúak, amelyek ezen körök egyikét sem tartalmazzák; ezeket a részgráfokat a $P \cup N$ élhalmaz olyan részhalmazainak elhagyásával kapjuk, amelyek mindegyik rossz kör élhalmazába belemetszenek. Az is magától értetődik, hogy ha G'-t úgy kaptuk, hogy kitöröltünk olyan éleket is, amelyek egyik rossz körben sem voltak benne, akkor - mivel azok újrabehúzásával nem ronthatjuk el a szigetvilág tulajdonságot - G' nem is lehet a szigetvilág részgráfok között maximális összsúlyú.

Ezek szerint a keresett G' részgráf azon részgráfok között keresendő, amelyeket úgy kapunk, hogy G minden rossz köréből elhagyunk legalább egy élet, de a rossz körökben nem szereplő éleket nem bolygatjuk. Így, ha ismerjük G rossz köreit (azaz az első lépést már elvégeztük), máris van egy egyszerű algoritmusunk az optimális G' megtalálására (ez lesz a második lépés): Soroljuk fel az összes olyan gráfot, amelyet úgy kapunk, hogy a G gráfból elhagyjuk bizonyos rossz körök bizonyos éleit, oly módon, hogy minden rossz körből hagyjunk el legalább egy élet. Az így kapott részgráfok közül vegyük a maximális összsúlyút.

Láthatóan ez az algoritmus még igencsak kezdetleges, de lényegében ezt a technikát fejlesztjük tovább.

2. fejezet

Előzmények

2.1. Majlender Péter ötlete: rövid körök

Tehát a feladat megoldásának első lépése a rossz körök megtalálása. Majlender Péternek volt az az ötlete, hogy először keressük meg a rövid rossz köröket, azaz a háromszögeket, illetve négyszögeket, és először csak ezek figyelembe vételével hajtsuk végre a feladat második lépését. Ezek után kerüljön sor a hosszabb rossz körök megkeresésére és kiküszöbölésére.

Tegyük fel, hogy a gráfunkban van h darab éldiszjunkt rossz kör. Ekkor nyilvánvalóan mindegyikből el kell hagynunk legalább egy-egy élet, azaz a h-nál kevesebb élet elhagyva kapható részgráfok szóba sem jöhetnek megoldásként, így a feladat megoldásának második lépésében hasznos előismeretekkel fogunk rendelkezni. Látható, hogy előnyös éldiszjunkt rossz körök minél nagyobb rendszerét ismerni. Majlender Péter javaslata azért lehet előnyös, mert rövid körök nagyobb valószínűséggel diszjunktak, mint a hosszúak.

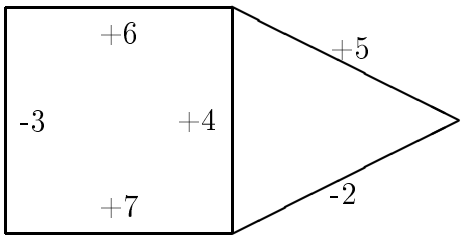
Gyakorlati szempontból azért is hasznos megfogadni Majlender Péter tanácsát, mert a rövid rossz köröket lényegesen rövidebb idő alatt meg lehet találni, mint a hosszúakat. A rövid rossz körök figyelembe vételével elhagyott élek között könnyen lehetnek hosszabb rossz körök élei is, így azokat már kiküszöböltük a következő lépés előtt, így kevesebb hosszú rossz kört kell (a rövidek megkeresésénél fáradtságosabb munkával) megkeresni.

Mivel Majlender Péter ezt a módszert a fenti tőzsdei problémára alkalmazta, érdemes megemlíteni, hogy ott a rossz körök nagy hányada rövid (3, illetve 4 hosszú) volt, és ezek kiküszöbölésével a hosszabb körök száma is jelentősen csökkent.

A fenti, gyakorlati szempontból jelentős előnyökkel szemben felmerül egy elméleti hátrány is: Ha a rövid köröket és a hosszú köröket két, egymás utáni lépésben küszöböljük ki a gráfból, könnyen előfordulhat, hogy nem

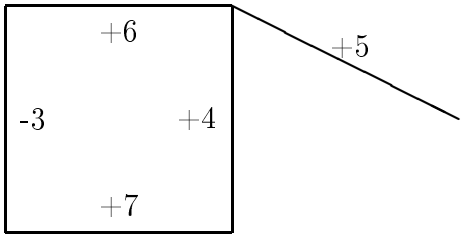
az optimális megoldást kapjuk. Erre adok egy könnyen átlátható példát, amiben az egyszerűség kedvéért csak a háromszögeket tekintsük rövid körnek, a négyszögeket már hosszúnak. Induljunk ki a következő gráfból:

Eredeti gráf



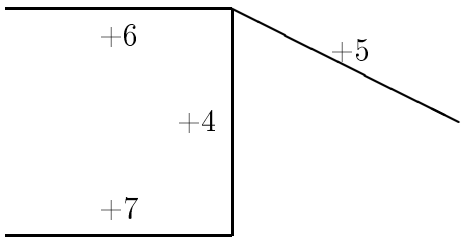
Ahol (a tőzsdei problémának megfelelően) azok az élek lesznek negatívak, amelyekre negatív szám van írva. Ebben kettő rossz kör van: a (-3, +6, +4, +7) négyszög és a (-2, +5, +4) háromszög. Első lépésben csak a háromszöget találjuk meg, és (itt éppen mohó módon) elhagyjuk a legkisebb súlyú (abszolútértékű) élét:

1. lépés után

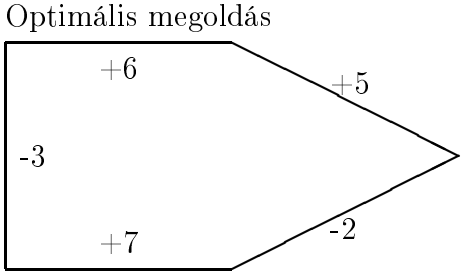


Ezután még megmarad a rossz négyszög. Második lépésben az találjuk meg és hagyjuk el a legkisebb súlyú élét:

2. lépés után



Most már nincs rossz kör, az összsúly 22. De ez nem optimális megoldás, mert van jobb: 23 összsúlyú.



Az is látszik, hogy az optimális esetben még a „szigetek" is egészen mások, mint a Majlender Péter javasolta módszerrel kapott esetben. Erre tehát a későbbiekben (azon algoritmusokban, ahol a kétlépéses rossz kör keresést alkalmazzuk), oda kell figyelni! (Bár természetesen előfordulhat, hogy több különböző minimális súlyú megoldás van, és két különböző program közül az egyik az egyiket, a másik a másikat találja meg. Ekkor is különbözhetnek a szigetek, pedig mindkettő optimális megoldás.)

2.2. Vizvári-Yilmaz, felsorolásos algoritmus

Témavezetőm Fatih Yilmazzal közös cikkében [3] olvasható eredményei az algoritmusunk második lépésére adnak megoldási javaslatot. Mivel a későbbiekben bemutatandó algoritmusaim egy része erre épül, ezért röviden ismertetem ezen eredményeket.

2.2.1. Az általános bináris programozási feladat

A fenti cikk az általános bináris programozási feladat megoldására keres algoritmust. Az általános bináris programozási feladat lineáris célfüggvénnyel a következő:

$$max\ \underline{cx}$$

$$\underline{x} \in S \subseteq \{0,1\}^n$$

ahol $\underline{c} \in \mathbb{N}^n$ és az általánosság megszorítása nélkül feltehető, hogy $c_1 \geq \ldots \geq c_n > 0$. Jól ismert, hogy az általános bináris programozási feladat NP-nehéz, ha S-ről nem teszünk fel semmit.

2.2.2. A felsorolásos módszer lényege

Tegyük fel, hogy az $\mathbf{1} \in S$, ahol $\mathbf{1}$ jelöli azt a vektort, amelynek mindegyik koordinátája 1. Ekkor nyilván ez lesz az optimális megoldás. Ha $\mathbf{1} \notin S$, akkor

a feladat optimális megoldásának, ha létezik, ki kell elégítenie a következő egyenlőtlenséget:

$$\sum_{j=1}^{n} x_j \leq n-1$$

Tekintsük a következő feladatot:

$$z_2 = max\ \underline{cx},$$

$$\sum_{j=1}^{n} x_j \leq n-1,$$

$$\underline{x} \in \{0,1\}^n,$$

és tegyük fel, hogy az optimális megoldása $\beta_2 \in \{0,1\}^n$ Látható, hogy

$$(\beta_2)_j = \begin{cases} 1 & \text{ha } j \in \{1,\ldots,n-1\} \\ 0 & \text{ha } j = n \end{cases}$$

ekkor nyilvánvaló, hogy $\beta_2 \in S$ esetén β_2 optimális megoldás. Most tegyük fel azt, hogy β_2 sem eleme S-nek. Ebből következik, hogy az eredeti feladat optimális megoldásának ki kell elégítenie a

$$\sum_{j=1}^{n} x_j \leq n-1,$$

$$\beta_2 \underline{x} \leq \beta_2 \mathbf{1} - 1$$

$$x \in \{0,1\}^n,$$

feltételeket. Ha folytatjuk ezt az eljárást, a következő általánosítás adódik: Legyenek $\mathbf{1} = \beta_1, \beta_2, \beta_3, \ldots, \beta_k$ $(2 \leq k \leq 2^n)$ a „legyártott" vektorok addig, amíg egyikük sem lehetséges megoldása az eredeti feladatnak. Ekkor a következő

$$z_{k+1} = max\ \underline{cx},$$

$$\underline{x} \in P_k := \{\underline{x} \in \{0,1\}^n : \beta_l \underline{x} \leq \beta_l \mathbf{1} - 1\ \forall l = 1 \ldots k\}$$

feladat megoldódik a következő iterációval, és az optimális megoldását β_{k+1} adja.

1. Lemma. *Tegyük fel, hogy* $\beta_1, \ldots, \beta_k \notin S$. *Ekkor* $S \subseteq P_k$.

Bizonyítás. Tegyük fel indirekt, hogy $\exists \underline{y} \in S$: $\underline{y} \notin \{\beta_1, \ldots, \beta_k\}$, de $\underline{y} \notin P_k$. Tekintsük a következő r indexet:

$$r := min\{l : \beta_l \underline{y} > \beta_l \mathbf{1} - 1 : 1 \leq l \leq k\}.$$

Nyilvánvaló, hogy $r \geq 2$, egyébként $r = 1, \underline{y} = \beta_1$ lenne. Következik, hogy

$$\beta_r \underline{y} = \beta_r \mathbf{1}$$

ebből

$$\underline{y} \geq \beta_r \text{ és } \underline{y} \neq \beta_r,$$

ezért

$$\underline{yc} > \beta_r \underline{c}$$

$$\lightning$$

□

3. Tétel. *Legyen k a legkisebb olyan index, amelyre $\beta_{k+1} \in S$. Ekkor ő az eredeti feladat optimális megoldása.*

Bizonyítás. Következik a fenti lemmából. □

Az iménti bekezdés a következő megoldási javaslatot sugallja: Soroljuk fel a β_k vektorokat, és álljunk meg az első olyannál, ami benne van az S halmazban. Boros Endre vette észre, hogy minden lépésben a feltételhalmaz csak a már „legyártott" vektorokat zárja ki, ezért a következő lépésben az a vektor lesz az optimális megoldás, amelyik az eddig fel nem sorolt vektorok közül a legnagyobb célfüggvényértékkel rendelkezik.

2.2.3. A felsoroló algoritmus

Az előző részben leírtak szerint meg kell valósítani a bináris vektorok felsorolását célfüggvényértékük szerint csökkenő sorrendben. Erre kerestek [3] szerzői jól működő megvalósítást, a bináris vektorok rákövetkezőjének talán kevésbbé ismert, illetve a bináris keresőfa jólismert fogalmának felhasználásával. Mindkét fogalmat a szokásostól kicsit eltérő módon vezetik be, mert a vektorokat adott sorrendben kell felsorolni, és a fa-műveletek közül csak a beszúrásra és a törlés egy speciális fajtájára van szükség.

Először a pontosan k egyest tartalmazó vektorok felsorolására adnak módszert. Legyen $\{i_1, \ldots, i_k\}$ és $\{j_1, \ldots, j_k\}$ az $\underline{u}$, illetve a $\underline{v}$ vektorok egyes értékű komponenseinek indexhalmaza. Ha $j_p \leq i_p \; \forall \, p = 1 \ldots k$, akkor $c_1 \geq \ldots \geq c_n$ miatt $\underline{cu} \geq \underline{cv}$.

4. Definíció. *Legyen* $\{i_1, \ldots, i_k\}$ *az* $\underline{u}$ *vektor egyeseinek indexhalmaza. Tegyük fel, hogy valamely* p *indexre az* $i_p < i_{p+1} - 1$ *egyenlőség teljesül, ahol* $i_{k+1} = n + 1$. *Ekkor definiáljuk a következő* $\underline{u'}$ *vektort:*

$$u'_j := \begin{cases} 1 & ha\ j = i_p + 1 \\ 0 & ha\ j = i_p \\ u_j & ha\ j \neq i_p, i_p + 1 \end{cases}$$

Az $\underline{u'}$ *vektor az* $\underline{u}$ *vektornak egy rákövetkezője.*

2. Megjegyzés. *Ha* $\underline{u'}$ *vektor az* $\underline{u}$ *vektornak rákövetkezője, akkor* $\underline{cu} \geq \underline{cu'}$. *És az egyenlőség akkor teljesül, ha* $c_{i_p+1} = c_{i_p}$. *Egy vektornak számos rákövetkezője lehet.*

4. Tétel. *Legyen* $\{i_1, \ldots, i_k\}$, *illetve* $\{j_1, \ldots j_k\}$ *az* $\underline{u}$ *és a* $\underline{v}$ *bináris vektorok egyeseinek indexhalmaza. Tegyük fel, hogy* $i_p \leq j_p\ \forall p = 1, \ldots, k$. *Ekkor* $\exists$ *bináris vektorok olyan sorozata, hogy* $\underline{u} = \underline{w_0}, \underline{w_1}, \ldots, \underline{w_t} = \underline{v}$ *és* $\underline{w_l}$ *vektor rákövtekezője a* $\underline{w_{l-1}}$ *vektornak. (* $\forall l = 1, \ldots, t$ *)*

Bizonyítás. Tegyük fel, hogy $\underline{u} \neq \underline{v}$ (egyébként $t = 0$). Legyen $p = max\{q : i_q < j_q\}$. Innentől $u_{i_p+1} = 0$ ugyanis ha $u_{i_p+1} = 1$ lenne, akkor $i_p + 1 \in \{i_1, \ldots, i_k\} \Rightarrow i_p + 1 = i_{p+1}$ miatt $i_{p+1} = i_p + 1 \leq j_p < j_{p+1}$ $\lightning$.

Legyen

$$(w_1)_j := \begin{cases} 1 & \text{ha } j = i_p + 1 \\ 0 & \text{ha } j = i_p \\ (w_0)_j & \text{ha } j \neq i_p, i_p + 1 \end{cases}$$

Ekkor a $\underline{w_1}$ egyeseinek indexhalmaza: $\{i_{1,1}, \ldots, i_{1,k}\} = \{i_1, \ldots, i_{p-1}, i_p + 1, \ldots, i_k\}$. Magától értetődik, hogy $i_{1,l} \leq j_l\ \forall l = 1, \ldots, k$.

Most vagy $\underline{w_1} = \underline{v}$, vagy a fenti eljárás $\underline{w_1}$-re megismételhető. □

Látható, hogy minden pontosan k darab egyest tartalmazó vektor rákövetkezők egy sorozatán keresztül örököse annak az $\underline{x}$ vektornak, amelyben $x_1 = x_2 = \ldots = x_k = 1$, és $x_{k+1} = \ldots = x_n = 0$. Így minden ilyen vektor felsorolható a következő algoritmussal, ahol L a felsorolt bináris vektorok listája, és $\underline{e_j}$ a j-edik standard egységvektor.

1. Algoritmus.

1. *Eljárás*
2. $L := \{\sum_{j=1}^{k} \underline{e_j} = (1, \ldots, 1, 0, \ldots, 0)\}$
3. *Ciklus, amíg* $L \neq \emptyset$

4. $\underline{u} \leftarrow L$

5. $L := L \cup \{\underline{v} : \underline{v}$ *a fenti* $\underline{u}$*-nak rákövetkezője* $\}$

6. Ciklus vége

7. Eljárás vége

Ahol $\underline{u} \leftarrow L$ azt jelenti, hogy az $\underline{u}$ vektort *kivesszük* az L listából.

Mélyebb szervezettség nélkül ez az algoritmus túl sokat fog dolgozni, mert két vektor között több, a fenti tételben megfogalmazott tulajdonságú, rákövetkezőkből álló vektorsorozat is lehet, így egy vektor többször is generálódhat (különböző vektorok rákövetkezőjeként). Például az $(1,0,1,0)$ vektornak rákövetkezőjeként belekerül a listába $(0,1,1,0)$ és $(1,0,0,1)$ is, majd $(0,1,0,1)$ vektor mindkettőjüknek rákövetkezője lévén kétszer is generálódik, sőt van olyan sorrendje a vektorok kiolvasásának, hogy hiába kerül ki egyszer a listából, egyszer újra belekerül.

A fenti probléma kiküszöbölésére vezették be [3] szerzői a bináris vektorok következő rendezését:

5. Definíció. $\underline{u} \triangleright \underline{v}$ *ha vagy* $\underline{cu} > \underline{cv}$*, vagy* $\underline{cu} = \underline{cv}$ *és* $\underline{u} \succ \underline{v}$*, ahol* $\succ$ *a lexikografikus rendezés.*

Bármely két bináris vektor összehasonlítható $\triangleright$ szerint, azaz bináris vektorok bármely halmazának van ezen rendezés szerinti legnagyobb eleme. Az is látható, hogy ha $\underline{u'}$ az $\underline{u}$ rákövetkezője, akkor $\underline{u} \triangleright \underline{u'}$.

2. Lemma. *Ha az 1. algoritmusban az* $\underline{u}$ *vektor kiválasztása a*

$$4'.\ \underline{u} \leftarrow (max\triangleright)L$$

szerint történik, akkor egy vektort sem választunk ki kétszer.

Bizonyítás. Ha az $\underline{u}$ vektort a t-edik lépésben kiválasztottuk, akkor a $\triangleright$ rendezés szerint ez volt a legnagyobb olyan vektor, amelyet L tartalmazott, és a 6. lépésben $\triangleright$ szerint kisebb vektorok halmazával helyettesítettük. Az algoritmus során végig igaz az, hogy a 6. lépésben a kiválasztottnál kisebb vektorok kerülnek a listába, ezért $\underline{u}$ nem kerülhet újra bele. □

Ezek után a következő algoritmust kapjuk az összes vektor felsorolására:

2. Algoritmus.

1. Eljárás

2. $L := \{\sum_{j=1}^{k} \underline{e_j} : k = 1, \ldots, n\}$

3. *Ciklus, amíg* $L \neq \emptyset$

4. $\underline{u} \leftarrow (max\triangleright)L$

5. $L := L \cup \{\underline{v} : \underline{v}$ *az* $\underline{u}$ *rákövetkezője*$\}$

6. *Ciklus vége*

7. *Eljárás vége*

Ahol $\underline{u} \leftarrow (max\triangleright)L$ azt jelenti, hogy az L lista $\triangleright$ szerint maximális elemét *kivesszük* az L listából, és beletesszük az $\underline{u}$ vektorba. A továbbiakban is így fogom jelölni ezt a lépést.

5. Tétel. *A 2. algoritmus 4. sorában minden vektort pontosan egyszer választunk ki.*

Bizonyítás. Az előző lemmából következően elég azt bizonyítani, hogy minden vektort legalább egyszer kiválasztottunk. Tegyük fel indirekt, hogy a $\underline{v}$ vektort egyszer sem soroltuk fel. $k := \sum_{j=1}^{n} v_j$ Tekintsünk egy rákövetkezőkből álló sorozatot $\sum_{i=1}^{n} \underline{e_i}$-től $\underline{v}$-ig. Az általánosság megszorítása nélkül föltehetjük, hogy ebből a sorozatból csak $\underline{v}$-t nem választottuk ki. Ekkor $\underline{w_t} = \underline{v}$ a $\underline{w_{t-1}}$ rákövetkezője, amit már valamelyik korábbi lépésben kiválasztottunk, azaz ott a $\underline{w_t} = \underline{v}$ rákövetkezőként belekerült az L listába, tehát, ha $\underline{v}$-t soha nem választottuk ki, akkor L soha sem ürül ki, ami azt jelenti, hogy végtelen sok, nála $\triangleright$ szerint nagyobb bináris vektor van $\lightning$. □

2.2.4. Alkalmazás a szigetvilág problémára

A [3] cikk által tárgyalt problémától eltér a mi problémánk. Nekünk minimalizálnunk kell a célfüggvényértéket, miközben arra kell ügyelnünk, hogy a megoldásvektor egyeseinek indexhalmaza minden rossz kör éleinek indexhalmazát messe. Látható, hogy ez valójában nem lényeges különbség.

2.3. Bruno Simeoni ötlete

A fenti algoritmussal az a gond, hogy egy kiválasztott $\underline{u}$ vektor összes rákövetkezőjét „legyártja”, akkor is ha az már benne van a listában. Csak azt küszöböltük ki, hogy ha valami egyszer kikerült a listából, az nem kerül oda vissza, de ha valami még nem került ki a listából, akkor természetesen szerepelhet a beszúrandó elemek között (csak éppen a lista ezután is egyszer fogja nyilvántartani). Bruno Simeoni javasolta a következő definíciókat:

6. Definíció. *Egy $\underline{u}$ bináris vektornak a $\underline{v}$ bináris vektor a gyereke, ha a $\underline{v}$ az $\underline{u}$ valamelyik* 0 *elemének* 1*-re változtatásával megkapható.*

7. Definíció. *Egy $\underline{u}$ bináris vektornak a $\underline{v}$ bináris vektor a testvére, ha a $\underline{v}$ az $\underline{u}$ valamelyik* 01 *szeletének* 10*-ra cserélésével kapható.*

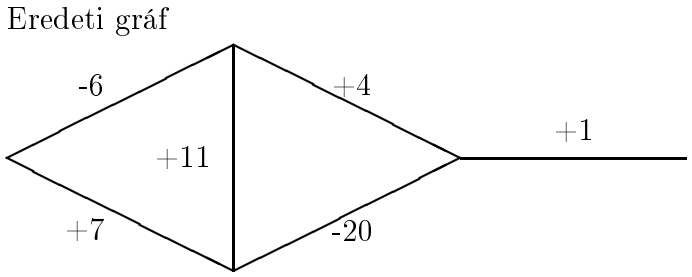
3. Megjegyzés. *Látható, hogy itt a „testvérség" nem reflexív, hanem irreflexív reláció.*

8. Definíció. *Legkisebb gyerek, illetve testvér: a gyerekek, illetve a testvérek közül a $\triangleright$ rendezés szerint legkisebb.*

Bruno Simeoni egyik ötlete a következő:

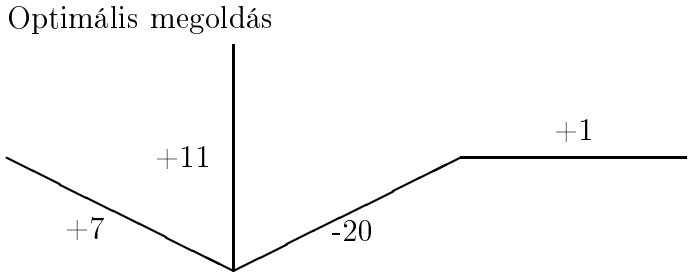
1. Sejtés. *Ha a* **0***, csupa 0 koordinátájú vektorból indulunk, és minden lépésben a legkisebb elem legkisebb testvérét, illetve legkisebb gyerekét vesszük bele a listába, akkor minden vektort pontosan egyszer fogunk felsorolni.*

Ha ez a sejtés igaznak bizonyulna, az kellemes volna a felsoroló algoritmus szempontjából, annál is inkább, mert a lista rendezetten tartása is kevesebb erőfeszítést igényelne. Sajnos azonban ez a sejtés ebben a formában nem igaz. Lesznek olyan vektorok, amelyeket íly módon nem tudunk felsorolni, és ezek között ott lehet az optimális megoldás is. Ennek illusztrálására ismét egy konkrét ellenpéldát adok:



Ahol a negatív éleket ismét az él súlya elé tett előjelekkel jeleztem. Az élsúlyok (természetesen előjel nélkül) csökkenő sorrendben a következők: $c_1 = 20 > c_2 = 11 > c_3 = 7 > c_4 = 6 > c_5 = 4 > c_6 = 1$.

Az optimális megoldást a $(0, 0, 0, 1, 1, 0)$ vektor egyeseinek megfelelő élek elhagyásával kapjuk:



Megmutatom, hogy a $(0,0,0,1,1,0)$ vektort nem soroljuk fel. Minden lépésben jelzem a $\triangleright$ szerint rendezett aktuális listát a súlyok feltüntetésével, de kihagyom belőle a kettőnél több egyest tartalmazó vektorokat, mert azoknak semmiféle leszármazottjuk sem tartalmazhat náluk kevesebb egyest: Indulunk a csupa nulla $(0,0,0,0,0,0)$ vektorból. Első lépésként belevesszük a listába az ő legkisebb gyerekét (testvére nincs): $(0,0,0,0,0,1)$.

$(0,0,0,0,0,1)$, súlya: 1

Következő lépésben ezt a vektort vesszük ki a listából, és az ő legkisebb testvérét: $(0,0,0,0,1,0)$, illetve legkisebb gyerekét: $(0,0,0,0,1,1)$ vesszük bele a listába.

$(0,0,0,0,1,0)$, súlya: 4

$(0,0,0,0,1,1)$, súlya: 5

Most $(0,0,0,0,1,0)$ vektort vesszük ki a listából, és az ő legkisebb testvérét: $(0,0,0,1,0,0)$, illetve legkisebb gyerekét: $(0,0,0,0,1,1)$ vesszük bele a listába.

$(0,0,0,0,1,1)$, súlya: 5

$(0,0,0,1,0,0)$, súlya: 6

Most a $(0,0,0,0,1,1)$ vektor kerül sorra, mint említettem az ő gyerekei már nem számítanak, hiszen kettőnél több egyest tartalmaznak, a legkisebb testvére: $(0,0,0,1,0,1)$.

$(0,0,0,1,0,0)$, súlya: 6

$(0,0,0,1,0,1)$, súlya: 7

A következő kiválasztott: $(0,0,0,1,0,0)$. Az ő legkisebb testvére: $(0,0,1,0,0,0)$, legkisebb gyereke: $(0,0,0,1,0,1)$.

$(0,0,0,1,0,1)$, súlya: 7

$(0,0,1,0,0,0)$, súlya: 7

A $(0,0,0,1,0,1)$ gyerekeivel az ismert okok miatt nem foglalkozom, a legkisebb testvére: $(0,0,1,0,0,1)$ (és nem az optimális megoldás, mert annak 10 a súlya, ennek pedig 8).

$$(0,0,1,0,0,0), \text{ súlya: } 7$$

$$(0,0,1,0,0,1), \text{ súlya: } 8$$

És most már látható, hogy ezek egyikének sem lesz semmilyen legkisebb gyerek - legkisebb testvér sorozaton keresztül sem a leszármazottja az optimális megoldás, hiszen legkisebb testvér és legkisebb gyerek képzésével is megmarad az az egyes, ami most a harmadik helyen áll, legfeljebb még előbbre csúszik $\lightning$.

Ha viszont a legkisebb testvér és a legkisebb gyerek fogalmát kicsit módosítjuk, akkor már igaz lesz a sejtés:

2. Sejtés. *A* $\{\mathbf{0}\}$ *listából indulva az*

1. *„a leghátsó* 01 *párt* 10 *párra cseréljük" („testvérképzés")*
2. *„ha az utolsó komponens* 0*, akkor azt* 1*-re változtatjuk" („gyerekképzés")*

műveletekkel már minden vektor megkapható.

Bizonyítás. Az egyesek száma szerinti indukcióval.

Tegyük fel, hogy minden, k darab egyest tartalmazó vektor előáll így ($k = 0$-ra nyilván igaz). Legyen $\underline{v}$ egy tetszőleges, $k+1$ darab egyest tartalmazó vektor.

- Ha $\underline{v}$ utolsó számjegye 1, akkor jelölje $\underline{v'}$ azt a vektort, amelynek az utolsó koordinátája 0, a többi koordinátája viszont megegyezik $\underline{v}$ megfelelő koordinátáival. Ekkor $\underline{v'}$ k darab egyest tartalmaz, így előáll a fenti módon, az utolsó komponensének 1-re változtatásával pedig $\underline{v}$ is előáll.

- Ha $\underline{v}$ utolsó számjegye 0, akkor jobbról balra haladva az első 1-est megkeressük. $\underline{v}$-re alkalmazzuk az 1. művelet inverzét annyiszor, hogy az utolsó helyre kerüljön az 1-es. Így kapjuk a $\underline{v''}$ vektort, ami olyan mint az előző pontban leírt vektor, ezért előáll. $\underline{v''}$ vektorra megfelelően sokszor alkalmazva az 1. műveletet a $\underline{v}$ vektor is előáll.

□

2. Állítás. *A második sejtésben definiált műveletekkel növeljük, de legalábbis semmiképpen sem csökkentjük a vektor súlyát.*

Bizonyítás.

- „testvérképzés” esetén: A leghátsó 01 párt 10-ra cseréljük. Jelöljük ezek helyét j-vel és $(j+1)$-gyel. Emlékeztetek rá, hogy feltettük, hogy $c_1 \geq \ldots \geq c_n > 0$. Az eredeti vektor súlyába beleszámító c_{j+1} és bele nem számító c_j szerepe a testvérben felcserélődik: a súly (c_{j+1})-gyel csökken, és c_j-vel növekedik. Így $c_j \geq c_{j+1}$ miatt teljesül az állítás.
- „gyerekképzés” esetén az eredeti vektor súlyához hozzáadódik $c_n > 0$.

□

2. Következmény. *A második sejtésben definiált műveletekkel egy $\underline{u}$ vektor minden leszármazottja legalább akkora súlyú, mint maga az $\underline{u}$ vektor.*

Ha a második sejtés (immáron állítás) alapján szervezzük az algoritmusunkat, akkor valóban szükségünk van a lista rendezetten tartását biztosító adatstruktúrákra, mert ekkor nem feltétlenül lesz a legenerált vektorok közötti első megengedett megoldás optimális. De ennek a vektornak már biztosan elhagyható az összes gyereke, hiszen azok biztosan nagyobb súlyúak, és ezek minden leszármazottja (testvéreikkel együtt) is nagyobb súlyú a fenti vektornál. De ezekkel a bekezdésekkel már át is tértem a témavezetőmmel közösen elért eredményeink bemutatására, amit a következő fejezetekben folytatok.

3. fejezet

A felsorolásos algoritmus javítása

Tehát [3] eredményeképpen (Bruno Simeoni javító célú ötlete nélkül) a következő algoritmust kaptuk a dolgozat címében szereplő probléma megoldására:

3. Algoritmus.

1. *Eljárás*
2. *Input: Egy $G(V, P \cup N, W)$ pozitív valós számokkal élsúlyozott előjeles gráf.*
3. *Körkeresés*
4. *Keressük meg G-ben a rossz köröket.*
5. *Legyen E a rossz körök élhalmazainak uniója.*
6. *Legyen $\{1, \ldots, n = |E|\}$ az $E = \{e_1, \ldots, e_n\}$ élhalmaz elemeinek indexhalmaza.*
7. *Legyen $\underline{\underline{A}}$ az a mátrix, amelynek sorai pont a rossz körök karakterisztikus vektorai.*
8. *Legyen $\underline{c}$ az a vektor, amelyre $c_j = W(e_j)\ \forall j = 1, \ldots, n$.*
9. *Körkeresés vége*
10. *Élelhagyás*
11. $L := \{\sum_{j=1}^{k} \underline{e_j} : k = 1, \ldots, n\}$
12. $\underline{u} := \mathbf{0}$

13. *Ciklus, amíg* $L \neq \emptyset$*, és amíg* $\underline{\underline{Au}}$ *vektor nem minden koordinátája legalább* 1.

14. $\underline{u} \leftarrow (min\triangleright)L$

15. $L := L \cup \{\underline{v} : \underline{v}$ *az* $\underline{u}$ *rákövetkezője*$\}$

16. *Ciklus vége*

17. *Élelhagyás vége*

18. *Output: A* G *gráfból elhagyva az* $\underline{u}$ *vektor egyeseinek megfelelő élek.*

19. *Eljárás vége*

Ahol $\underline{u} \leftarrow (min\triangleright)L$ azt jelenti, hogy az L lista $\triangleright$ szerint minimális elemét *kivesszük* az L listából, és beletesszük az $\underline{u}$ vektorba. A továbbiakban is így fogom jelölni ezt a lépést.

Ennek az algoritmusnak a „külső felépítését" egy ideig nem fogom megváltoztatni, azaz a rossz körök megkeresését és a rossz körök kiküszöbölését két külön részben fogom megoldani. Kezdjük a rossz körök keresésével.

3.1. Rossz körök keresése

Az, hogy a G gráfot milyen adatstruktúrában tároljuk, elsősorban attól függ, hogy a gyakorlatban megoldandó probléma gráfja milyen módon van tárolva. Egyelőre adatsruktúrától függetlenül vizsgálódjunk!

A rossz háromszögek megtalálása a következő módon történhet. Minden negatív élre végrehajtjuk a következő eljárást: Megkeressük az egyik végpontjának a szomszédait a pozitív élek alkotta részgráfban, megkeressük a másik végpontjának a szomszédait a pozitív élek alkotta részgráfban, majd vesszük ennek a két halmaznak a közös részét. A másik módszer a rossz háromszögek megkeresésére az, hogy minden negatív él egyik végpontjából kettő hosszú utakat keresek a pozitív élek részgráfjában a másik végpontjába. A rossz négyszögek megkeresésére szintén több módszer adódik. Az egyik, hogy minden negatív él egyik végpontjából egy hosszú úton elérhető, a másik végpontjából kettő hosszú úton elérhető pontokat keresek. A másik, hogy minden negatív él egyik végpontjából három hosszú utakat keresek a másik végpontjába. A méghosszabb körök keresésére mégtöbb lehetőség van. Most látjuk, milyen jó hasznát vehetnénk Majlender Péter ötletének.

Majlender Péter javaslatainak megfogadásával a fenti algoritmus a következőképpen módosulhatna:

4. Algoritmus.

1. *Eljárás*
2. *Input: Egy $G(V, P \cup N, W)$ pozitív valós számokkal élsúlyozott előjeles gráf.*
3. *Körkeresés*
4. *Keressük meg G-ben a* rövid *rossz köröket.*
5. *Legyen E a rövid rossz körök élhalmazainak uniója.*
6. *Legyen $\{1, \ldots, n = |E|\}$ az $E = \{e_1, \ldots, e_n\}$ élhalmaz elemeinek indexhalmaza.*
7. *Legyen $\underline{\underline{A}}$ az a mátrix, amelynek sorai pont a rövid rossz körök karakterisztikus vektorai.*
8. *Legyen $\underline{c}$ az a vektor, amelyre $c_j = W(e_j)\ \forall j = 1, \ldots, n$.*
9. *Körkeresés vége*
10. *Élelhagyás*
11. $L := \{\sum_{j=1}^{k} \underline{e_j} : k = 1, \ldots, n\}$
12. $\underline{u} := \mathbf{0}$
13. *Ciklus, amíg $L \neq \emptyset$, és amíg $\underline{\underline{A}}\underline{u}$ vektor nem minden koordinátája legalább* 1.
14. $\underline{u} \leftarrow (min\triangleright)L$
15. $L := L \cup \{\underline{v} :$ *$\underline{v}$ az $\underline{u}$ rákövetkezője*$\}$
16. *Ciklus vége*
17. *Élelhagyás vége*
18. *Output: A G gráfból hagyjuk el az $\underline{u}$ vektor egyeseinek megfelelő éleket.*
19. *Körkeresés*
20. *Keressük meg G-ben a még megmaradt rossz köröket.*
21. *Legyen E a megmaradt rossz körök élhalmazainak uniója.*

22. *Legyen* $\{1, \ldots, n = |E|\}$ *az* $E = \{e_1, \ldots, e_n\}$ *élhalmaz elemeinek indexhalmaza.*

23. *Legyen* $\underline{\underline{A}}$ *az a mátrix, amelynek sorai pont a megmaradt rossz körök karakterisztikus vektorai.*

24. *Legyen* $\underline{c}$ *az a vektor, amelyre* $c_j = W(e_j)\ \forall j = 1, \ldots, n.$

25. *Körkeresés vége*

26. *Élelhagyás*

27. $L := \{\sum_{j=1}^{k} \underline{e_j} : k = 1, \ldots, n\}$

28. $\underline{u} := \mathbf{0}$

29. *Ciklus, amíg* $L \neq \emptyset$, *és amíg* $\underline{\underline{A}}\underline{u}$ *vektor nem minden koordinátája legalább 1.*

30. $\underline{u} \leftarrow (min\triangleright)L$

31. $L := L \cup \{\underline{v} : \underline{v}$ *az* $\underline{u}$ *rákövetkezője*$\}$

32. *Ciklus vége*

33. *Élelhagyás vége*

34. *Output: A G gráfból elhagyva az* $\underline{u}$ *vektor egyeseinek megfelelő élek.*

35. *Eljárás vége*

Sajnos, mint láttuk, ez az algoritmus nem feltétlenül fogja az optimális megoldást adni. Viszont van lehetőség arra, hogy mégis megtaláljuk az optimális megoldást, miközben kihasználjuk Majlender Péter ötletét is: A hosszabb rossz körök megkeresését és kiküszöbölését ne csak a rövid rossz köröket kiküszöbölő optimális megoldásra, hanem a rövid rossz köröket kiküszöbölő *összes* megoldásra futtassuk le!

5. Algoritmus.

1. *Eljárás*

2. *Input: Egy* $G(V, P \cup N, W)$ *pozitív valós számokkal élsúlyozott előjeles gráf.*

3. *Körkeresés*

4. *Keressük meg G-ben a* rövid *rossz köröket.*
5. *Legyen E a rövid rossz körök élhalmazainak uniója.*
6. *Legyen $\{1, \ldots, n = |E|\}$ az $E = \{e_1, \ldots, e_n\}$ élhalmaz elemeinek indexhalmaza.*
7. *Legyen $\underline{\underline{A}}$ az a mátrix, amelynek sorai pont a rövid rossz körök karakterisztikus vektorai.*
8. *Legyen $\underline{c}$ az a vektor, amelyre $c_j = W(e_j)\ \forall j = 1, \ldots, n$.*
9. *Körkeresés vége*
10. *Élelhagyás*
11. $L := \{\sum_{j=1}^{k} \underline{e_j} : k = 1, \ldots, n\}$
12. $\underline{u} := \mathbf{0}$
13. *Ciklus, amíg $L \neq \emptyset$*
14. $\underline{u} \leftarrow (min\triangleright)L$
15. *Elágazás: Ha $\underline{\underline{A}}\underline{u}$ vektor minden koordinátája legalább 1, akkor:*
16. *A G gráfból hagyjuk el az $\underline{u}$ vektor egyeseinek megfelelő éleket: G'.*
17. *Körkeresés „2”*
18. *Keressük meg G'-ben a még megmaradt rossz köröket.*
19. *Legyen E a megmaradt rossz körök élhalmazainak uniója.*
20. *Legyen $\{1, \ldots, n = |E|\}$ az $E = \{e_1, \ldots, e_n\}$ élhalmaz elemeinek indexhalmaza.*
21. *Legyen $\underline{\underline{A}}'$ az a mátrix, amelynek sorai pont a megmaradt rossz körök karakterisztikus vektorai.*
22. *Legyen $\underline{c}$ az a vektor, amelyre $c_j = W(e_j)\ \forall j = 1, \ldots, n$.*
23. *Körkeresés „2” vége*
24. *Élelhagyás „2”*
25. $L' := \{\sum_{j=1}^{k} \underline{e_j} : k = 1, \ldots, n\}$

26. $\underline{u} := \mathbf{0}$

27. Ciklus, amíg $L' \neq \emptyset$*, és amíg* $\underline{\underline{A}}\underline{u}$ *vektor nem minden koordinátája legalább 1.*

28. $\underline{u} \leftarrow (min\triangleright)L$

29. $L' := L' \cup \{\underline{v} : \underline{v}$ *az* $\underline{u}$ *rákövetkezője*$\}$

30. Ciklus vége

31. Elágazás: Ha az $\underline{u}$ *vektor súlya kisebb, mint az eddigi legjobb megoldásé (*$\underline{opt}$*), akkor:*

32. $\underline{opt} := \underline{u}$

33. Elágazás vége

34. Élelhagyás „2” vége

35. Elágazás vége

36. $L := L \cup \{\underline{v} : \underline{v}$ *az* $\underline{u}$ *rákövetkezője*$\}$

37. Ciklus vége

38. Élelhagyás vége

39. Output: A G gráfból elhagyva az $\underline{opt}$ *vektor egyeseinek megfelelő élek.*

40. Eljárás vége

Ekkor persze lényegesen többet kell dolgozni, de Majlender Péter azért tette javaslatát, mert a gyakorlati problémában a rövid rossz körök kiküszöbölése után lényegesen kevesebb hosszú rossz kör maradt, így tehát remélhető, hogy a gyakorlatban nem veszítünk olyan sokat ezzel a megoldással.

3.2. Élek elhagyása

Akárhogyan is van tárolva az eredeti probléma gráfja, a rossz köröket kereső algoritmusokat már úgy is beprogramozhatjuk, hogy a problémamegoldás második felét adó „élelhagyás” algoritmusunk már az ő számára legmegfelelőbb adatszerkezetben kapja meg az inputját.

3.2.1. A „felsorolásos élelhagyás”

Tehát az „élelhagyás” algoritmust a [3] cikkre és Bruno Simeoni ötletére alapozom: a második sejtés alapján szervezem a „felsorolásos élelhagyás” algoritmusát. Emlékeztetek arra, hogy ekkor valóban szükségünk van a lista rendezetten tartását biztosító adatstruktúrákra, mert ekkor nem feltétlenül lesz a legenerált vektorok közötti első megengedett megoldás optimális. De ennek a vektornak már biztosan elhagyható az összes gyereke, hiszen azok biztosan nagyobb súlyúak, és ezek minden leszármazottja (testvéreikkel együtt) is nagyobb súlyú a fenti vektornál.

6. Algoritmus.

1. *Eljárás*
2. *Input:* $\underline{\underline{A}} \in \{0,1\}^{l\times n}$ *mátrix és* $\underline{c} \in \{0,1\}^n$ *vektor.*
3. $L := \{\sum_{j=1}^{k} \underline{e_j} \,:\, k = 1,\ldots,n\}$
4. $w_{min} := \sum_{j=1}^{n} c_j$
5. $\underline{z} := \mathbf{1}$
6. *Ciklus, amíg* $L \neq \emptyset$
7. $\underline{u} \leftarrow (min\triangleright)L$
8. *Elágazás*
9. *Ha* $\underline{\underline{A}}\,\underline{u} \geq \mathbf{1}$, *akkor:*
10. $w := \underline{uc}$
11. *Elágazás: Ha* $w < w_{min}$, *akkor:*
12. $\underline{z} := \underline{u}$
13. $w_{min} := w$
14. *Elágazás vége*
15. *Ha* $\underline{\underline{A}}\,\underline{u} \not\geq \mathbf{1}$, *akkor:*
16. $L := L \cup \{\underline{v} \,:\, \underline{v}$ *az* $\underline{u}$ *vektor legkisebb gyereke, ha van* $\}$
17. $L := L \cup \{\underline{v} \,:\, \underline{v}$ *az* $\underline{u}$ *vektor legkisebb testvére, ha van* $\}$

18. *Elágazás vége*

19. *Ciklus vége*

20. *Eljárás vége*

Ahol a „legkisebb testvér", illetve a „legkisebb gyerek" fogalmakat a második sejtés értelmében használjuk, és a $\geq$ vektorok között koordinátánként van értelmezve, a $\not\geq$ pedig azt jelenti, hogy „nem minden koordinátára $\geq$".

3. Állítás. *Ez az algoritmus az optimális megoldást találja meg.*

Bizonyítás. A második sejtés bizonyításánál láttuk, hogy ha nem hagynánk figyelmen kívül a lehetséges megoldásokat adó vektorokat, akkor minden vektort felsorolnánk; a bizonyítás után pedig azt is láttuk, hogy ezek leszármazottainak elhagyásával nem hagyjuk ki az optimális megoldást. A fenti algoritmus tehát a bináris vektoroknak egy olyan halmazát vizsgálja végig, amely biztosan tartalmazza az optimális megoldást, ebből a halmazból kiválogatja a lehetséges megoldásokat, és azok közül megkeresi a legkisebb súlyút. □

3.2.2. A „felsorolásos élelhagyás" algoritmus javítása

Az előbbi algoritmus „javítható", hiszen minden olyan vektornak elhagyhatók a leszármazottjai, amely vektornak már most nagyobb a súlya, mint az eddig megtalált lehetséges megoldások közül a legkisebb összsúlyúé.

7. Algoritmus.

1. *Eljárás*

2. *Input:* $\underline{\underline{A}} \in \{0,1\}^{l \times n}$ *mátrix és* $\underline{c} \in \{0,1\}^n$ *vektor.*

3. $L := \{\sum_{j=1}^{k} \underline{e_j} \,:\, k = 1, \ldots, n\}$

4. $w_{min} := \sum_{j=1}^{n} c_j$

5. $\underline{z} := \mathbf{1}$

6. *Ciklus, amíg* $L \neq \emptyset$

7. $\underline{u} \leftarrow (min\triangleright)L$

8. $w := \underline{uc}$

9. *Elágazás: Ha* $w < w_{min}$*, akkor:*

10. Elágazás

11. Ha $\underline{\underline{A}}\underline{u} \not\geq \mathbf{1}$*, akkor:*

12. $L := L \cup \{\underline{v} :$ $\underline{v}$ *az* $\underline{u}$ *vektor legkisebb gyereke, ha van* $\}$

13. $L := L \cup \{\underline{v} :$ $\underline{v}$ *az* $\underline{u}$ *vektor legkisebb testvére, ha van* $\}$

14. Ha $\underline{\underline{A}}\underline{u} \geq \mathbf{1}$*, akkor:*

15. $\underline{z} := \underline{u}$

16. $w_{min} := w$

17. Elágazás vége

18. Elágazás vége

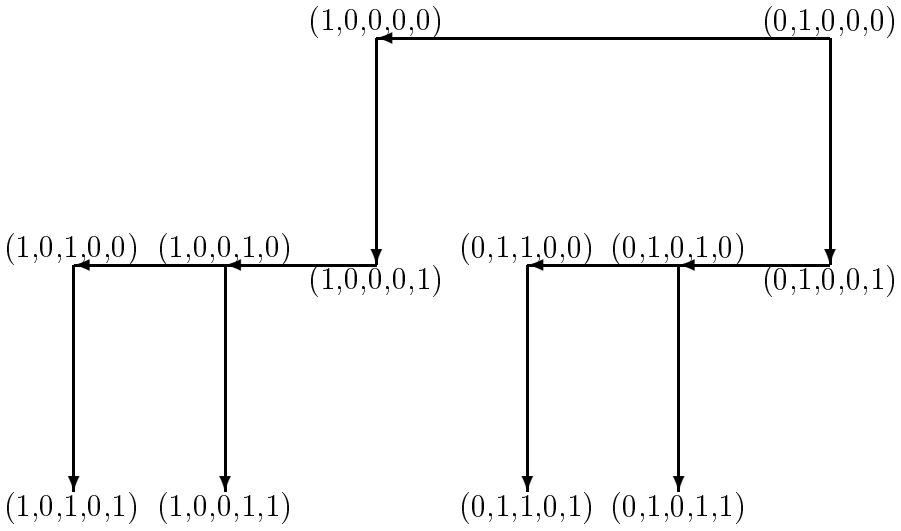
19. Ciklus vége

20. Eljárás vége

Azért tettem idézőjelbe a „javítható” szót, mert ez az algoritmus bizonyos szempontból (látszólag) többet dolgozik az előzőnél: Míg az előző algoritmus (látszólag) csak a lehetséges megoldásoknak számolja ki a súlyát, addig ez az algoritmus minden sorra kerülő vektor súlyát kiszámolja. Azonban ez a megállapítás nem igaz, hiszen a $\triangleright$ rendezés szerinti elsőbbségi sor (az L lista) rendezetten tartásához minden beszúrandó elemnek ki kell számolni a súlyát, és ezt a súlyt nem is olyan nehéz kiszámolni, hiszen minden beszúrandó elem egy már ismert súlyú vektor gyereke, vagy testvére. Most, hogy kicsorbítottam az ellenérv élét, már látható, hogy ez utóbbi algoritmus tényleg jobb az előzőnél: nem számol többször súlyt, viszont kevesebb vektornak képzi gyerekét, illetve testvérét.

3.3. Az eddigi megoldásokból származó ötletek

Az előző algoritmusok *lényegében* gráfot bejáró algoritmusok voltak: $G(V, \overrightarrow{T} \cup \overrightarrow{GY})$ az a gráf, amelynek pontjai az n dimenziós bináris vektorok, és $(\underline{u}, \underline{v}) \in \overrightarrow{GY} \iff \underline{v}$ vektor a $\underline{u}$ vektorból „gyerekképzés” művelettel kapható. $(\underline{u}, \underline{v}) \in \overrightarrow{T} \iff \underline{v}$ vektor a $\underline{u}$ vektorból „testvérképzés” művelettel kapható. A következő ábra egy ilyen gráf egy részletét mutatja, a $\overrightarrow{T}$-beli éleket vízszintes, a $\overrightarrow{GY}$-beli éleket függőleges nyilakkal ábrázoltam:



Ennek a gráfnak a csupa nulla vektort jelentő pontjából indultak az algoritmusok, és megfelelő feltételek teljesülése esetén egy elsőbbségi sorba belevették az éppen vizsgált pontból a fenti gráf irányított élei mentén közvetlenül elérhető pontokat. Egy-egy algoritmus ennek a gráfnak egy-egy részgráfját, méghozzá az elsőbbségi sor használatának köszönhetően részfáját (sőt: részfenyőjét) reprezentálja.

A fent leírtak sokkal általánosabban is igazak, nemcsak az eddig bemutatott algoritmusokat jellemzik. A legtöbb olyan algoritmus, amely bináris vektorok felsorolásával keresi egy halmazfedési, illetve halmazkitöltési probléma optimális megoldását, lényegében egy, a fentihez hasonló gráfot jár be. Az algoritmust elsősorban az általa bejárt gráf jellemzi. A következő fejezetben ezt kihasználva fogok újabb algoritmusokat javasolni.

4. fejezet

Iterációs algoritmusok

Az előző fejezet algoritmusainál a $\overrightarrow{T}$-beli élek által alkotott részgráf összefüggő komponenseit nevezzük el „generációknak". Egy generáción belül a gráf pontjai által reprezentált bináris vektorok ugyanannyi egyest tartalmaznak. Ezen gráfot vizsgálgatva az az ötlet jutott eszembe, hogy elképzelhető lenne olyan algoritmus is, melynek gráfjában szintén lennének generációk (azaz az egyik vektorképző művelettel egy vektorból képezhető vektorok olyan halmazai, amelyeken belül az egyesek száma állandó), de ezeket a generációkat sokkal szorosabb szálak fűznék az inputként kapott előjeles gráfhoz.

4.1. Diszjunkt rendszerek, mint a generációk jellemzői

Jelölje B az inputként kapott előjeles gráf rossz köreinek egy éldiszjunkt rendszerét. Az elhagyandó élek optimális megoldást adó H halmazáról biztosan tudhatjuk, hogy B minden körének élhalmazába belemetsz. Mivel ezen élhalmazok diszjunktak, azt is tudjuk, hogy H előáll oly módon, hogy a B minden körének élhalmazából egy-egy él, és még egyéb élek.

Tehát elképzelhető olyan algoritmus, amelynek működését leíró gráfban egy generációt alkotnak azok a vektorok, amelyek a B rendszer minden körének élhalmazából pontosan egy-egy elem elhagyását jelentik. Sajnos ez még így nem pontos, hiszen a következő generációkat is definiálni kellene. Ez a következőképpen történhet:

Az első generáció lesz az, amit az imént említettünk. Ezen vektorok közül azokból fog kinőni egy-egy új generáció, amelyek nem lehetséges megoldások, ugyanis ha ügyelünk arra, hogy egy generáció egyik tagja se lehessen kisebb súlyú a generáció „ősénél", akkor a lehetséges megoldások leszármazottait nyugodtan figyelmen kívül hagyhatjuk.

Egy új generációt a következő módon hozhatunk létre: A generáció „őse" egy olyan vektor, ami nem lehetséges megoldás, tehát az egyeseinek megfelelő élek elhagyásával nem tűnik el az összes rossz kör. Vegyük a megmaradó rossz köröket, és legyen B_1 ezeknek egy éldiszjunkt rendszere. Az új generáció vektorainak mindegyikében szerepelni fog a generáció „ősének" minden egyese, és még a B_1 minden körének élhalmazából egy-egy élnek megfelelő helyeken egy-egy egyes.

Látható, hogy így automatikusan teljesül az a feltétel, hogy egy generáció minden tagjának súlya legalább akkora, mint a generáció ősének a súlya.

4.2. Az iterációs algoritmusok

Tehát a generációk fenti, iterációs definíciója természetes módon generál egy algoritmusosztályt, amelynek elemei olyan iterációs algoritmusok, amelyek a fent definiált generációk meghatározta gráfokat járják be. Ezen algoritmusok közös tulajdonsága a következő: Keresnek egy éldiszjunkt rossz-kör rendszert és elkezdik legenerálni az ezen rendszer definiálta generáció vektorait. Ezen vektorokat súlyuk szerint növekvő sorrendben végigvizsgálják, és ha valamelyik nem lehetséges megoldás, akkor az egyeseinek megfelelő élek elhagyásával kapott gráfra saját magukat újra meghívják.

Azt reméljük, hogy az előző fejezethez hasonlóan ez a gráf is *fa* lesz, és ezért egy vektor nem fog többször is előfordulni a bejárás során. A következő fejezetben megmutatom, hogy ez hiú remény: ennek a fejezetnek az algoritmusai még gyermekcipőben járnak. De meg fogom adni a javításukat.

4.2.1. Maximális diszjunkt rendszert használó algoritmus

Az iterációs algoritmusok egyik csoportja olyan gráfok bejárását valósítja meg, amelyekben a generációk minél nagyobb számosságúak. Mohó értelemben ilyen gráfokat kaphatunk akkor, ha az új generációk generálásánál valamilyen értelemben maximális diszjunkt rendszereket használunk. Lehet éppen maximális elemszámú éldiszjunkt rendszereket használni, bár a maximális összélszámú éldiszjunkt rendszerek természetesen nagyobb elemszámú generációkat eredményeznek.

Két lehetőség kínálkozik:

- Vagy legeneráljuk egy generáció összes vektorát, majd ezeken súlyuk szerint növekvő sorrendben végigmenve ellenőrizzük, hogy lehetséges

megoldások-e, és ha nem, akkor belemegyünk az őbelőlük kinövő generációba (ha nyilvántartjuk az eddigi legjobb megoldás súlyát, akkor ennek túllépésénél megállhatunk).

- Vagy egy generáció elemeit eleve súlyuk szerint növekvő sorrendben generáljuk (és itt is megállhatunk, ha túllépjük az eddigi legjobb megoldás súlyát).

Ha a diszjunkt rendszer egyetlen rossz körből áll, akkor könnyű megvalósítani a generáció elemeinek súly szerint növékvő sorrendben való generálását: egyszerűen indexük szerint csökkenő sorrendben felsoroljuk az egyetlen rossz kör éleit. Ha viszont több rossz kör alkotja az éldiszjunkt rendszert, akkor több egyes is mozgatható, és annak eldöntése, hogy melyik egyes mozgatásával nő legkevésbbé a súly (hiszen oda kell figyelni arra is, hogy a generációból senkit se hagyjunk ki), többletszámolást igényel, vagy azt is nyilván kell tartani, hogy a súlyvektorban $\underline{c}$ a koordináták különbségeinek mi a növekvő sorrendje.

A fentiek alapján maximális diszjunkt rendszereken alapuló eljárásra a következő algoritmus adódik:

8. Algoritmus. *Diszjunkt Rendszer Keresése ($DR(A) = (\underline{\underline{A'}}, B)$)*

- *Input: $\underline{\underline{A}}$ mátrix, sorai a metszendő objektumok (rossz körök) indikátorvektorai.*
- *Output:*
  - *$\underline{\underline{A'}}$ mátrix: sorai a metszendő objektumok egy maximális éldiszjunkt rendszerének indikátorvektorai*
  - *B indexhalmaz: az $\underline{\underline{A'}}$ mátrix* nem csupa nulla *oszlopainak indexei*

9. Algoritmus. *Ellenőrzés($\underline{v}$)*

- *Input: $\underline{v}$ vektor*
- *Output:*
  - *$\emptyset$ (nulla sorból álló mátrix), ha $\underline{v}$ megengedett megoldás,*
  - *az $\underline{\underline{A}}$ mátrix: sorai azon rossz körök indikátorvektorai, amelyekbe $\underline{v}$ nem metsz bele.*

10. Algoritmus. *Felsorolás($\underline{\underline{A'}}, B$)*

- *Input: DR(A) outputja*

- *Output: Azon vektorok súly szerint rendezett listája, amelyek* $\underline{\underline{A'}}$ *minden sorába pontosan egyszer metszenek bele.*

11. Algoritmus. *Iterációs lépés(*$\underline{v}, \underline{\underline{A}}$*)*
W és $\underline{z}$ *globális változók*

1. $L :=$ *Felsorolás(*$DR(\underline{\underline{A}})$*)*
2. *Ciklus, amíg L ki nem ürül*
3. $\underline{u} \leftarrow L$
4. $\underline{w} := \underline{u} + \underline{v}$
5. *Elágazás*
6. *Ha* $\underline{w}$ *súlya* $< W$*, akkor*
7. *Elágazás*
8. *Ha Ellenőrzés(*$\underline{w}$*)* $= \emptyset$*, akkor:*
9. $W :=$ $\underline{w}$ *súlya és* $\underline{z} := \underline{w}$
10. *egyébként:*
11. $\underline{\underline{C}} :=$ *Ellenőrzés(*$\underline{w}$*)*
12. *Iterációs lépés(*$\underline{w}, \underline{\underline{C}}$*)*
13. *Elágazás vége*
14. *egyébként abortáljuk a ciklust*
15. *Elágazás vége*
16. *Ciklus vége*

Indulás: $\underline{z} = \mathbf{1}$, $W = \sum_{j=1}^{n} c_j$, Iterációs lépés($\mathbf{0}$, rossz körök mátrixa).

Ennek a módszernek az a nagy hátránya, hogy egy teljes generáció összes vektorát generálja, akkor is, ha azokra később sort sem fog keríteni. Ezért érdemes foglalkozni azzal az esettel, amikor a diszjunkt rendszert egyetlen rossz kör alkotja, hiszen akkor a generáció vektorainak felsorolását súly szerint növekvő sorrendben végezhetjük, és ha túlléptük az eddigi legjobb vektor súlyát, akkor meg is állhatunk.

4.2.2. Egy kör generálta iterációs lépések

Ha a diszjunkt rendszert egyetlen rossz kör alkotja, akkor az őáltala meghatározott generációban a vektorokat fel lehet sorolni oly módon, hogy ennek az egyetlen rossz körnek az éleit soroljuk fel súlyuk szerint növekvő sorrendben.

12. Algoritmus. *Diszjunkt Rendszer Keresése ($DR(\underline{\underline{A}}) = (\underline{a}, B)$)*

- *Input: $\underline{\underline{A}}$ mátrix, sorai a metszendő objektumok (rossz körök) indikátorvektorai.*
- *Output:*
  - *$\underline{a}$ vektor: a metszendő objektumok egyikének indikátorvektora*
  - *B indexhalmaz: az $\underline{a}$ vektor* nem nulla *elemeinek indexei*

13. Algoritmus. *Ellenőrzés($\underline{v}$)*

- *Input: $\underline{v}$ vektor*
- *Output:*
  - *$\emptyset$ (nulla sorból álló mátrix), ha $\underline{v}$ megengedett megoldás,*
  - *az $\underline{\underline{A}}$ mátrix: sorai azon rossz körök indikátorvektorai, amelyekbe $\underline{v}$ nem metsz bele.*

14. Algoritmus. *Felsorolás($\underline{a}, j$)*

- *Input: DR($\underline{\underline{A}}$) outputja, és $j \leq B$ szám.*
- *Output: az a vektor, amelyben $\underline{a}$ j-edik egyesének helyén egy egyes áll, a többi eleme nulla.*

15. Algoritmus. *Iterációs lépés($\underline{v}, \underline{a}, B$) W és $\underline{z}$ globális változók*

1. *$L :=$ Felsorolás($\underline{a}, 1$)*
2. *Ciklus j=1 ... B*
3. *$\underline{u} :=$ Felsorolás($\underline{a}, j$)*
4. *$\underline{w} := \underline{u} + \underline{v}$*
5. *Elágazás*
6. *Ha $\underline{w}$ súlya $< W$, akkor:*

7. *Elágazás*
8. *Ha Ellenőrzés($\underline{w}$) = $\emptyset$, akkor*
9. *$W := \underline{w}$ súlya és $\underline{z} := \underline{w}$*
10. *egyébként:*
11. *$\underline{\underline{C}} :=$ Ellenőrzés($\underline{w}$)*
12. *Iterációs lépés($\underline{v}, DR(\underline{\underline{C}})$)*
13. *Elágazás vége*
14. *egyébként abortáljuk a ciklust*
15. *Elágazás vége*
16. *Ciklus vége*

Az áttekinthetőség kedvéért írjuk fel (rekurzív) függvényként az iterációs lépést. Tehát egy iterációs lépés a következő:

$$f(\underline{v}, \underline{\underline{A}}, \underline{z}) = \underline{w}$$

$$\text{ahol } \underline{w} = \begin{cases} \underline{z} & \text{ha az összes, } \underline{\underline{A}}\text{-t kielégítő,} \\ & \underline{v} \text{ minden egyesét tartalmazó vektor} \\ & \text{nehezebb } \underline{z}\text{-nél} \\ \underline{u} & \text{ha } \underline{u} \text{ az } \underline{\underline{A}}\text{-t kielégítő,} \\ & \underline{v} \text{ minden egyesét tartalmazó legkönnyebb vektor,} \\ & \text{ha a súlya kisebb, mint } \underline{z}\text{-é} \end{cases}$$

$g(\underline{v}, \underline{\underline{A}}) = \underline{\underline{A'}}$ az $\underline{\underline{\text{A}}}$ azon sorai, amelyeket $\underline{\text{v}}$ nem elégít ki lehet $\emptyset$ is

$$h(\underline{\underline{A}}, j) = \begin{cases} \underline{e} & \text{az a vektor, amelynek az } \underline{\underline{A}} \\ & \text{mátrix első sorában lévő j-edik egyes oszlopának} \\ & \text{megfelelő koordinátája egyes, a többi nulla} \\ \emptyset & \text{ha } \underline{\underline{A}} \text{ első sorában} \\ & j\text{-nél kevesebb egyes van} \end{cases}$$

$$J(\underline{\underline{A}}) = \text{ az } \underline{\underline{A}} \text{ első sorában lévő egyesek száma}$$

Rekurzió az $\underline{\underline{A}}$ sorainak számára:

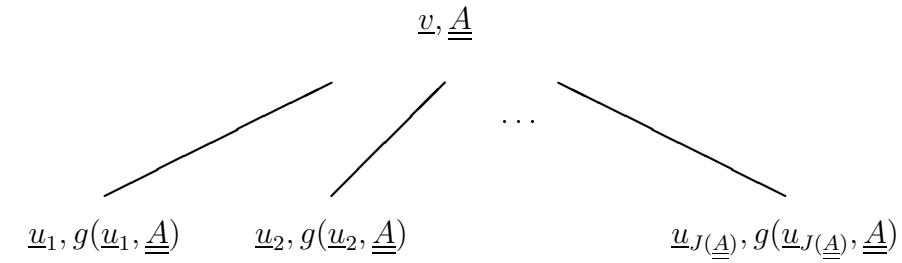
$$f(\underline{v}, \emptyset, \underline{z}) = \begin{cases} \underline{v} & \text{ha } \underline{v} \text{ minden egyesét tartalmazó vektor könnyebb } \underline{z}\text{-nél} \\ \underline{z} & \text{egyébként} \end{cases}$$

$$f(\underline{v},\underline{\underline{A}},\underline{z}) = \begin{cases} \text{Ciklus}(j = 1\text{-től } J(\underline{\underline{A}})\text{-ig}) & \\ & \underline{u} := \underline{v} + h(\underline{\underline{A}}, j) \\ & \underline{z} := f(\underline{u}, g(\underline{u}, \underline{\underline{A}}), \underline{z}) \\ \text{Ciklus vége} & \end{cases}$$

A nulla sorú mátrixra megadtam a függvény értékét, a rekurzív hívások során pedig szigorúan egyre kevesebb sorú mátrixokra hívja meg magát a függvény. Nem eljárásszerűen felírva:

$$f(\underline{v},\underline{\underline{A}},\underline{z}) = min\{f(\underline{v} + h(\underline{\underline{A}}, j), g(\underline{v} + h(\underline{\underline{A}}, j)), \underline{z}) : j = 1 \dots J(\underline{\underline{A}})\}$$

Ekkor az iterációs gráf egy csúcsa:



Ahol $\underline{u}_j = \underline{v} + h(\underline{\underline{A}}, j)$

4. Állítás. *A fenti algoritmus optimális megoldást ad.*

Bizonyítás. Indirekt tegyük fel, hogy $\exists \underline{w}$ lehetséges megoldás, ami *jobb*, mint bármelyik felsorolt. Ekkor $\underline{w}$ belemetsz az $\underline{\underline{A}}$ első sorába. Legyen valamelyik közös él koordinátája i_1. $\underline{w}$ belemetsz az $\underline{\underline{A}}_1 := g(h(\underline{\underline{A}}, i_1), \underline{\underline{A}})$ első sorába is. Az $\underline{\underline{A}}_1$ és $\underline{w}$ valamelyik közös egyesének koordinátája legyen i_2. $\underline{w}$ belemetsz az $\underline{\underline{A}}_2 := g(h(\underline{\underline{A}}, i_1) + h(\underline{\underline{A}}_1, i_2), \underline{\underline{A}})$ első sorába is. Az $\underline{\underline{A}}_2$ és a $\underline{w}$ valamelyik közös egyesének koordinátája legyen i_3. És így tovább. $\underline{w}$ belemetsz az $\underline{\underline{A}}_d := g(\sum_{m=1}^{d} h(\underline{\underline{A}}_{m-1}, i_m), \underline{\underline{A}})$ első sorába is, amennyiben $\underline{\underline{A}}_d$ nem a nulla sorból álló mátrix. (Itt $\underline{\underline{A}}_0 = \underline{\underline{A}}$.) Az $\underline{\underline{A}}_d$ és a $\underline{w}$ valamelyik közös egyesének koordinátája legyen i_{d+1}. Előbb utóbb eljutunk az itrációs gráf egy levelébe (azaz előbb utóbb $\underline{\underline{A}}_d$ a nulla sorból álló mátrix lesz). Az ottani vektor minden egyese a $\underline{w}$ vektornak is egyese, az ottani vektor megoldás és $\underline{w}$ súlya legalább akkora, mint az ő súlya. Ez ellentmond $\underline{w}$ optimalitásának. □

4.3. Probléma az iterációs algoritmusokkal

Mint már említettem, hiú ábránd azt remélni, hogy a fenti iterációs algoritmusok nem sorolnak fel többször is néhány vektort. Ennek bemutatására

vizsgáljuk meg tüzetesebben az iterációs lépéseket. Látható, hogy lehet olyan vektor, amit többször is generálunk: például legyen három rossz kör

$$(0,1,0,0,0,1,0,1,0,0,1)$$

$$(1,0,0,1,1,0,0,0,1,0,1)$$

$$(1,0,1,0,1,0,0,1,0,1,0)$$

karakterisztikus vektorokkal adva. Az első által létrehozott generációban benne lesz

$$(0,0,0,0,0,0,0,0,0,0,1)$$

és

$$(0,0,0,0,0,0,0,1,0,0,0)$$

az első generáció első vektora, mint ős, és a második rossz kör által létrehozott generáció második eleme

$$(0,0,0,0,0,0,0,1,0,0,1)$$

és ez megegyezik a fenti első generáció második vektora, mint ős, és a harmadik rossz kör által létrehozott generáció első elemével.

Ezt ki kellene zárni. Erre adódik a következő lehetőség: Mivel egy vektor, mint ős generálta generációban, illetve az abból leszármazó generációkban minden „értelmes" vektort felsorolunk azok közül, amelyekben az adott vektor egyeseinek helyén egyesek állnak, ezért a bejárandó gráf többi ágában elég azokat a vektorokat generálni, ahol ezeken a helyeken *nem mindenütt* áll egyes.

4.4. Huszárvágás

Most egy merész húzással zárjunk ki sok-sok vektort: az egy $\mathcal{G}$ generációba tartozó vektorok esetén a j-ediknek sorrakerülő ($\underline{v}_j$-vel jelölt) vektor, mint ős által létrehozott új ($\mathcal{G}_j$-vel jelölt) generációból, és a $\mathcal{G}_j$ generáció vektorainak leszármazottjai közül is *hagyjuk ki* azokat a vektorokat, amelyek által meghatározott élhalmazok a $\mathcal{G}$ generációt létrehozó rossz kör első $j-1$ élét tartalmazó élhalmazba belemetszenek. Azaz a bejárandó gráfnak ebben az ágában csak azokat a vektorokat vizsgáljuk, ahol a fenti helyeken *sehol sem* áll egyes.

$$f(\underline{v}, \underline{\underline{A}}, \updownarrow, \underline{z}, \underline{y}) = \underline{w}$$

Itt az $\underline{y}$ az újdonság: ez mondja meg, hogy mely éleket tartalmazó vektorokat ne vegyük figyelembe.

$$g_1(\underline{v}, \underline{\underline{A}}, \underline{y}) = \begin{cases} \underline{\underline{A'}} & \text{az } \underline{\underline{\mathrm{A}}} \text{ azon sorai, amelyeket } \underline{\mathrm{v}} \text{ nem elégít ki} \\ & \textit{de} \text{ ezekből hagyjuk ki az } \underline{y} \text{ egyeseit} \end{cases}$$

$$g_2(\underline{v}, \underline{\underline{A}}, \underline{y}) = \begin{cases} \downarrow & \text{ha az } \underline{\underline{\mathrm{A}}} \text{ sorait } \underline{\mathrm{v}} \text{ nem elégíti ki} \\ \uparrow & \text{ha az } \underline{\underline{\mathrm{A}}} \text{ sorait } \underline{\mathrm{v}} \text{ kielégíti} \end{cases}$$

$$f(\underline{v}, \emptyset, \uparrow, \underline{z}) = \begin{cases} \underline{v} & \text{ha } \underline{v} \text{ vektor könnyebb } \underline{z}\text{-nél} \\ \underline{z} & \text{egyébként} \end{cases}$$

$$f(\underline{v}, \underline{\underline{A}}, \downarrow, \underline{z}, \underline{y}) = \begin{cases} \underline{x} := \underline{y} \\ \text{Ciklus}(j = 1 \text{ től } J(\underline{\underline{A}})\text{-ig}) \\ \qquad\qquad\qquad\qquad \underline{u} := \underline{v} + h(\underline{\underline{A}}, j) \\ \qquad\qquad\qquad\qquad \underline{z} := f(\underline{u}, g(\underline{u}, \underline{\underline{A}}, \underline{x}), \underline{z}, \underline{x}) \\ \qquad\qquad\qquad\qquad \underline{x} := \underline{x} + h(\underline{\underline{A}}, j) \\ \text{Ciklus vége} \end{cases}$$

Ha az f-ben üres ciklus fut (azaz $\underline{\underline{A}}$ minden egyesét kihagytuk az $\underline{y}$ miatt), akkor adja vissza az eddigi legjobb megoldást ($\underline{z}$-t). Joggal hihetnénk azt, hogy ez már olyan durva beavatkozás, hogy az optimális megoldás megtalálását is veszélyezteti. Ennek ellenére igaz a következő állítás:

6. Tétel. *A fenti algoritmus optimális megoldást ad.*

Ennek az állításnak a bizonyítása hasonló a korábbiakhoz.

Bizonyítás. Indirekt tegyük fel, hogy $\exists \underline{w}$ lehetséges megoldás, ami *jobb*, mint bármelyik felsorolt. Ekkor $\underline{w}$ belemetsz az $\underline{\underline{A}}$ első sorába. Legyen az *első* közös él koordinátája i_1. Hangsúlyozom, hogy itt nagyon fontos az, hogy az első közös élet tekintsük. $\underline{w}$ belemetsz a $\underline{\underline{A}}_1 := g(h(\underline{\underline{A}}, i_1), \underline{\underline{A}})$ első sorába is. Ennek megintcsak az *első* közös koordinátája legyen i_2. $\underline{w}$ belemetsz a $\underline{\underline{A}}_2 := g(h(\underline{\underline{A}}, i_1) + h(\underline{\underline{A}}_1, i_2), \underline{\underline{A}})$ első sorába is. Az $\underline{\underline{A}}_2$ és a $\underline{w}$ *első* közös egyesének koordinátája legyen i_3. És így tovább. $\underline{w}$ belemetsz a $\underline{\underline{A}}_d := g(\sum_{m=1}^{d} h(\underline{\underline{A}}_{m-1}, i_m), \underline{\underline{A}})$ első sorába is, amennyiben $\underline{\underline{A}}_d$ nem a nulla sorból álló mátrix. (Itt is $\underline{\underline{A}}_0 = \underline{\underline{A}}$.) Az $\underline{\underline{A}}_d$ és a $\underline{w}$ *első* közös egyesének koordinátája legyen i_{d+1}. Mindenütt vegyük az *első* közös élet.

Valójában az a bizonyítás lényege, hogy mindig az *első* metszéspontot nézzük. Ugyanis $\forall j$: $\exists$ olyan ága az iterációs gráfnak, amelyben az i_j-edik koordináta egyes, és ebben az ágban a meghatározó rossz kör indikátorvektorának az ezt megelőző egyesei ki vannak zárva, de mivel a $\underline{w}$ vektorral közös egyesek között az i_j-edik az első, ezért az adott ágban biztosan csak olyan egyesek vannak kizárva, amely koordináták $\underline{w}$-ben is nullák.

Előbb utóbb eljutunk az itrációs gráf egy levelébe. Az ottani vektor minden egyese a $\underline{w}$ vektornak is egyese, az ottani vektor megoldás és $\underline{w}$ súlya legalább akkora, mint az ő súlya. Ez ellentmond $\underline{w}$ optimalitásának. □

Most, hogy beláttuk az algoritmus helyességét, nézzük, mennyi fölösleges munkát takarítottunk meg:

7. Tétel. *A fenti algoritmus minden vektort* legfeljebb egyszer *sorol fel, azaz az iterációs gráf: fa.*

Bizonyítás. Tegyük fel indirekt, hogy $\underline{v}$ vektorba kettő úton is eljuthatunk az iterációs gráfban. Ekkor ezek az utak valahol szétágaznak: jelölje ennek az iterációs lépésnek a meghatározó rossz körét (pontosabban ezen kör indikátorvektorát) $\underline{a}_l$. Mivel itt ágazik szét a két út, $\underline{v}$ $\underline{a}_l$-be két helyen is belemetsz: $i_{l,1} < i_{l,2}$ helyeken. De az $i_{l,2}$-ik ágban az $i_{l,1}$ koordinátájú helyek kötelezően nullák $\lightning$. □

4.5. Próbálkozás a „Körkeresés” és az „Élelhagyás” integrálására

Az iterációs algoritmusok felvetik annak lehetőségét, hogy a „Körkeresés” és az „Élelhagyás” algoritmusokat összevonjuk, mégpedig a következőképpen:

16. Algoritmus. *Ellenőrzés($\underline{v}$)*

- *Input: $\underline{v}$ vektor*
- *Output:*
  - *$\emptyset$, ha a $G(V, P \cup N)$ gráfból elhagyva a $\underline{v}$ vektor egyeseinek megfelelő éleket nem marad rossz kör;*
  - *a $G(V, P \cup N)$ gráfból elhagyva a $\underline{v}$ vektor egyeseinek megfelelő éleket a maradék gráf egyik rossz körének indikátorvektora;*

17. Algoritmus. *Felsorolás($\underline{a}, j$)*

- *Input: egy $\underline{a}$ vektor, és j szám.*
- *Output: az a vektor, amelyben $\underline{a}$ j-edik egyesének helyén egy egyes áll, a többi eleme nulla.*

18. Algoritmus. *Iterációs lépés($\underline{v}, \underline{a}$)*
W és $\underline{z}$ globális változók

1. $L :=$ *Felsorolás*$(\underline{a}, 1)$
2. *Ciklus* $j = 1 \ldots \underline{a}$ *egyeseinek száma.*
3. $\underline{u} :=$ *Felsorolás*$(\underline{a}, j)$
4. $\underline{w} := \underline{u} + \underline{v}$
5. *Elágazás*
6. *Ha* $\underline{w}$ *súlya* $< W$, *akkor*
7. *Elágazás*
8. *Ha Ellenőrzés*$(\underline{w}) = \emptyset$, *akkor*
9. $W :=$ $\underline{w}$ *súlya és* $\underline{z} := \underline{w}$
10. *egyébként Iterációs lépés*$(\underline{v}$, *Ellenőrzés*$(\underline{w}))$
11. *Elágazás vége*
12. *egyébként abortáljuk a ciklust*
13. *Elágazás vége*
14. *Ciklus vége*

Tehát itt az „Ellenőrzés” nevű lépésbe rejtettem el a „Körkeresés” algoritmusát. Ennek az lehet a hátránya, hogy minden iterációs lépésben végig kell bogarászni a gráfot, és ha jócskán maradt mindkét típusú éle, mégsem maradt rossz köre, ez hosszú ideig tarthat. Ennek ellenére hasonló megfontoláson alapuló algoritmusok még elő fognak kerülni.

5. fejezet

A gyakorlat

5.1. Adatstruktúrák

Ebben a részben azon gondolkodom el, hogy az eddig megadott algoritmusok minél gyorsabb futásához, illetve minél kevesebb tár felhasználásához milyen adatstruktúrák alkalmazása lenne célszerű.

5.1.1. Az eredeti gráf tárolása

A tőzsdei problémában felmerült gráf természetes módon egy négyzetes mátrixszal van megadva, ahol a sorok illetve oszlopok az egyes részvények, a mátrix elemei pedig a meghatározó sornak és oszlopnak megfelelő részvényárfolyamok közötti korreláció. A tőzsdei probléma esetén mindenképpen, de (ha egy meghatározott élsúly érték alatt elhanyagoljuk az éleket) általában is előfordulhatnak nagyon ritka adjecencia mátrixok is, amelyeket célszerűbb lenne szomszédsági listában ([4]) tárolni.

A szomszédsági lista természetesen igencsak megkönnyíti a rossz körök keresését. Ilyen formában megadott gráf esetén esetleg még az élelhagyás és körkeresés integrálásának is több létjogosultsága lehet.

Bár a szomszédsági lista pusztán a gráf tárolására célszerű adatszerkezetnek tűnik, az eredeti felsorolásos algoritmus számára mégis jobb, ha a megtalált rossz körök indikátorvektorait tömbben tároljuk. Az iterációs algoritmus viszont gyorsabb lehet akkor, ha a rossz körök indikátorvektorait láncolt listában tároljuk, legalábbis nagy élszámú gráf viszonylag rövid rossz körei esetén biztosan.

Én azt a megoldást fogom alkalmazni, hogy adjecenciamátrixszal adott gráfban megkeresve a rossz köröket, a rossz köröket már a láncolt adatszerkezethez hasonló módon, mégis tömbökben fogom tárolni: Egy mutatótömb

elemei különböző hasszúságú tömbökre mutatnak, ezen tömbök tartalmazzák az egyes rossz körök egyeseinek indexeit. Mivel [1] eredményei szerint a gyakorlatban túlnyomórészt rövid körök szerepelnek, ezzel a struktúrával tárat nyerhetünk és még a program futását is gyorsabbá tehetjük.

Itt vehetjük nagy hasznát Majlender Péter ötletének. Tároljuk ugyanis a rossz köröket hosszúságuk szerint növekvő sorrendben. Ekkor az egyes vektorok ellenőrzésénél ellenőrizhetjük először a rövid köröket, és az iterációs lépések során próbálkozhatunk először a rövidebb körök kielégítésével.

5.1.2. A körkeresés algoritmusa

A körkeresés algoritmusának megvalósítására a legegyszerűbb mód az, hogy minden egyes negatív élre az egyik végpontjából mondjuk mélységi kereséssel megkeressük az összes utat a másik végpontjába. Ezen az algoritmuson egy nagy kitöltöttségű gráf esetén hiába is javítanánk, maga az output mérete olyan nagy, hogy a kiírása exponenciális idejű.

A rossz körök számát tehát meg kellene becsülnöm. Nézzük a teljes előjeles gráf esetét! K_n-ben a $\{z_i, z_j\}$ élen átmenő körök száma: $n-2$ háromszög, $(n-2)(n-3)$ négyszög, $\ldots$, $(n-2)(n-3)\ldots(n-(k-1))$ k-szög, azaz összesen

$$\sum_{k=1}^{n-2} \frac{(n-2)!}{(n-2-k)!}.$$

K_n-ben a $\{z_i, z_j\}$ és a $\{z_k, z_l\}$ éleken átmenő körök száma:

$$\sum_{l=0}^{n-4} \sum_{k=0}^{n-4-l} \frac{(n-4)!}{(n-4-k-l)!},$$

ha a két él végpontjai különböznek, és

$$\sum_{l=0}^{n-3} \frac{(n-3)!}{(n-3-l)!},$$

ha a két él egyik végpontja egybeesik. Ha két negatív él van, akkor a rossz körök száma:

$$2(\sum_{k=1}^{n-2} \frac{(n-2)!}{(n-2-k)!} - \sum_{l=0}^{n-4} \sum_{k=0}^{n-4-l} \frac{(n-4)!}{(n-4-k-l)!}),$$

ha a két él végponja különböző, és

$$2(\sum_{k=1}^{n-2} \frac{(n-2)!}{(n-2-k)!} - \sum_{l=0}^{n-4} \frac{(n-3)!}{(n-3-l)!}),$$

ha a két élnek van közös végpontja. Legyen

$$M := max\{(\sum_{l=0}^{|V|-3} \frac{(|V|-3)!}{(|V|-3-l)!}), (\sum_{l=0}^{n-4} \sum_{k=0}^{n-4-l} \frac{(n-4)!}{(n-4-k-l)!})\},$$

és legyen

$$m := min\{(\sum_{l=0}^{|V|-3} \frac{(|V|-3)!}{(|V|-3-l)!}), (\sum_{l=0}^{n-4} \sum_{k=0}^{n-4-l} \frac{(n-4)!}{(n-4-k-l)!})\},$$

Ekkor a rossz körök számára felső becslés az

$$|N|(\sum_{k=1}^{n-2} \frac{(n-2)!}{(n-2-k)!} - m),$$

és alsó becslés az

$$|N|(\sum_{k=1}^{n-2} \frac{(n-2)!}{(n-2-k)!} - (|N|-1)M).$$

Tehát teljes gráfban a rossz körök száma legalább

$$|N|(\sum_{k=1}^{|V|-2} \frac{(|V|-2)!}{(|V|-2-k)!}) - |N|(|N|-1)M,$$

és legfeljebb

$$|N|(\sum_{k=1}^{|V|-2} \frac{(|V|-2)!}{(|V|-2-k)!}) - |N|m.$$

Egy él elhagyása a teljes gráfból eggyel növeli azon körök számának együtthatóját, amelyeket nem szabad számolnunk, azaz M, illetve m együtthatóját.

5.1.3. A felsorolásos algoritmusok esete

A felsorolásos algoritmusoknál legfontosabbak az elsőbbségi sort megvalósító struktúrák. [3]-ban az elsőbbségi sort egy olyan bináris fával valósították meg, amelyben egy pont bal gyerekében lévő vektor ▷ benne lévő vektor ▷ jobb gyerekében lévő vektor. És a gyökérnek mindig csak bal gyereke van, így mindig tudjuk, ki a legnagyobb elem a listában. Ezt a szigetvilágok esetére csak annyiban kell módosítani, hogy a gyökérnek mindig csak jobb gyereke legyen.

5.1.4. Az iterációs algoritmusok esete

Az iterációs algoritmusok között lényeges különbséget jelent az, hogy súly szerint növekvő sorrendben tudják-e felsorolni a generációk elemeit. Amennyiben nem (például a maximális éldiszjunkt rendszert használó algoritmus, ami ezt csak körülményesen tudná megtenni), akkor a generáció generálásakor szintén egy elsőbbségi sorba kell írni az elemeket.

Az egyetlen körrel meghatározott generációkat használó algoritmusban viszont nincs szükség elsőbbségi sorra, annál inkább egy-egy láncolt listára, ami a meghatározó rossz körök éleit súly szerint növekedő sorrendben tartalmazza. Mint fent írtam, ezt nem láncolt listával, hanem különböző hosszú tömbökkel fogom megoldani.

5.2. Programok gyakorlati tesztelése

Mivel nincs saját számítógépem, ezért az algoritmusok gyakorlati tesztelésére két természetes út volt: vagy a „cs.elte.hu" hálózat gépeit használom, vagy számítógéppel rendelkező évfolyamtársaim segítségét kérem. Mivel ez egy NP-teljes probléma, és a programokba is kerülhet hiba, ezért könnyen előfordulhatnak problémák (elő is fordultak). Ezen problémáktól megkímélendő a „cs" hálózatot, Joó Dániel[1] első éves fizikus évfolyamtársam számítógépét használtuk.

5.2.1. Az előkészítés

Először is egy adjecenciamátrixszal adott előjeles gráfból építünk éllistákat, külön a negatív élekre (ezeket elég egyszer tárolni) és külön a pozitív élekre (ezeket célszerű minden végpontjuknál felsorolni). Az adjecenciamátrixban a pontpároknak megfelelő helyen az adott él előjeles súlyát várjuk inputként (illetve, ha ott nincs él, akkor egy nullát várunk).

5.2.2. A körkeresés és az élelhagyás

A rossz körök megkeresését a következő módon valósítjuk meg: Minden negatív élre a pozitív éleken történő mélységi kereséssel megkeressük az összes utat az egyik végpontjából a másikba. Ez az algoritmus nagy kitöltöttségű gráfok esetén irdatlan nagy outputot produkál. (Minden, az összes rossz kört megkereső algoritmus ezt tenné.)

[1] Neki, és a szintén első éves fizikus Varga Lászlónak ezúton is szeretném megköszönni az algoritmusaim gyakorlati kipróbálásában nyújtott segítséget.

A körkeresés programjának kipróbálása során „saját bőrömön tapasztalhattam meg" azt, milyen nagy jelentősége van Majlender Péter ötletének. Hiszen komolyabb méretű, nagy kitöltöttségű gráfokkal nem tudnánk megbírkózni. Négy lehetőségünk van:

- Vagy már az input mátrixban elhanyagoljuk a viszonylag kicsi korrelációkat,
- vagy az eredeti ötlet alapján először megkeressük a rövid rossz köröket, azokat kiküszöböljük, és az optimális megoldásra megkeressük a hosszabb rossz köröket, majd őket is kiküszöböljük.
- A harmadik lehetőség az, hogy a rövid rossz köröket megkeressük, azokra lejátszuk az iterációs algoritmust, és az iterációs fa minden levelében az ottani megengedett megoldásra (amennyiben az kisebb súlyú, mint az eddig megtalált „globális" optimális megoldás), megkeressük a megmaradt rossz köröket, és onnan indítva újra indítjuk az iterációs algoritmust.
- A negyedik lehetőség a körkeresés és élelhagyás teljes integrálása.

A harmadik megoldás ötlete lényegében az 5. algoritmus ötletének alkalmazása az iterációs módszerre. Itt a „huszárvágást" is lehet alkalmazni, mert ez a módszer úgy működik, *mintha* megkerestünk volna minden rossz kört, csak éppen a megengedettség ellenőrzésénél először a rövid körök kielégítését vizsgálnánk, tehát a „tiltott helyek" az iterációs gráf leveléből, mint gyökérből induló újabb iterációs gráfban is „tiltott helyként" élnek tovább.

A harmadik módszer hátránya, hogy az első iterációs gráf (majdnem) minden levelében újra le kell futtatni a körkeresés algoritmusát (csak azokban a levelekben, ahol remény lehet az eddigi legjobbnál kisebb súlyú megoldás megtalálására), de ezen lehet spórolni! A rövid körök iterációs gráfjában egymás után felsorolt, a rövid körökre nézve lehetséges megoldások által meghagyott hosszú rossz körök halmazai között lehet átfedés, és ezt a következő levélben a körkeresésnél kihasználhatjuk.

5.2.3. A tesztelt algoritmusok

A gyakorlatban két algoritmust valósítottunk meg:

- Egy olyan algoritmust, amely inputként várja azt is, hogy milyen csoportosításban keresse meg a rossz köröket. Ezeket a csoportokat blokkoknak neveztük el. Az algoritmus először megkeresi az inputként megkapott „blokkosító vektor" első eleménél rövidebb rossz köröket, azokra

lejátssza az iterációs algoritmust, és az iterációs fa minden levelében az ottani megengedett megoldásra (amennyiben az kisebb súlyú, mint az eddig megtalált „globális" optimális megoldás), megkeresi a megmaradt rossz körök közül azokat, amelyek az inputként megkapott „blokkosító vektor" második eleménél rövidebbek. Innen indítva újra elkezdi az iterációs algoritmust, és az iterációs fa minden levelében az ottani megengedett megoldásra (amennyiben az kisebb súlyú, mint az eddig megtalált „globális" optimális megoldás), megkeresi a megmaradt rossz körök közül azokat, amelyek az inputként megkapott „blokkosító vektor" harmadik eleménél rövidebbek. És így tovább. A „blokkosító vektor" utolsó eleme nyilván az input gráf pontszáma. Ezt az algoritmust többféleképpen hívtuk meg.

- A másik algoritmussal Majlender Péter eredeti ötlete alapján először megkeressük a rövid rossz köröket, azokat kiküszöböljük, és az optimális megoldásra megkeressük a hosszabb rossz köröket, majd őket is kiküszöböljük. Ezt közelítésnek nevezzük, és azt vizsgáljuk mennyire jól közelíti az optimális megoldást.

A függelékben blocktipus/Közelítés jelöli azt, hogy melyik algoritmus eredménye következik.

5.2.4. A tesztelő gráfok

A tesztelő gráfoknak négy csoportját generáltuk:

- A pozitív és negatív élek aránya közel megegyező (ezek szerepelnek az „A" függelékben)
  - teljes kitöltöttségű
  - 10%-os kitöltöttségű
- A negatív élek aránya közel 20%-os (ezek szerepelnek a „B" függelékben)
  - 50%-os kitöltöttségű
  - 10%-os kitöltöttségű

Ezeket a paramétereket (a másik szakján közgazdászhallgató) Varga László javasolta, mert a tőzsdén sok részvény között nulla a korreláció, és a pozitív korreláció jellemzőbb a negatívnál.

A függelékben „dim" jelzi, hogy a vizsgált mátrix hányszor hányas.

A függelék a teszelő program outputja, ezért csak az utólag beleírt szavak tartalmaznak ékezeteket.

Irodalomjegyzék

[1] Peter L. Hammer, Majlender Péter, Bruno Simeoni, Vizvári Béla; Ali Tajdar, *Report on the Results of the RUTCOR Research Group for Dow Jones Stocks*

[2] Chartrand, G., *Introductory Graph Theory*, Dover Publications Inc., New York, 19xx., reprint, eredeti: *Graphs as mathematical models* , 1977.

[3] Vizvári Béla, Fatih Yilmaz, *An Ordering (Enumerative) Algorithm for Nonlinear 0-1 Programming*, Global Optimalization 5(1994), 277-291.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Algoritmusok*, Műszaki könyvkiadó, Budapest, 1999., eredeti: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, *Introduction to algorithms* The Massachusetts Institute of Technology, 1990.

A. Függelék

A.1. 10%-os kitöltöttségű input mátrixok

A.1.1.

dim:20
blocktipus:0
runtime:0.0549450549 sec
iteracios lepesek szama 3
megoldas sulya:0.243000

dim:20
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 5
megoldas sulya:0.243000

dim:20
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 5
megoldas sulya:0.243000

dim:20
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 37
megoldas sulya:0.243000

dim:20
Közelítés
runtime:0.0549450549 sec
iteracios lepesek szama 4
megoldas sulya:0.243000

dim:33
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 28
megoldas sulya:0.870000

dim:33
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 32
megoldas sulya:0.870000

dim:33
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 30
megoldas sulya:0.870000

dim:33
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 102
megoldas sulya:0.870000

dim:33
Közelítés
runtime:0.0000000000 sec
iteracios lepesek szama 83
megoldas sulya:2.824000

A.1.2.

dim:20
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 1
megoldas sulya:0.000000

dim:20
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 2
megoldas sulya:0.000000

dim:20
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 2
megoldas sulya:0.000000

dim:20
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 18
megoldas sulya:0.000000

dim:20
Közelítés
runtime:0.0549450549 sec
iteracios lepesek szama 2
megoldas sulya:0.000000

dim:33
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 27
megoldas sulya:0.968000

dim:33
blocktipus:1
runtime:0.0549450549 sec
iteracios lepesek szama 13
megoldas sulya:0.968000

dim:33
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 30
megoldas sulya:0.968000

dim:33
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 45
megoldas sulya:0.968000

dim:33 Közelítés
runtime:0.0000000000 sec
iteracios lepesek szama 8
megoldas sulya:1.203000

A.2. Teljes kitöltöttségű input mátrixok

A.2.1.

dim:6
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 5
megoldas sulya:0.646000

dim:6
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 6
megoldas sulya:0.646000

dim:6
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 6
megoldas sulya:0.646000

dim:6
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 8
megoldas sulya:0.646000

dim:6
Közelítés
runtime:0.0000000000 sec

iteracios lepesek szama 6
megoldas sulya:0.743000

dim:7
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 305
megoldas sulya:2.172000

dim:7
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 307
megoldas sulya:2.172000

dim:7
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 294
megoldas sulya:2.172000

dim:7
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 319
megoldas sulya:2.172000

dim:7
Közelítés
runtime:0.0000000000 sec
iteracios lepesek szama 2561
megoldas sulya:5.628000

dim:8
blocktipus:0
runtime:0.0549450549 sec
iteracios lepesek szama 2035
megoldas sulya:2.740000

dim:8
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 1650
megoldas sulya:2.740000

dim:8
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 1650
megoldas sulya:2.740000

dim:8
blocktipus:3
runtime:0.0549450549 sec
iteracios lepesek szama 1417
megoldas sulya:2.740000

dim:8
Közelítés
runtime:3.0219780220 sec
iteracios lepesek szama 139484
megoldas sulya:6.848000

A.2.2.

dim:6
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 12
megoldas sulya:0.734000

dim:6
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 14
megoldas sulya:0.734000

dim:6
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 9

megoldas sulya:0.734000

dim:6
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 11
megoldas sulya:0.734000

dim:6
Közelítés
runtime:0.0000000000 sec
iteracios lepesek szama 41
megoldas sulya:3.500000

dim:7
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 34
megoldas sulya:0.967000

dim:7
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 43
megoldas sulya:0.967000

dim:7
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 27
megoldas sulya:0.967000

dim:7
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 30
megoldas sulya:0.967000

dim:7
Közelítés
runtime:0.0549450549 sec
iteracios lepesek szama 741
megoldas sulya:6.920000

dim:8
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 620
megoldas sulya:1.890000

dim:8
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 604
megoldas sulya:1.890000

dim:8
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 604
megoldas sulya:1.890000

dim:8
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 247
megoldas sulya:1.890000

dim:8
runtime:0.6043956044 sec
iteracios lepesek szama 35676
megoldas sulya:6.821000

dim:9
blocktipus:0
runtime:0.5494505495 sec
iteracios lepesek szama 5532
megoldas sulya:2.685000

dim:9
blocktipus:1
runtime:0.0549450549 sec

iteracios lepesek szama 3711
megoldas sulya:2.685000

dim:9
blocktipus:2
runtime:0.0549450549 sec
iteracios lepesek szama 3711
megoldas sulya:2.685000

dim:9
blocktipus:3
runtime:0.0549450549 sec
iteracios lepesek szama 2968
megoldas sulya:2.685000

A.2.3.

dim:6
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 20
megoldas sulya:1.022000

dim:6
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 22
megoldas sulya:1.022000

dim:6
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 24
megoldas sulya:1.022000

dim:6
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 30
megoldas sulya:1.022000

dim:6
Közelítés
runtime:0.0000000000 sec
iteracios lepesek szama 226
megoldas sulya:3.564000

dim:7
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 105
megoldas sulya:1.761000

dim:7
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 97
megoldas sulya:1.761000

dim:7
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 129
megoldas sulya:1.761000

dim:7
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 160
megoldas sulya:1.761000

dim:7
Közelítés
runtime:0.0549450549 sec
iteracios lepesek szama 3081
megoldas sulya:7.789000

dim:8
blocktipus:0
runtime:0.0549450549 sec
iteracios lepesek szama 357
megoldas sulya:2.266000

dim:8
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 298
megoldas sulya:2.266000

dim:8
blocktipus:2
runtime:0.0549450549 sec
iteracios lepesek szama 298
megoldas sulya:2.266000

dim:8
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 512
megoldas sulya:2.266000

dim:8
Közelítés
runtime:1.0989010989 sec
iteracios lepesek szama 51929
megoldas sulya:8.577000

dim:9
blocktipus:0
runtime:3.4615384615 sec
iteracios lepesek szama 17301
megoldas sulya:3.963000

dim:9
blocktipus:1
runtime:0.2197802198 sec
iteracios lepesek szama 9465
megoldas sulya:3.963000

dim:9
blocktipus:2
runtime:0.1648351648 sec
iteracios lepesek szama 9465
megoldas sulya:3.963000

dim:9
blocktipus:3
runtime:0.2747252747 sec
iteracios lepesek szama 12716
megoldas sulya:3.963000

B. Függelék

B.1. 10%-os kitöltöttségű input mátrixok

B.1.1.

dim:20
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 1
megoldas sulya:0.000000

dim:20
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 2
megoldas sulya:0.000000

dim:20
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 2
megoldas sulya:0.000000

dim:20
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 18
megoldas sulya:0.000000

dim:20
Közelítés
runtime:0.0000000000 sec
iteracios lepesek szama 2
megoldas sulya:0.000000

dim:30
blocktipus:0
runtime:0.0549450549 sec
iteracios lepesek szama 313
megoldas sulya:0.874000

dim:30
blocktipus:1
runtime:0.0549450549 sec
iteracios lepesek szama 316
megoldas sulya:0.874000

dim:30
blocktipus:2
runtime:0.0549450549 sec
iteracios lepesek szama 327
megoldas sulya:0.874000

dim:30
blocktipus:3
runtime:0.1648351648 sec
iteracios lepesek szama 234
megoldas sulya:0.874000

dim:30
Közelítés
runtime:0.0000000000 sec
iteracios lepesek szama 5
megoldas sulya:0.190000

B.1.2.

dim:20
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 4
megoldas sulya:0.604000

dim:20
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 5
megoldas sulya:0.604000

dim:20
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 5
megoldas sulya:0.604000

dim:20
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 21
megoldas sulya:0.604000

dim:20
Közelítés
runtime:0.0000000000 sec
iteracios lepesek szama 5
megoldas sulya:0.669000

dim:30
blocktipus:0
runtime:0.0549450549 sec
iteracios lepesek szama 479
megoldas sulya:1.400000

dim:30
blocktipus:1
runtime:0.2197802198 sec
iteracios lepesek szama 282
megoldas sulya:1.400000

dim:30
blocktipus:2
runtime:0.0549450549 sec
iteracios lepesek szama 474
megoldas sulya:1.400000

dim:30
blocktipus:3
runtime:0.2197802198 sec
iteracios lepesek szama 386
megoldas sulya:1.441000

dim:30
Közelítés
runtime:0.0000000000 sec
iteracios lepesek szama 25
megoldas sulya:1.581000

B.2. 50%-os kitöltöttségű input mátrixok

B.2.1.

dim:8
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 34
megoldas sulya:1.031000

dim:8
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 21
megoldas sulya:1.031000

dim:8
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 21
megoldas sulya:1.031000

dim:8
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 33
megoldas sulya:1.031000

dim:8
Közelítés

runtime:0.0000000000 sec
iteracios lepesek szama 9
megoldas sulya:1.220000

dim:11
blocktipus:0
runtime:1.3736263736 sec
iteracios lepesek szama 6024
megoldas sulya:2.576000

dim:11
blocktipus:1
runtime:0.1648351648 sec
iteracios lepesek szama 897
megoldas sulya:2.576000

dim:11
blocktipus:2
runtime:0.1098901099 sec
iteracios lepesek szama 4064
megoldas sulya:2.576000

dim:11
blocktipus:3
runtime:0.1098901099 sec
iteracios lepesek szama 1689
megoldas sulya:2.576000

dim:11
Közelítés
runtime:0.6043956044 sec
iteracios lepesek szama 32997
megoldas sulya:7.728000

B.2.2.

dim:8
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 29
megoldas sulya:0.938000

dim:8
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 17
megoldas sulya:0.938000

dim:8
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 17
megoldas sulya:0.938000

dim:8
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 46
megoldas sulya:0.938000

dim:8
Közelítés
runtime:0.0000000000 sec
iteracios lepesek szama 28
megoldas sulya:2.019000

dim:11
blocktipus:0
runtime:2.3076923077 sec
iteracios lepesek szama 6811
megoldas sulya:3.160000

dim:11
blocktipus:1
runtime:0.0549450549 sec
iteracios lepesek szama 1926
megoldas sulya:3.160000

dim:11
blocktipus:2
runtime:0.0549450549 sec

iteracios lepesek szama 4339
megoldas sulya:3.160000

dim:11
blocktipus:3
runtime:0.1098901099 sec
iteracios lepesek szama 1185
megoldas sulya:3.160000

dim:11
Közelítés
runtime:5.6593406593 sec
iteracios lepesek szama 269612
megoldas sulya:10.600000

B.2.3.

dim:8
blocktipus:0
runtime:0.0000000000 sec
iteracios lepesek szama 23
megoldas sulya:0.889000

dim:8
blocktipus:1
runtime:0.0000000000 sec
iteracios lepesek szama 25
megoldas sulya:0.889000

dim:8
blocktipus:2
runtime:0.0000000000 sec
iteracios lepesek szama 25
megoldas sulya:0.889000

dim:8
blocktipus:3
runtime:0.0000000000 sec
iteracios lepesek szama 33
megoldas sulya:0.889000

dim:8
Közelítés
runtime:0.0000000000 sec
iteracios lepesek szama 12
megoldas sulya:0.889000

dim:11
blocktipus:0
runtime:2.3626373626 sec
iteracios lepesek szama 1190
megoldas sulya:2.364000

dim:11
blocktipus:1
runtime:0.0549450549 sec
iteracios lepesek szama 361
megoldas sulya:2.364000

dim:11
blocktipus:2
runtime:0.0549450549 sec
iteracios lepesek szama 490
megoldas sulya:2.364000

dim:11
blocktipus:3
runtime:0.0549450549 sec
iteracios lepesek szama 218
megoldas sulya:2.364000

dim:11
Közelítés
runtime:0.1098901099 sec
iteracios lepesek szama 6546
megoldas sulya:7.762000