

Szegedi Tudományegyetem

Informatikai Tanszékcsoport

Képfeldolgozás és Számítógépes Grafika Tanszék

Empirikus vizsgálatok hv-konvex bináris mátrixok két vetületből történő rekonstrukciójára

TDK dolgozat

Készítette:
Ozsvár Zoltán
programtervező
informatikus szakos
hallgató

Témavezető:
Dr. Balázs Péter
egyetemi adjunktus

Szeged
2012

Tartalomjegyzék

Tartalomjegyzék.....	2
1. TOMOGRÁFIÁRÓL ÁLTALÁBAN	4
1.1. A tomográfia	4
1.2. Diszkrét tomográfia	5
1.3. Bináris tomográfia	6
1.4. Alapfogalmak	6
1.5. Bináris képek speciális geometriai tulajdonságai	8
2. ALGORITMUSOK.....	9
2.1.1. Mag-burok algoritmus.....	9
2.1.2. Megvalósítás tömbös adatszerkezettel.....	12
2.1.3. Megvalósítás First-Last adatszerkezettel	13
2.2.1. Szimulált hűtésen alapuló rekonstrukciós algoritmus	15
2.2.2. Ryser algoritmus	15
2.2.3. Szimulált hűtés	17
2.3.1. Naiv algoritmus	19
3. AZ EREDMÉNYEK.....	21
3.1. Rekonstrukció pontossága	22
3.2. Futási idők	25
3.3. Összegzés	29
Irodalomjegyzék	30

Tartalmi összefoglaló

A tomográfia célja egy adott objektum belsejének a vizsgálata annak jelentős roncsolása nélkül. Amennyiben a vizsgált tárgy homogén, úgy a *bináris tomográfiát* hívhatjuk segítségül a feladat megoldására. Mivel a kevés vetületből való képalkotás jelentősen aluldefiniált, így – a feladat megkönnyítése érdekében – általában feltételezzük, hogy a vizsgált objektum rendelkezik valamilyen geometriai tulajdonsággal. Ezek közül az egyik leggyakrabban vizsgált feltétel a *hv*-konvexitás.

Bizonyított, hogy a *hv*-konvex mátrixok vízszintes és függőleges vetületeiből való rekonstruálása NP-teljes probléma. A dolgozat során azt vizsgáljuk, hogy miben rejlik a probléma nehézsége. Különböző adatszerkezeteket alkalmazó mag-burok és szimulált hűtésen alapuló rekonstrukciós algoritmusok teljesítményét vizsgáljuk. Arra a kérdésre keressük a választ, hogy hogyan befolyásolja a kép mérete és annak komponenseinek száma a futási időt, valamint a rekonstrukció pontosságát.

A teszteket egy nagyméretű egyenletes eloszlásból származó adathalmazon hajtottuk végre, mely különböző méretű és komponensszámú képeket tartalmazott. Kutatásaink során megállapítottuk, hogy mindegyik algoritmus esetén az eredeti és rekonstruált képek közötti különbségek sokkal inkább a komponensek számától, mint a kép méretétől függnének. A futási időket vizsgálva a mag-burok algoritmus a képek méreteire és a komponensek számára egyaránt érzékeny, a szimulált hűtéses rekonstrukciós algoritmus sebessége viszont inkább az úgy nevezett kapcsoló komponensek számától függ.

1. A TOMOGRÁFIÁRÓL ÁLTALÁBAN

1.1. A tomográfia

A tomográfia a görög „tomos” (szelet) szóból ered, és szeleteken alapuló képalkotást jelent. A tomográfia feladata egy adott objektum belsejének vizsgálata annak jelentős roncsolása nélkül. Ennek eléréséhez a rekonstruálandó tárgy kétdimenziós szeleteinek előállítása szükséges. Ez után a kapott szeletekből előállítható a 3 dimenziós objektum leírása. Matematikailag az objektum modelljét (a leírást) a keresztmetszeti képek (szeletek) bizonyos irányokból vett vonal menti integráljaiból (vetületeiből) kapjuk. A gyakorlati megvalósítás során valamilyen sugárzással (Röntgen-, elektronsugárzás, stb.) „világítjuk át” a testet. A sugarak a megfigyelt tárgyon különböző mértékben elnyelődve hatolnak át, és egy detektorsorba érkeznek, amely méri azok intenzitásait. A rendelkezésre álló adatok alapján a detektor által mért I intenzitás és a kezdeti I_0 intenzitás közötti összefüggés a fizikából ismert Beer-Lamber törvény által adható meg.

$$I = I_0 \cdot e^{-\mu \Delta x} ,$$

Ezek által meghatározhatjuk azt, hogy milyen Δx vastagságú μ lineáris elnyelődési együtthatójú anyagon haladt át a sugár. A fenti képlet csak homogén tárgyak esetén alkalmazható, viszont a következő összefüggés segítségével $\mu_1, \mu_2, \dots, \mu_n$ elnyelődési együtthatójú, de azonos Δx vastagságú anyagon áthaladó sugárzás esetén is számolhatunk.

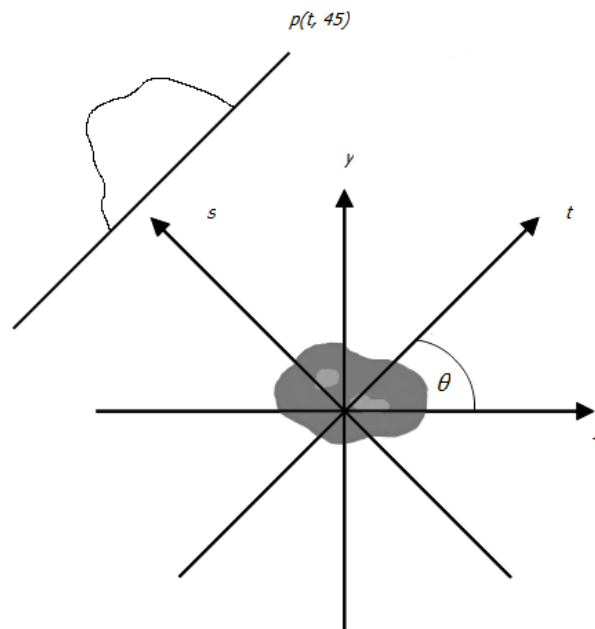
$$I = I_0 \cdot e^{-\sum_{i=1}^n \mu_i \Delta x} .$$

Folytonos modellt esetén pedig a következő képletet alkalmazhatjuk

$$I = I_0 \cdot e^{-\int_0^d \mu(x) dx} ,$$

ahol d a Röntgen-forrás és a detektor közötti távolság, x az adott sugárnyalábon vett távolság a forrástól számítva, $\mu(x)$ pedig az elnyelődési együttható értékét jelöli az x pontban.

Legyen $f(x, y)$ az a függvény, ami a vizsgált objektum elnyelődési együtthatóit reprezentálja tetszőleges (x, y) pontban. Jelölje t az x , s pedig az y tengely megadott θ szögű elforgatottját. Rögzített θ esetén az $f(x, y)$ függvény θ szögű *vetületének* nevezzük a $p(t, \theta) = \int_{-\infty}^{\infty} f(t \cdot \cos \theta - s \cdot \sin \theta, t \cdot \sin \theta + s \cdot \cos \theta) ds$ függvényt (1. ábra).



1. ábra. A két változós függvény és a $\theta = 45^\circ$ szög által meghatározott vetülete

A képrekonstrukció során adott vetületekből kell előállítanunk az eredeti függvényt. A komputeres tomográfia legismertebb alkalmazásai az orvosi képalkotásban találhatók meg, de az eljárás alkalmazható még különböző ipari vizsgálatoknál, földtani anyagvizsgálataknál és számos más területen is [1].

1.2. Diszkrét tomográfia

Egy kép elkészítéséhez több száz vetület kinyerésére lehet szükség, ami a gyakorlatban nem mindig lehetséges a számítások időigényessége vagy a vizsgálat roncsoló hatása miatt. Az ilyen esetekben szükségünk van arra, hogy különböző feltételek biztosításával csökkentsük a megoldáshoz szükséges vetületek számát.

A *diszkrét tomográfia* akkor alkalmazható, ha a vizsgálandó objektum csak néhány előre ismert elnyelési együtthatójú anyagból áll, így a rekonstruálandó képen csak néhány különböző szürke intenzitás jelenhet meg. Ennek a plusz információnak a segítségével sok esetben sokkal kevesebb vetületből is megadható megfelelő minőségű kép. A diszkrét

tomográfia olyan alkalmazásoknál használatos, ahol a nagy vetületszám elérése fizikai korlátokba ütközhet, költséges lehet, vagy károsíthatja a vizsgált objektumot. Ebben az esetben a hagyományos rekonstrukciós algoritmusok nem használhatók, mivel nem adnak kielégítő minőségű képet.

1.3. Bináris tomográfia

A *bináris tomográfia* a diszkrét tomográfia speciális esete, ahol a vizsgált kép bináris, tehát csak két értéket tartalmazhat. A képeken 0 és 1 jelölik rendre az anyag hiányát vagy jelenlétét az adott pozícióban. A megszorításoknak köszönhetően sokszor már 2 vetülettel is jó eredményeket lehet kapni. A megoldáshalmaz méretének csökkentése érdekében kiegészítő korlátokat szokás meghatározni, amelyek a vizsgált kép geometriai tulajdonságaira vonatkozhatnak. Tehát a rekonstruált mátrixnak (képnek) ki kell elégítenie az előre megadott sor és oszlop összegek (vetületeket) mellett geometriai feltételeket is. Bizonyos geometriai tulajdonságok esetén létezik polinom idejű rekonstrukciós algoritmus, míg más esetekben a rekonstrukció NP-nehéz. A bináris tomográfia legfontosabb alkalmazásai az orvosi képalkotásban (kontrasztanyag használatával), különböző ipari alkatrészek nem roncsoló vizsgálataiban valamint, elektromikroszkópos vizsgálatoknál merülnek fel.

1.4. Alapfogalmak

A továbbiak megértéséhez a bináris tomográfia alábbi fogalmainak ismerete szükséges. Egy tetszőleges bináris kép ábrázolható a $\mathbb{Z}_2$ 2-dimenziós egészrács egy véges halmazával, melyet *diszkrét halmaznak* nevezünk. A bináris képet 2-dimenziós bináris mátrix segítségével is ábrázolhatjuk. A továbbiakban tegyük fel, hogy $m \times n$ -es bináris mátrixokkal dolgozunk, ahol m, n jelölik rendre a mátrix sorainak és oszlopainak a számát. Legyen A egy $m \times n$ méretű bináris mátrix. Az A bináris mátrix *horizontális vetületén* az $R(A) = (r_1, \dots, r_m)$ vektor, *vertikális vetületén* pedig az $S(A) = (s_1, \dots, s_n)$ vektort értjük, ahol

$$r_i = \sum_{j=1}^n a_{ij} \quad (i = 1, \dots, m)$$

$$s_j = \sum_{i=1}^m a_{ij} \quad (j = 1, \dots, n)$$

Jelölje $\mathcal{G}$ az $m \times n$ -es mátrixok egy adott osztályát. A rekonstrukciós feladat a következőképpen fogalmazható meg.

Rekonstrukció ($\mathcal{G}$): Legyenek adottak az $m, n \in \mathbb{N}$ egész számok és az $R \in \mathbb{N}^m$ és $S \in \mathbb{N}^n$ vektorok. Konstruáljunk olyan $A \in \mathcal{G}$ $m \times n$ -es mátrixot, melyre $R(A) = R$ és $S(A) = S$.

Ehhez a feladathoz szorosan kapcsolódik az egyértelműség eldöntésének kérdése is.

Unicitás ($\mathcal{G}$): Legyenek adottak az $m, n \in \mathbb{N}$ egész számok és egy $A \in \mathcal{G}$ $m \times n$ méretű mátrix. Létezik-e olyan $A' \neq A$ $\mathcal{G}$ -beli bináris mátrix, melyre $R(A') = R(A)$ és $S(A') = S(A)$.

1.3. Definíció: Egy bináris mátrix $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ és $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ részmátrixait kapcsolókomponenseknek nevezzük. A kapcsolókomponensek pozícióiban lévő értékeket invertálva egy olyan bináris mátrixot kapunk, melynek horizontális és vertikális vetületei ugyanazok, mint az eredeti mátrixnak. Ezt a tulajdonságot tekintve, ha egy bináris mátrix tartalmaz kapcsoló komponenset tartalmaz, akkor és csak akkor nem egyértelmű az összes diszkrét halmazt tartalmazó osztályra nézve. Több kapcsoló komponensnek lehet közös eleme is (2. ábra). Ismert továbbá az alábbi feltétel.

1.1. Tétel: Legyenek A és B bináris mátrixok, melyek horizontális és vertikális vetületei megegyeznek és $B \neq A$. Ekkor az A mátrixból el lehet jutni B mátrixba (vagy fordítva) kapcsoló komponensek véges számú cseréjével.

1	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	0	1	1
0	0	1	1	1

2. ábra. Több kapcsoló komponensnek lehet közös eleme

1.5. Bináris képek geometriai tulajdonságai: összefüggés és konvexitás

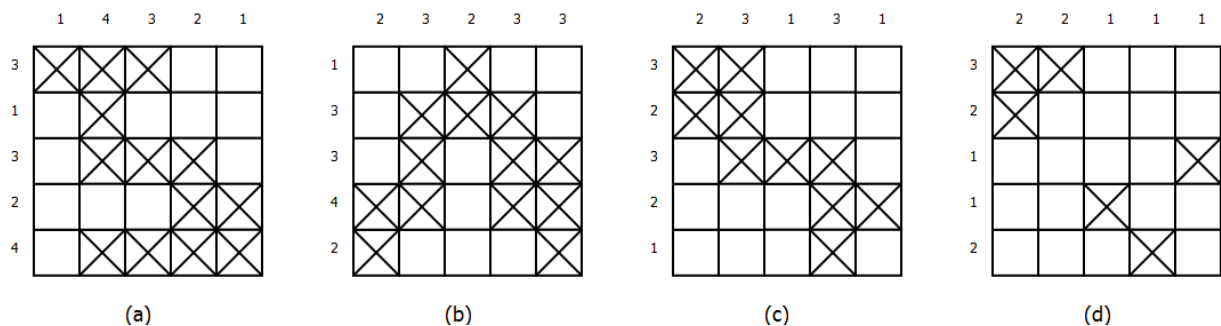
Sok esetben a kapott 2 vetület túlságosan kevés információt tartalmaz, így ugyanarra a rekonstrukciós feladatra több különböző megoldást kaphatunk. Különböző megszorításokkal azonban csökkenthetjük a lehetséges megoldások számát. Ilyen megszorítás lehet, ha feltételezzük, hogy a keresett alakzat konvex vagy információnk van a komponensek számáról. A következőkben, olyan algoritmusokat fogunk megvizsgálni, melyek ezen megszorításokat használják.

1.1. Definíció: Egy F diszkrét halmaz két pontja, $P = (p_1, p_2)$ és $Q = (q_1, q_2)$, 4-szomszéd, ha $|p_1 - q_1| + |p_2 - q_2| = 1$. A $P_0, \dots, P_k$ különböző pontokból álló sorozat 4-út, ha a P_0 pontból P_k pontba egy F diszkrét halmazba, ha a sorozat minden pontja F -ben van és P_i és P_{i+1} 4-szomszédok. Egy F diszkrét halmaz 4-összefüggő, ha F bármely két pontja között van F -beli 4-út.

A 4-összefüggő halmazokat *poliominóknak* is nevezzük. Ha egy diszkrét halmaz nem 4-összefüggő. Ilyenkor több poliominóból áll. Egy diszkrét halmaz maximális 4-összefüggő részhalmazai a halmaz *komponenseit* adják.

1.2. Definíció: Egy F diszkrét halmaz *horizontálisan konvex* (vagy *h-konvex*) ha minden sora 4-összefüggő. Egy F diszkrét halmaz *vertikálisan konvex* (vagy *v-konvex*), ha minden oszlopa 4-összefüggő. Ha egy diszkrét halmaz horizontálisan és vertikálisan is konvex, akkor *hv-konvexnek* nevezzük.

A bevezetett fogalmakat az 3. ábra szemlélteti.



3. ábra. (a) Egy h-konvex poliominó. (b) Egy v-konvex poliominó. (c) Egy hv-konvex poliominó. (d) Egy általános hv-konvex diszkrét halmaz, mely 4 komponenst tartalmaz.

2. ALGORITMUSOK HV-KONVEX MÁTRIXOK REKONSTRUÁLÁSÁRA

A csak horizontális vagy csak vertikális konvex képek rekonstruálása esetén is NP-teljes, még abban az esetben is, ha feltételezzük, hogy a kép 4-összefüggő [2, 3]. Ha azonban a diszkrét halmaz egyszerre *hv*-konvex és 4-összefüggő (8-) akkor a feladat polinomiális időben megoldható. Az általános *hv*-konvex bináris mátrixok esetén a rekonstrukció ismét NP-teljes [2]. A dolgozat célja ez utóbbi probléma mélyebb vizsgálata.

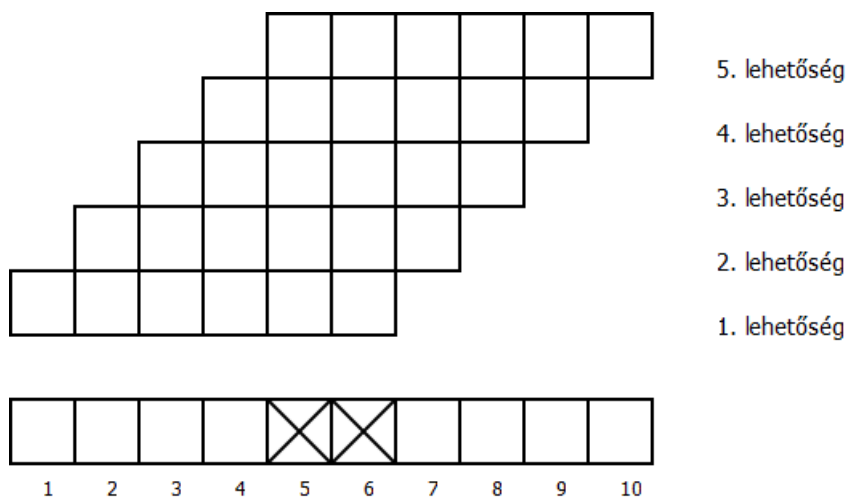
A továbbiakban három különböző elven alapuló rekonstruáló algoritmust ismeretünk a feladat megoldására.

2.1. Mag-burok algoritmus

A [4] cikkben ismertetett mag-burok algoritmus volt az első egy olyan heurisztikus eljárás, melyet átlagos *hv*-konvex képek rekonstruálására dolgoztak ki. Az eljárás során adott két halmaz, a mag és a burok. A *mag* azon pontok halmaza, amelyek biztosan hozzátartoznak a megoldáshoz. Ezen pontok összessége szükségszerűen egy *hv*-konvex alakzatot ad. A *burok* azon pontokból áll, amelyek esetlegesen hozzátartozhatnak a megoldást reprezentáló diszkrét halmazhoz. A következőkben a mag által tartalmazott pontokat *magpontoknak*, míg a burok által tartalmazott pontokat *burokpontoknak* is nevezzük. Az algoritmus az üres magból és az összes képpontot tartalmazó burokból indul ki. Minden iterációban megpróbáljuk növelni a magot és csökkenteni a burkot a sor- és oszlopösszegek segítségével. Ha a két halmaz elemei megegyeznek, akkor egy megoldást találtunk. Ha a mag nem változik, de még nem találtunk megoldást, akkor egy veremmemóriában eltároljuk az aktuális helyzetet és egy pontot választunk, amit a burok tartalmaz, de a mag nem. Ezt a maghoz hozzáadva folytatjuk az eljárást. Ha olyan eset fordul elő, hogy a burok tartalmaz olyan pontot, ami a magban nem található és üres a verem, akkor nem találtunk megoldást. A teljes eljárás ismertetése előtt bemutatjuk az eljárásban szereplő operátorokat.

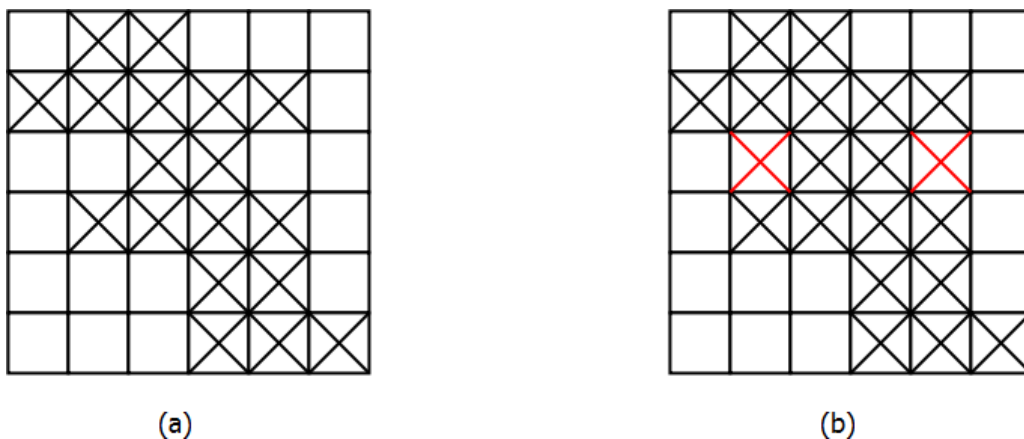
$c(B, R)$, $c(B, S)$ operátor: Paraméterei a burok pontok diszkrét halmaza B , és a függőleges S vagy vízszintes R vetületi vektor. Az operátor a paraméterben kapott vetülettől függően minden sorban vagy oszlopban a biztosan a maghoz tartozó pontokat jelöli ki. Mivel *hv*-konvex képekkel dolgozunk, minden sorban és oszlopban egybefüggő pontsorozatoknak kell lenniük. Megállapítjuk a vetület alapján, hogy milyen hosszú ez a pontsorozat és hányféleképpen lehet azt elhelyezni az adott sorban vagy oszlopban. A mag operátor azokat a

pontokat fogja kiválasztani melyek az összes lehetséges elhelyezésnél részt vesznek a megoldásban. Például ha egy 10 egység hosszú sorban szeretnénk meghatározni a magpontokat ahol a vízszintes vetület 6 egység, ami egy 6 hosszú összefüggő csík lerakását jelenti, akkor az összesen 5 különböző módon helyezhető el (4. ábra) és ahol átfedés van az összes lehetőség között (ábrán $\times$ -el jelölve) azok a pontok biztosan szerepelni fognak a megoldásunkban.



4. ábra Egy 10 egység hosszú sorban vagy oszlopban a mag meghatározása. $\times$ jelöli a magpontokat.

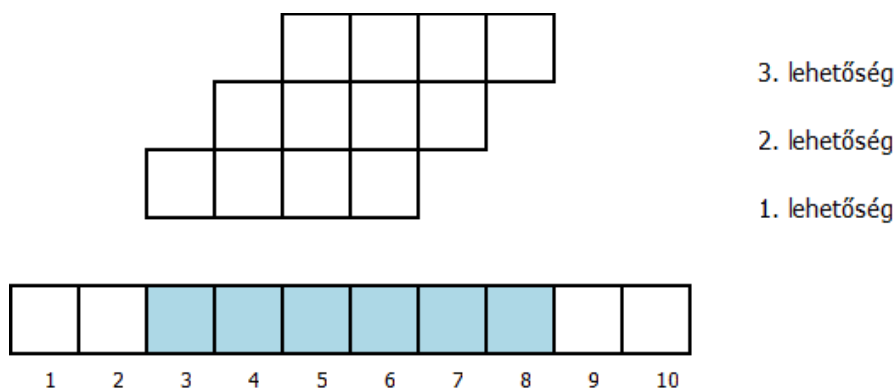
$J(A)$ operátor: A mag operátor eredménye a sorösszegeken h -konvex és az oszlopösszegeken v -konvex, viszont ezen két halmaz uniója nem biztos, hogy hv -konvex. Ezért szükség van a J operátorra, mely A input esetén visszatér a legkisebb mindkét irányban konvex A -t tartalmazó halmazzal (5. ábra).



5. ábra. Egy tetszőleges bináris mátrix (a) J operátor előtt és (b) J operátor után (piros $\times$ -ek jelölik az új elemeket)

$b(C, R)$, $b(C, S)$ operátor: Paraméterei a mag pontok diszkrét halmaza C és a függőleges S vagy vízszintes R vetületi vektor. Az operátor megvizsgálja az adott sorban vagy oszlopban a vetület és magpontok számának különbségét. Ha ez nulla, akkor a mag megfelel az adott sor/oszlop vetületének, tehát a burok ugyanakkora, mint a mag. Ha viszont nagyobb, mint nulla, akkor burok nagysága az aktuális mag két szélétől kiszámítható a vetület és magpontok számának különbségéből.

Például ha egy 10 egység hosszú sorban szeretnénk megállapítani a burokpontokat úgy, hogy az 5. és 6. pont a maghoz tartozik már, és a vetület érték 4, akkor 3-féleképpen helyezhetünk el egy 4 egység hosszú csíkot, és a burok azokat a pontokat fogja tartalmazni, amelyek legalább az egyik lehetséges elhelyezésben szerepeltek (6. ábra).



6. ábra. Egy 10 egység hosszú sorban a burok meghatározása. Kék mezők jelölik a burok pontjait.

A program pszeudokódja az alábbiak szerint adható meg:

1. legyen S egy üres verem, $k = 0$; $C_0 = \emptyset$; $B_0 = T$
2. $C := J(c(B_k, R) \cup c(B_k, S) \cup C_k)$
3. $B := B_k \cap b(C_k, R) \cap b(C_k, S)$
4. ha $(C_k = B_k)$ akkor {van megoldás, ürítsük a vermet, írjuk ki a megoldást, VÉGE}
5. ha $(C = C_k)$ akkor $\{(C, B, ((i, j) \in B \setminus C) \rightarrow S); C := C \cup \{(i, j)\}, C_k + 1 := C; B_k + 1 := B; k := k + 1\}$
6. ha $(C_k \subset B_k)$ akkor {2. lépés}
 - 6.1 különben ha $(C_k \not\subset B_k$ és S üres) akkor {nincs megoldás, VÉGE}
 - 6.2 különben $\{S \rightarrow (C, B, (i, j)), C_k + 1 := C, B_k + 1 \setminus \{(i, j)\}; k := k + 1$ és lépünk 2.-re}

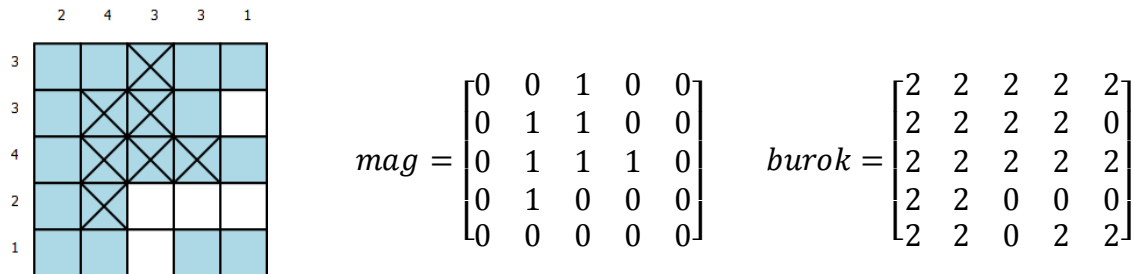
A pszeudokód alapján megadott algoritmus az első megoldás esetén leáll. Ha azt szeretnénk, hogy a program keresse meg az összes megoldást, csak annyit kell módosítanunk a kódon, hogy a 6. lépésben nem ürítjük ki a vermet és nem állítjuk le a programot, hanem elmentjük a kapott megoldást, majd kivesszük a következő elemet a veremből és folytatjuk a program futását, amíg nem lesz üres a verem.

2.1.1. Megvalósítás tömbös adatszerkezettel

A magburok algoritmus két különböző megvalósítását készítettük el. Mindkettő a leírt pszeudokód szerint működik, különbség a magot és burkot reprezentáló adatszerkezetekben és az azokat alkalmazó függvényben van.

Az első implementációban a magot és a burkot egy-egy $m \times n$ dimenziós tömb reprezentálja. A magot reprezentáló tömbben a mag egyesekkel van jelölve, míg a burok operandus esetén a burokpontok kettesekkel. Nullával azok a pontok vannak jelölve, melyeket biztosan nem tartalmaz a rekonstruált kép. Ezeket az adatokat tartalmazhatja egy tömb is, viszont az egyszerűség miatt két tömböt használunk, ahol az egyik csak a mag, míg a másik csak a burok állapotát reprezentálja. Ennek az az előnye, hogy a mag- és burok operátorok esetén kevesebb esetet kell megvizsgálunk és leegyszerűsödnek a halmazműveletek is (7. ábra).

A fentiekben leírtak egy egyszerű, de nagy tárhelyet foglaló megvalósítást eredményeznek. Az algoritmus egyszerű utasításokból létrehozható, a legkomplikáltabb utasítások a halmazműveletek.



7. ábra. Egy példa a tömbös adatszerkezetre, ahol ×-el jelöljük a magpontokat, a kék mezők pedig a burokpontokat. Mellette ugyanaz a mátrix tömbös adatszerkezettel reprezentálva.

2.1.2. Megvalósítás First-Last adatszerkezettel

A második megvalósítás alapja két-két darab $(2, m)$ és $(2, n)$ dimenziójú tömb, melyek a mag és burok sorainak és oszlopainak első és utolsó elemeit tartalmazzák. A reprezentáció a kép *hv*-konvexitását használja ki, mivel az első és utolsó elem elegendő információt nyújt a rekonstruálásban. Ez egy bonyolultabb adatszerkezet, melyet nehezebb megvalósítani, de kevesebb memóriát foglal. A halmazműveletek (unio, metszet) egyszerűen megvalósítható, viszont az összes többi művelet bonyolult, a tömbös megvalósításhoz képest sokkal összetettebb és nagyobb számítás igényű.

A 7. ábra példája esetén a magot a következőképpen fejezhetjük ki (-1 azokban a sorokban/oszlopokban szerepel, ahol nincs mag):

SorElső: 3, 2, 2, 2, -1

SorUtolsó: 3, 3, 4, 2, -1

OszlopElső: -1, 2, 1, 4, -1

OszlopUtolsó: -1, 4, 3, 3, -1

A burok adatai:

SorElső: 1, 1, 1, 1, 1

SorUtolsó: 5, 4, 5, 2, 5

OszlopElső: 1, 1, 1, 1, 1

OszlopUtolsó: 5, 5, 3, 5, 5

A mag *hv*-konvex, a burok azonban nem feltétlenül az. Vagyis, a magot csak sorokkal vagy oszlopokkal is reprezentálni lehet, a burkot viszont nem, mivel például az utolsó oszlopban az adatok alapján az első pozíciótól az ötödikig van a burok, de megszakításokkal. Ezért szükségünk van a sorok és az oszlopok metszetére, aminek segítségével kapjuk meg a burokpontokat.

2.2. Szimulált hűtésen alapuló rekonstrukciós algoritmus

Most egy olyan eljárást mutatunk be, mely a két vetülettel való $h\nu$ -konvex alakzatok vagy közel $h\nu$ -konvex alakzatok rekonstruálását végzi el a Dhal és Flatberg által publikált egészértékű programozás és szimulált hűtés segítségével [5,6]. Ezek mellett az algoritmus kihasználja az 1.1. tételben kimondott összefüggést.

A szimulált hűtésen alapuló algoritmus használata során szükségünk van egy eljárásra, mellyel számszerűsíteni tudjuk, hogy egy bináris mátrix mennyire $h\nu$ -konvex.

Legyen A egy bináris mátrix és a_{ij} a mátrix elemének értéke az (i, j) pozícióban. Az alábbi képlet összeszámolja a szomszédos egyeseket:

$$f(A) = \sum_{i=1}^{m-1} \sum_{j=1}^n a_{i,j} a_{i+1,j} + \sum_{i=1}^m \sum_{j=1}^{n-1} a_{i,j} a_{i,j+1}$$

Az $f(A)$ segítségével egy egész számot kapunk egy A bináris mátrix konvexitásának a mértékéről. Továbbiakban a célunk az, hogy maximalizáljuk az f függvény értékét az alábbi megszorítások mellett:

$$\sum_{j=1}^n a_{ij} = h_i$$
$$\sum_{i=1}^m a_{ij} = v_j$$

ahol $a_{ij} \in \{0, 1\}$ ($i = 1, \dots, m$ és $j = 1, \dots, n$).

2.2. Ryser algoritmus

A szimulált hűtéses rekonstrukciós algoritmusnak szüksége van egy viszonylag jó és könnyen megkapható kezdeti állapotra is. A kapott vetületek alapján Ryser algoritmus segítségével fogjuk megadni a kezdeti állapotot.

A Ryser algoritmus egy egyszerű rekonstrukciós eljárás, mely függőleges és vízszintes vetületekből ad egy a vetületeket kielégítő bináris mátrixot. Ez esetben a rekonstruált képre

nincs semmilyen előírás (pl.: konvexitás). Egyedül az alábbi feltételeknek kell, hogy megfeleljenek az $R \in \mathbb{N}^m$ és $S \in \mathbb{N}^n$ vektorok:

$$\sum_{i=1}^m r_i = \sum_{j=1}^n s_j,$$

$$r_i \leq n \quad (i = 1, \dots, m),$$

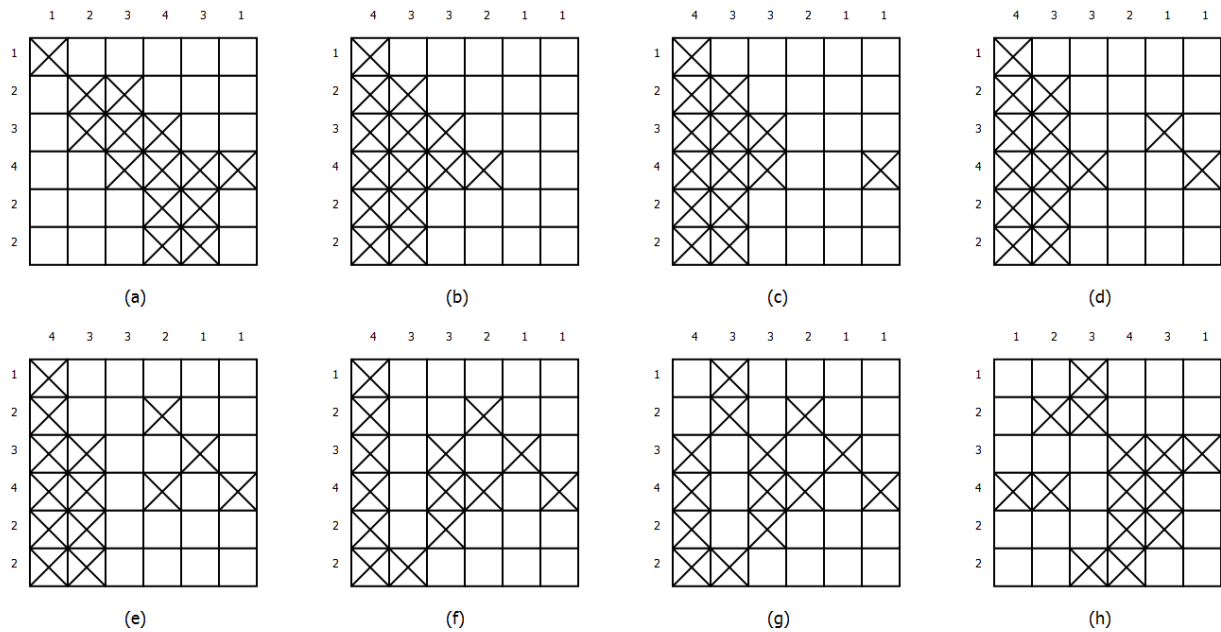
$$s_j \leq m \quad (j = 1, \dots, n).$$

A továbbiakban feltételezzük, hogy ezek az egyszerű feltételek teljesülnek. Az algoritmus a következő lépésekből áll:

1. Rendezzük a bináris mátrix S vektorát nem növekvő sorrendbe. Jelölje S' a kapott vektort. A sorrendbe állításhoz szükséges permutációt nevezzük π -nek.
2. Az R sorösszegek alapján balról-jobbra töltjük fel az A mátrixot.
3. Minden $k = n, \dots, 2$ -re hajtsuk végre a következőt: ha $s_k^* < s'_k$ akkor az A^* első $(n - 1)$ oszlopából álló mátrix legjobboldalibb 1-eseket tartalmazó l -edik oszlopából toljunk $s_k^* - s'_k$ darab 1-est a k -adik oszlopba azokban a sorokban, ahol az l -edik oszlopban 1-es, a k -adikban pedig 0 áll. Amennyiben választhatunk, akkor a legfelsőbb sorokban lévő 1-eseket toljuk át. Amennyiben nincs $s_k^* - s'_k$ darab 1-es ilyen pozícióban az l -edik oszlopban, akkor a hiányzó elemeket az $l - 1, l - 2$, stb. oszlopokból toljuk át.
4. Az A^* mátrix oszlopait rendezzük eredeti sorrendbe π^{-1} permutáció által.

A Ryser algoritmus által kapott megoldás általában nem ad egyértelmű eredményt a kapcsoló komponensek nagyszámú előfordulása miatt.

Az algoritmus lépéseit az alábbi példán szemléltetjük. Legyen $R = (1, 2, 3, 4, 2, 2)$ és $S = (1, 3, 3, 4, 3, 1)$. Az eredeti kép a 8.a ábrán látható. $S' = (4, 3, 3, 2, 1, 1)$, amit a $\pi = [4 \ 3 \ 5 \ 2 \ 1 \ 6]$ permutáció alkalmazásával kapunk. Permutáció után kapott mátrixunk a 8.b ábra, aminek oszlopösszege $S^* = (5, 4, 2, 1, 0, 0)$. Következő lépésekben a jobbról balra haladva áthúzzunk megfelelő számú elemet a balra lévő oszlopokból melyek így kielégítik az S^* vektort (8.c ábra – 8.g ábra). Végül utolsó lépésként az A^* mátrix oszlopait π^{-1} permutáció segítségével az eredeti sorrendbe rendezzük (8.h ábra).



8. ábra. A Ryser algoritmus lépései konkrét példán keresztül

2.2. Szimulált hűtés

Az optimalizáló algoritmusok működésének alapja, hogy akkor lépnek egy szomszédos állapotba, ha a következő állapot az általunk vett szempontokból jobb, mint az aktuális állapot. Sajnos gyakori eset, hogy a szomszédos állapotok nem jobbak, mint az aktuális állapotunk, viszont ez utóbbi csak lokálisan a legjobb állapot, globálisan nem az. Ebben az esetben beragadunk az algoritmus a lokális optimumba. Ezt a problémát oldja meg a *szimulált hűtés* algoritmus. A kiindulási állapotból indulunk az optimum felé. Ha az azon vezető úton az aktuálisból jobb állapotba kerülünk tovább lépünk, ha rosszabb állapotba jutnánk, akkor véletlenszerűen eldöntjük, hogy megteesszük-e azt a lépést vagy sem. Egy ilyen lépés valószínűsége folyamatosan csökken, amit a *hűtési terv* határoz meg. Ennek a tulajdonságnak köszönhetően az algoritmus könnyen ki tud lépni a lokális optimumokból, így nagyobb eséllyel találja meg a lehető legjobb állapotot.

Mi ebben az esetben a szimulált hűtéssel a bináris mátrix konvexitásának a maximumát keressük, tehát maximalizáljuk a függvényt. Első lépésben választunk egy kezdeti megoldást, ez a sor és oszlop vetületek alapján a Ryser algoritmus végeredménye. Továbbá megadunk egy T_0 kezdeti hőmérsékletet. Az algoritmus rendelkezik egy *hűtési tervvel*. A hűtési tervet a következő paraméterek határozzák meg:

- T_0 kezdeti hőmérséklet
- a hőmérséklet csökkenésének a menete

- a megállási feltételben szereplő legalacsonyabb hőmérséklet

A hűtési terv mellett megadunk egy számlálót is, ennek neve *Nbr*. Ha nem fogadunk el egy szomszédot aktuális állapotnak, akkor megnöveljük az értékét. Ellenkező esetben lenullázzuk. A számláló arra szolgál, hogy megállási feltételt adjon abban az esetben, ha bizonyos mennyiségű állapotátmenet után sem találunk új aktuális állapotot.

A kezdeti megoldás megadása után az aktuális állapotban választunk egy tetszőleges szomszédot. Ezt úgy tesszük meg, hogy egy véletlenül kiválasztott kapcsoló komponenszt megcserélünk. Ezek után megvizsgáljuk, hogy a szomszédos állapotot elfogadhatjuk-e aktuális állapotnak.

- Ha a szomszédos állapot jobb, akkor elfogadjuk és ő lesz az új aktuális állapot, csökkentjük a hőmérsékletet és nullázzuk a számlálót.
- Ha nem jobb, mint az aktuális állapot akkor választunk egy véletlen számot 0 és 1 között és összehasonlítjuk az $\exp\left(\frac{-|f(aktualis) - f(szomszedos)|}{Temp}\right)$ kifejezés értékével, ahol *Temp* az aktuális hőmérsékletet jelöli.
 - Ha a véletlenszám a kifejezés értékénél kisebb, akkor a szomszédos állapotot elfogadjuk, új aktuális állapot lesz, csökkentjük a hőmérsékletet és nullázzuk a számlálót.
 - Különben növeljük a számlálót.

Ezeket a lépéseket addig ismételjük, meg a megállási feltételek közül valamelyik nem teljesül, vagyis elérünk egy bizonyos hőmérsékletet vagy egy adott számú aktuális állapot változás nélküli esett. Minél nagyobb a kezdeti hőmérséklet annál nagyobb valószínűséggel fogad el az algoritmus rosszabb állapotokat is. A hőmérséklet csökkenésének a menete befolyásolja az átmenetek számát. Kismértékű csökkenés esetén több állapotátmenetre kerül sor, aminek hatására az eljárás nagyobb eséllyel találja meg a globális optimumot, viszont ezzel növekszik a futásidő nagysága. Bizonyított, hogy végtelen sokáig tartó hőmérséklet csökkenéssel az eljárás 1 valószínűséggel megtalálja a globális optimális állapotot. Az algoritmus időigénye az állapotok átmenetek számának a növekedésével lineárisan nő. Tárigénye viszont kicsi, mivel csak az aktuális állapotot tárolja. Esetünkben a kezdeti hőmérsékletet 5-nek vettük, megállási feltétel pedig az volt, ha a hőmérséklet elérte a 0.0005 értéket vagy 1000 iteráció után sem találtunk jobb eredményt.

Az algoritmus pszeudokódja az alábbiak alapján adható meg.

1. *Kezdetimegoldás*(Ryser algoritmus)
2. *Temp* = 5, *Nbr* = 0

3. amíg ($Temp > 0.0005$ és $Nbr < 1000$):

3.1. kapcsoló komponens kiválasztás

3.2. ha $((f(elozo) - f(szomszedos) < 0)$ vagy $(p < \exp(-|d| / Temp))$) akkor

$elozo = szomszedos$; $Temp$ csökken; $Nbr = 0$;

különben

$Nbr++$;

ahol $f(X)$ az X bináris mátrix konvexitásának a mértékét fejezi ki, $d = (f(elozo) - f(szomszedos))$. p pedig véletlen szám ($0 < p < 1$). A hőmérsékletet exponenciálisan, $Temp = Temp \cdot 0.9995$ művelet alapján csökkentettük.

2.3. Komponensek elhelyezkedésén alapuló algoritmus

Végül egy olyan algoritmust ismertetünk, mely a hv -konvex mátrixokat a komponensek befoglaló téglalapjainak meghatározásával igyekszik rekonstruálni a kapott képet [7,8,9].

Legyen F egy bináris kép k darab komponenssel ($k \geq 1$), ahol az l -edik komponens az $I_l \times J_l = [i_l, i'_l] \times [j_l, j'_l]$ legkisebb diszkrét téglalapban található meg.

Az F komponensei *diszjunktak*, ha bármely $1 \leq l, l' \leq k$ és $l \neq l'$ esetén $I_l \cap I_{l'} = \emptyset$ és $J_l \cap J_{l'} = \emptyset$.

2.1. Definíció: Legyen $S = (s_1, s_2, \dots, s_m)$, ahol $1 \leq i_1 \leq i_2 \leq m$. A R $[i_1, i_2]$ intervallummal meghatározott *parciális összege* az alábbi módon adjuk meg

$$S_{[i_1, i_2]} = \sum_{i=i_1}^{i_2} s_i$$

A definíció a $R = (r_1, r_2, \dots, r_n)$ vektor esetén hasonlóan adható meg.

2.2. Definíció: Legyen $\mathcal{S}$ poliominók osztálya, $S \in \mathbb{N}^m$ és $R \in \mathbb{N}^n$. Az $[i_1, i_2] \times [j_1, j_2]$ téglalap *kompatibilis* az $\mathcal{S}$ osztályra nézve, ha létezik olyan $F \in \mathcal{S}$ poliominó, melyre $\mathcal{H}(S) = (s_{i_1}, \dots, s_{i_2})$ és $\mathcal{V}(R) = (r_{j_1}, \dots, r_{j_2})$.

Ha $[i_1, i_2]$ és $[j_1, j_2]$ kompatibilisek egy tetszőleges $\mathcal{S}$ osztályra akkor a következő feltételek igazak:

- $S_{[i_1, i_2]} = R_{[j_1, j_2]}$
- $s_i \leq j_2 - j_1 + 1$ minden $i_1 \leq i \leq i_2$ -re
- $r_j \leq i_2 - i_1 + 1$ minden $j_1 \leq j \leq j_2$ -re

2.1. Lemma. Tetszőleges $S \in \mathbb{N}^m$ és $R \in \mathbb{N}^n$ vektorok esetén a kompatibilis téglalapok száma legfeljebb $mn \cdot \min\{m, n\}$.

A fenti lemma következménye, hogy tetszőleges $S \in \mathbb{N}^m$ és $R \in \mathbb{N}^n$ vektorok esetén a kompatibilis téglalapok száma a bal felső sarokban az i -edik (j -edik oszlopban) $m \cdot \min\{m, n\}$ ($n \cdot \min\{m, n\}$) által határolt.

Minden $i_1 \leq i \leq i_2$ és $j_1 \leq j \leq j_2$ -re defináljuk A_{i_1, i_2, j_1, j_2} boolean változót, ahol $A^{\mathcal{P}}_{i_1, i_2, j_1, j_2} = igaz$, akkor is csakis akkor, ha $[i_1, i_2] \times [j_1, j_2]$ által határolt téglalapban konzisztens és a $\mathcal{P}$ osztály tagja, különben *hamis*.

Ezek mellett jelöljük $\mathcal{HV}_4$ -el a hv -konvex poliomínók osztályát.

Az algoritmus pszeudokódja az alábbiak alapján adható meg.

Bemenet: $H \in \mathbb{N}^m$ és $V \in \mathbb{N}^n$ vektorok

Kimenet: Egy k komponens számú bináris kép, melyre $\mathcal{H}(F) = (s_1, \dots, s_i)$ és $\mathcal{V}(F) = (r_1, \dots, r_j)$ teljesül vagy „nincs megoldás”

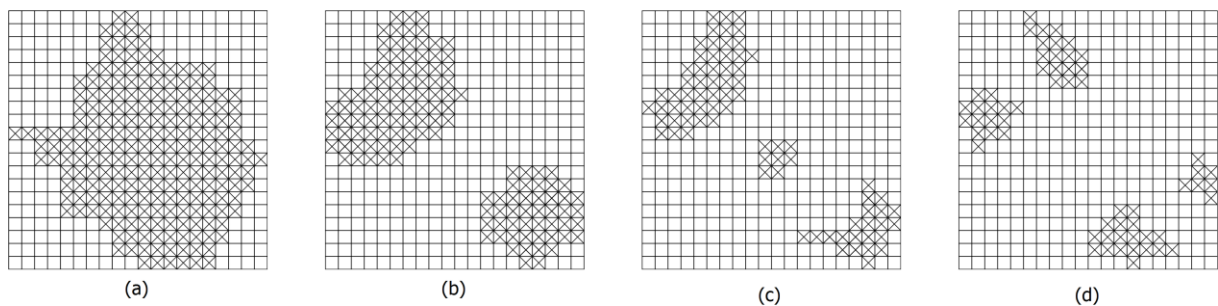
0. minden $1 \leq i_1 \leq i_2 \leq m$ és $1 \leq j_1 \leq j_2 \leq n$ $A^{\mathcal{HV}_4}_{i_1, i_2, j_1, j_2} = hamis$
1. minden $1 \leq i_1 \leq m$ és $1 \leq j_1 \leq n$ kiszámolja és eltárolja az összes (i_2, j_2) -t, mely $[i_1, i_2] \times [j_1, j_2]$ téglalpok kompatibilisek.
2. minden konzisztens téglalapra kiszámolni $A^{\mathcal{HV}_4}_{i_1, i_2, j_1, j_2}$ -t
3. Megkeresni azokat az $(i_1, i'_1, j_1, j'_1), \dots, (i_k, i'_k, j_k, j'_k)$ négyeseket, melyekre $A^{\mathcal{HV}_4}_{i_l, i'_l, j_l, j'_l} = igaz$ minden $l = 1, \dots, k$ -ra, és $i_l = 1$, $i'_l = i'_{l-1} + 1$ ($l = 2, \dots, k$), és $i'_k = m$, és a téglalapok $[i_1, i'_1] \times [j_1, j'_1], \dots, [i_k, i'_k] \times [j_k, j'_k]$ oszlopai páronként diszjunktak;
4. ha a 3. lépés sikeres akkor a megoldást kimenetként megkapjuk a komponensek minimális befoglaló $[i_1, i'_1] \times [j_1, j'_1], \dots, [i_k, i'_k] \times [j_k, j'_k]$ téglalapjait;

különben „nincs megoldás”;

Az algoritmus futásideje $O(m^2 n^2 \cdot \min\{m^3, n^3\} + \min\{m, n\}^{3k})$, ahol a k változó a bináris képen található komponenseknek a száma.

3. A KISÉRLETI EREDMÉNYEK

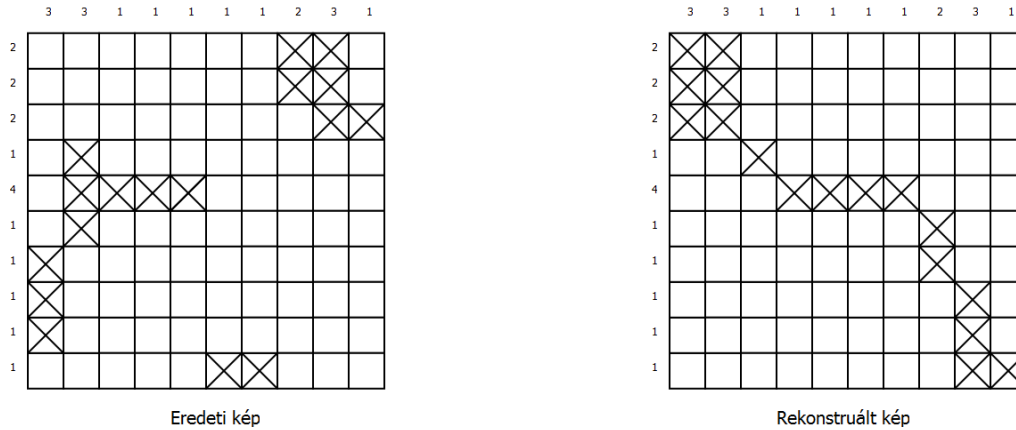
Mivel tudjuk, hogy a $h\nu$ -konvex mátrixok rekonstruálása NP-teljes, így a dolgozat célja annak vizsgálata, hogy milyen tényezők befolyásolják a különböző ismertett rekonstrukciós eljárások teljesítményét. Viszonyításképpen több szempontot veszünk figyelembe. Szem előtt tartjuk a rekonstrukciók pontosságát, valamint a futásidők nagyságát. Ezen adatok alapján megpróbáljuk megállapítani, hogy pontosan mi határozza meg az adott rekonstrukciós probléma nehézségét. A tesztelés egy Compaq 615 számítógépen (processzor: AMD Athlon X2, 2.1 GHz, L2 cache: 1 MB; memória: 2 GB, DDR2) történt. Az algoritmusok Javában Eclipse fejlesztői környezetben készültek, kivéve a komponensek elhelyezkedésén alapuló algoritmus, mely egy már rendelkezésre álló C++ implementációját használtuk. Az összes teszt lefuttatása shell script segítségével történt. A teszt során összesen 20000 képen vizsgáltuk a programokat. A képek 10×10 , 20×20 , 30×30 , 40×40 és 50×50 -es méretűek, ahol minden méretben megtalálható 100 darab különböző egy-, kettő-, három- és négy komponenses kép. Ezek mellett a komponensek elhelyezkedésén alapuló algoritmus és a mag-burok algoritmus összehasonlítása alatt ugyanolyan kettő-, három- és négy komponenses 60×60 , 80×80 , 100×100 -as méretű képeket is felhasználtunk (9. ábra).



9. ábra. Példa tesztképek. (a) egy komponenst tartalmazó. (b) két komponenst tartalmazó. (c) három komponenst tartalmazó. (d) négy komponenst tartalmazó.

3.1. Rekonstrukció pontossága

Mivel a két vetületből való rekonstrukció erősen alulkorlátozott probléma, így két hv -konvex kép között jelentős különbség lehet, miközben mindkettő ugyanazokat a vetületeket elégíti ki (10. ábra). Fontos tehát először is azt megvizsgálnunk, hogy az ismert algoritmusok mennyire adnak pontos megoldást.



10. ábra. Két különböző kép, melyek ugyanazokat a vetületeket elégítik ki

Legyen $M^o = (m_{i,j}^o)$ az eredeti kép bináris mátrixa és $M^r = (m_{i,j}^r)$ a rekonstruált kép bináris mátrixa. A rekonstrukció pontosságát (relative mean error - RME) az alábbi képlet segítségével állapítjuk meg:

$$RME = \frac{\sum_{i,j} |m_{i,j}^o - m_{i,j}^r|}{\sum_{i,j} |m_{i,j}^o|}$$

Az 1. táblázat összesíti az algoritmusok rekonstrukcióinak átlagos pontosságát és annak szórásnégyzetét a képméret és a komponens számainak függvényében. A táblázat alapján látható, hogy a szimulált hűtéses algoritmus teljesít a legrosszabbul, viszont a két mag-burok algoritmus csak néhány esetben haladja meg a 0.1 feletti hibaátlagot. A szimulált hűtésnél használt hűtéstervünk alapján az eljárás az esetek nagy részében nem adott eredményül teljesen hv -konvex mátrixokat, ez okozhatja a pontatlanabb eredményeket. Ellenben a mag-burok algoritmusok esetén, ha a rekonstruált kép teljesen el is tér az eredeti képtől, akkor is hv -konvex. Ha a kép csak egy komponenset tartalmaz, akkor nem nagy az eltérés, de minél több komponensből áll, annál kevésbé valószínű, hogy a rekonstruált kép azonos az eredeti

képpel. Ennek okát a kapcsoló komponensek számában kell keresnünk. Az egy komponens tartalmazó kép esetén jelentősen kevesebb kapcsoló komponens van, mint egy nagyobb komponensszámú képen, ezért statisztikailag nagyobb az esély, hogy az algoritmusok találnak egy megoldást, ami nem hasonlít az eredeti képre, de ugyanazokkal a vetületekkel rendelkezik.

mag-burok (tömb)										
	10		20		30		40		50	
	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet
1	0,01	0,09	0,002	0,01	0,0002	0,001	0,0002	0,001	0,0001	0,0008
2	0,1	0,35	0,01	0,04	0,006	0,01	0,03	0,17	0,01	0,09
3	0,26	0,54	0,04	0,11	0,01	0,04	0,06	0,29	0,12	0,34
4	0,7	0,77	0,11	0,22	0,05	0,16	0,1	0,4	0,18	0,51
mag-burok (First-Last)										
	10		20		30		40		50	
	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet
1	0,01	0,09	0,002	0,01	0,0002	0,001	0,0002	0,001	0,0001	0,0008
2	0,1	0,35	0,01	0,04	0,006	0,01	0,03	0,17	0,01	0,09
3	0,26	0,54	0,04	0,11	0,01	0,04	0,06	0,29	0,12	0,34
4	0,7	0,77	0,11	0,22	0,05	0,16	0,1	0,4	0,18	0,51
Szimulált hűtésen alapuló rekonstrukció										
	10		20		30		40		50	
	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet
1	0,41	0,33	0,57	0,36	0,68	0,36	0,71	0,46	0,69	0,39
2	0,76	0,53	1,01	0,49	1,07	0,5	1,13	0,54	1,14	0,53
3	0,88	0,71	1,1	0,61	1,21	0,57	1,31	0,61	1,35	0,59
4	1,06	0,82	1,34	0,66	1,38	0,64	1,38	0,67	1,39	0,63

1. táblázat. A képminőségek eredményeit összesítő táblázat

A kapott eredmények mellett megvizsgálhatjuk, hogy a szimulált hűtésen alapuló rekonstrukciós algoritmus a megadott hűtés tervvel mennyire volt hatékony. Ennek vizsgálata esetén figyelembe vesszük a hűtés kezdeti állapotának – a Ryser algoritmus eredményének – és a program végeredményének a konvexitás különbségét ($f(S)$). Végül a kapott értéket osztjuk az objektum-pixelex számával (*EgyesekSzáma*). Tehát:

$$javításMértéke = \frac{|f(R) - f(S)|}{EgyesekSzáma} ,$$

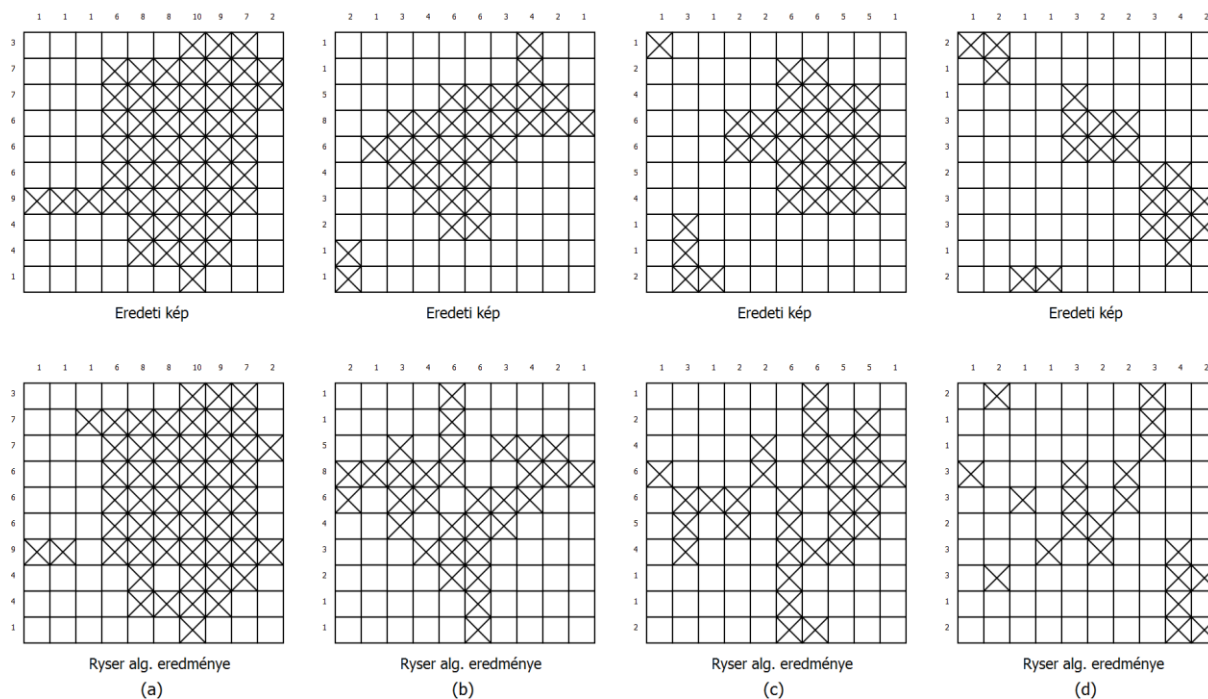
ahol az f függvény a paraméterül kapott kép konvexitásának a mértékét adja eredményül.

A javítás mértéke a kép komponenseinek számával egyenesen növekszik. Ennek oka, hogy nagyobb komponens szám esetén a Ryser algoritmus kevésbé jó kezdeti megoldást ad (11. ábra). Kevesebb komponensből álló képeken a képmérethez képest több egyes szerepel, nagyobb értékűek a sor- és oszlopvetületek, ennek eredményeként a visszaadott kép „tömörebb”, kevesebb kapcsoló komponenset tartalmaz, így kevesebbet lehet javítani a konvexitásán. Viszont, ha több komponens van a képen, akkor a vetületi értékek kisebbek, és a kapott kép kevésbé fog hasonlítani az eredeti képre.

Ryser-SA (minoseg-atlag)										
	10		20		30		40		50	
	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet
1	0,11	0,09	0,09	0,08	0,09	0,06	0,09	0,07	0,07	0,06
2	0,25	0,014	0,16	0,11	0,15	0,9	0,14	0,09	0,15	0,08
3	0,26	0,18	0,19	0,13	0,17	0,11	0,16	0,1	0,17	0,09
4	0,26	0,2	0,21	0,15	0,19	0,13	0,16	0,11	0,17	0,1

2. táblázat. A Ryser algoritmus és a szimulált hűtéses rekonstrukció végeredményének különbségeit összefoglaló táblázat

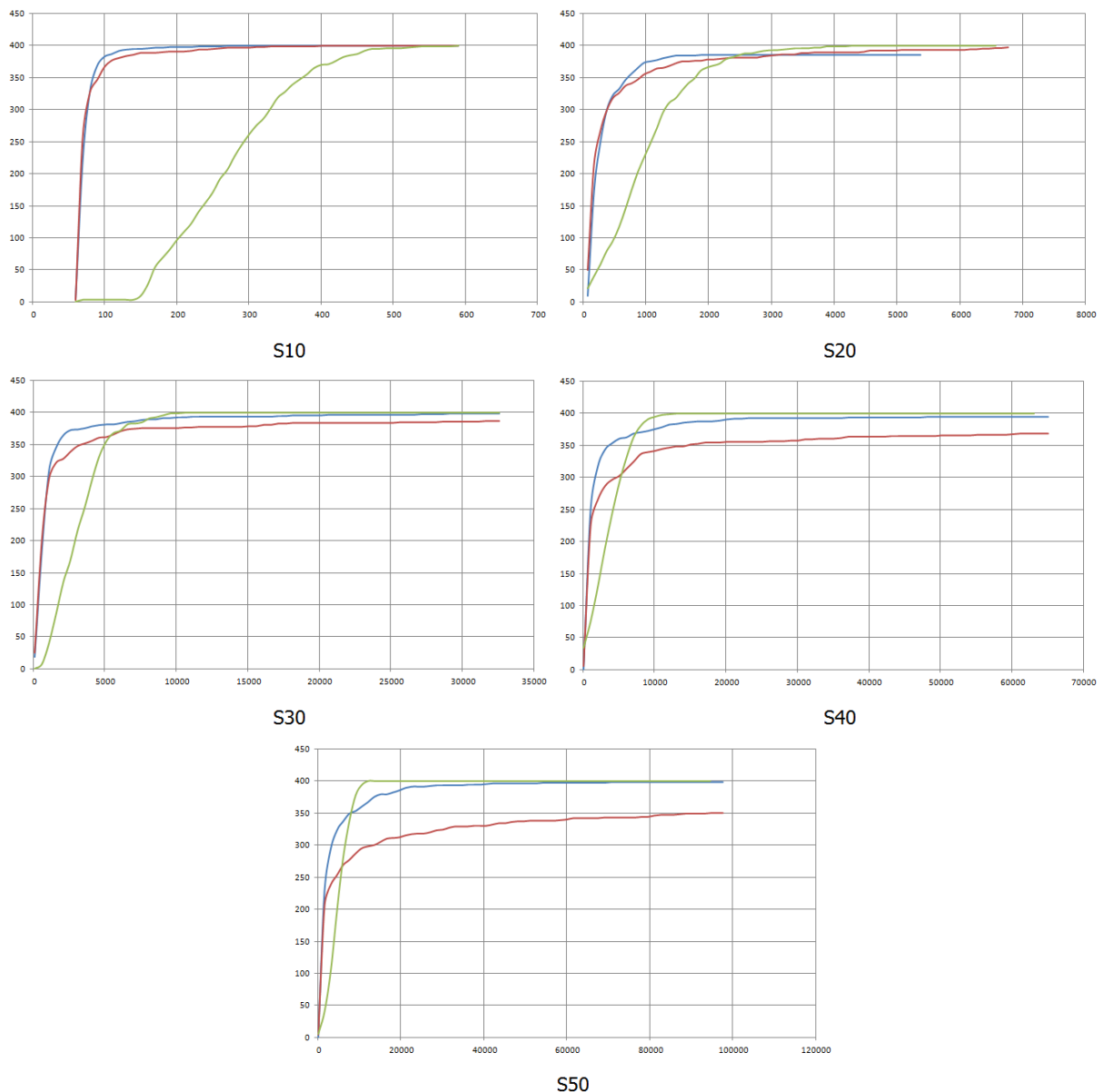
A képek méretének növekedése ugyanakkor nem befolyásolja nagymértékben a javítás értékét, vagyis a szimulált hűtésen alapuló rekonstrukciós algoritmus kevésbé érzékeny a képek méretbeli változására. A 2. táblázatban megtekinthető, ahogy méretenként az átlagos eltérés 0.01-0.09. A komponensek számát tekintve az egy komponens számú képeken javítja a legkevesebbet az algoritmus, a nagyobb komponens számú képeken növekszik a javítás mértéke a komponensek mennyiségével. Az egy és kettő komponens számú képek közötti átlagos javítás 0.07-0.14-el is növekedhet, viszont több komponens esetén már nem növekszik ilyen nagymértékben a javítás értéke.



11. ábra. Ryser algoritmus megoldásainak eltérései az eredeti képhez képest. (a) 6 pontban, (b) 15 pontban, (c) 22 pontban, (d) 24 pontban térnek el egymástól.

3.2. Futási idők

A 12. ábra az algoritmusok futásidejeinek eloszlásait mutatja be. A futásidők között néhány esetben kiugróan kicsi vagy nagy értékeket tapasztalunk. Az ilyen esetek száma relatíve csekély, így nem tekintjük őket szignifikánsoknak. Ezen okok miatt figyelmen kívül hagyjuk a legjobb és legrosszabb 5%-ot a kapott adatok között.

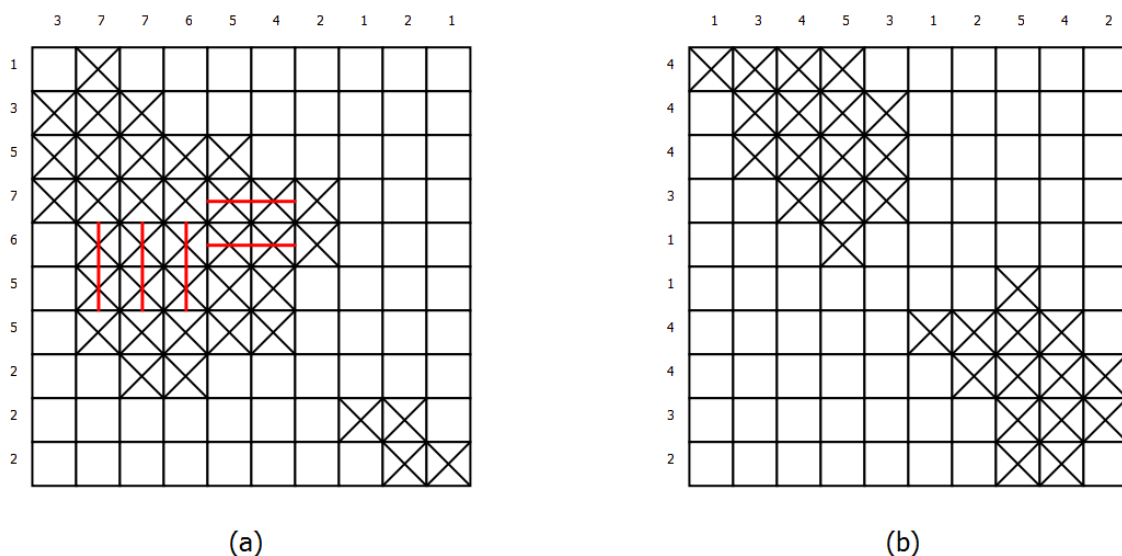


12. ábra. A 10, 20, 30, 40, 50-es méretű képek futásidő eloszlása. A vízszintes tengelyek az időt jelölik milliszekundumban. A függőleges tengelyek a lefutott tesztesetek mennyiségét.

Jelölések: piros - First-Last adattípus, kék – Tömbös adattípus, zöld – Szimulált hűtéses rekonstrukció

A 3. táblázat a nem szignifikáns adatok eltávolítása után kapott eredményeket tartalmazza, ahol látható, hogy a komponensek növekedésével az átlagos futásidő és a szórásnégyzet is növekszik a mag-burok algoritmus esetén. Ahol a mag-burok anélkül tud eredményt adni, hogy verem műveleteket kelljen végeznie (vagy nem kell sokszor végrehajtania), ott a First-Last adatszerkezetet alkalmazó algoritmus jobban teljesít, mint a tömbös. A First-Last adatstruktúrát használó algoritmus azon függvényei, melyek a pontok találgatását végzik nagyobb műveletigényűek, ezért nagyobb méretű, több komponensből álló képek esetén ez a

módszer jelentősen rosszabbul teljesít a tömbös adatszerkezetű algoritmushoz képest. A pontok próbálgatása a bal felső ponttól a jobb alsó pontig halad, ezért ha a bal felső sarokban találhatóak objektumpontok, akkor a futásidő jelentősen kisebb. A komponensek számának növekedése azért hat kedvezőtlenül a futásidőre, mert a komponensek számának a növekedésével a halmaz kevesebb képpontot tartalmaz, így kisebbek lesznek a vetületi vektorok elemei. Ha egy vetületi érték nem nagyobb, mint a kép méretének fele akkor az algoritmus nem tud kezdeti megoldást létrehozni abban a sorban vagy oszlopban. Ha az R és S vektorok minden eleme kisebb, mint a kép méretének a fele akkor az algoritmus nem tud kezdeti megoldást felállítani és kénytelen a veremből próbálgatni a lehetséges megoldásokat. A legrosszabb eset, az ha a komponensek közel azonos méretűek, ekkor egyidőben viszonylag kicsi a területük és már két komponens esetén nem lehetséges kezdeti megoldást megadni a magoperátor segítségével. (13. ábra)



13. ábra. Két különböző két komponenses kép. (a) Fel lehet állítani kezdeti megoldást. A $c(B_0, R)$ és $c(B_0, S)$ operátorok által képzett halmazok elemeit vízszintes és függőleges piros vonalak jelzik. (b) Egy olyan eset, mikor nem lehetséges kezdeti megoldás felállítani.

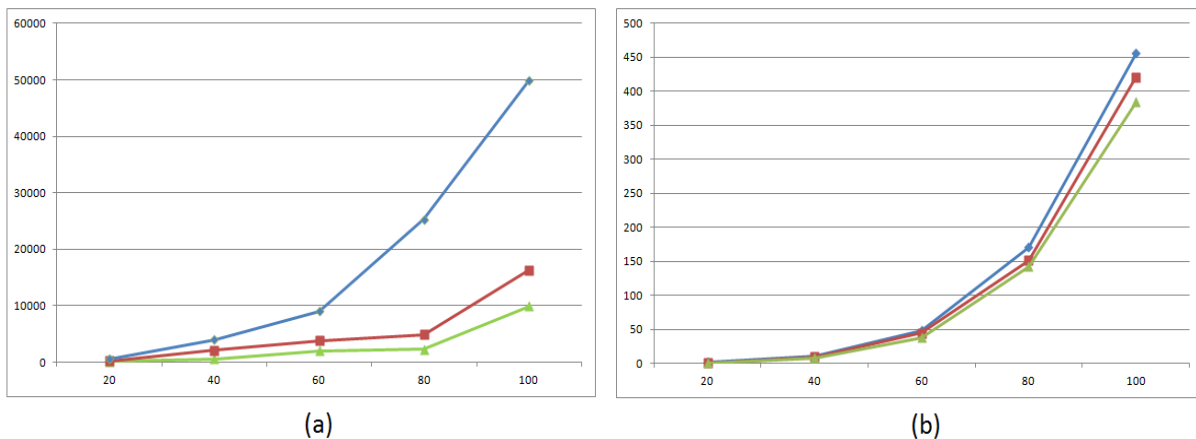
Szimulált hűtés esetén a komponensek számának növekedésével növekszik a futás idő is. Ennek oka, hogy a több komponenses kép több kapcsolókomponenst tartalmazhat, amelynek következményében, több adatot kell számon tartani. A kisebb méretű képek esetén a magburok algoritmus akár 10 lépésben is meg tudja határozni a rekonstruált kép pontjait, míg a szimulált hűtéssel rekonstruáló algoritmusnak több ezer lépés is kellhet. A magburok

algoritmusnak azonban egy iterációban sokkal több műveletet kell elvégeznie, mint a szimulált hűtéses eljárásnak.

mag-burok (tömb)										
	10		20		30		40		50	
	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet
1	63,32	3,21	109,62	45,16	189,40	86,81	297,24	128,47	414,87	156,86
2	71,86	10,24	191,16	77,27	535,19	183,76	577,82	245,98	1283,98	835,17
3	75,17	9,37	257,63	112,59	1032,99	491,07	2089,04	1847,07	3290,17	2899,45
4	72,68	4,92	581,67	244,95	2079,08	2088,48	3842,85	4006,69	6725,81	5687,12
mag-burok (First-Last)										
	10		20		30		40		50	
	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet
1	64,10	3,69	89,64	31,22	223,98	135,36	324,46	308,59	355,06	210,53
2	68,35	8,30	159,41	79,45	521,02	331,74	1129,00	1202,92	3266,66	4953,86
3	76,21	13,60	251,57	183,66	1349,13	1258,85	3947,02	6894,63	13109,22	20285,44
4	73,81	17,95	1132,48	1163,64	5500,48	10444,95	11135,62	16509,00	25450,32	30058,24
Szimulált hűtésen alapuló rekonstrukció										
	10		20		30		40		50	
	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet	átlag	szórás négyzet
1	240,69	61,88	616,96	442,9	2513,16	1138,69	4206,1	1593,77	7619,78	1003,88
2	262,46	62,83	1103,42	511,72	3224,13	1362,97	5582,69	2013,56	7558,32	1030,59
3	283,69	60,8	1193,73	434,1	3239,11	1193,69	6076,86	2024,21	7835,67	1119,04
4	290,46	63,34	1308,22	522,38	3100,45	1315,79	5734,78	1896,57	7405,66	1087,91

3. táblázat. Futásidők átlaga és szórása milliszekundumban kifejezve

A kapott eredmények mellett megvizsgáltuk a mag-burok és a komponensek elhelyezkedésén alapuló algoritmus futásidejeiket nagyobb képeken. A 14. ábra által ábrázolt eredmények alapján látható, hogy a magburok algoritmus futásideje a komponensek számának a növekedése alapján exponenciálisan növekszik. Mindezek mellett a komponensek számának és a kép méretének a növekedésével egyre nagyobb különbségek alakulnak ki a futásidők átlagai között, míg a naív algoritmusnál inkább a kép mérete határozza meg a futásidők átlagának a nagyságát.



14. ábra. Az átlagos futás idők eloszlása, ahol a függőleges tengely a milliszekundumban megadott időt, a vízszintes tengely pedig a képek méreteit ábrázolja. A zöld három jelöli a 2-, piros négyzet a 3-, kék négyzet pedig a 4 komponenses képeket. (a) mag-burok algoritmus futásidejeinek esetén. (b) komponensek elhelyezkedésén alapuló algoritmus esetén.

3.3. Összegzés és további tervek

A dolgozat célja annak megállapítása volt, hogy milyen tényezők befolyásolják a két vetület által rekonstruáló algoritmusok teljesítményeit. A kísérleteink eredményeiből megállapíthatjuk, hogy a mag-burok algoritmus esetén érdemes nagy hangsúlyt fektetni a vermet használó találgatások műveletigényeire. Továbbá kiderítettük, hogy a vizsgált algoritmusok mindegyikének más a futásidőre érzékeny pontja. A mag-burok és a komponensek elhelyezkedésén alapuló algoritmus érzékeny a komponens számra illetve a vizsgált kép méretével jelentősen nő a futás idő. A mag-burok további tulajdonsága, hogy a futásidőket a komponensek elhelyezkedése is befolyásolja. A szimulált hűtésen alapuló rekonstrukciós algoritmus futásidejét a kapcsoló komponensek mennyisége és a hűtés paraméterezése határozza meg. Ezzel az utóbbi módszerrel rövidebb futásidőket érhetünk el, viszont a rekonstrukció alatt nincs garantálva hogy a kapott kép *hv*-konvex.

További tervekben meg szeretnénk vizsgálni az algoritmusokat nagyobb komponens számú képekre. Még jobban megvizsgálni, mikor lassú a mag-burok algoritmus és ezek alapján tovább fejleszteni. Ennek eredményeképpen kidolgozhatóak lennének olyan algoritmusok melyek a komponensek egymáshoz viszonyított helyzetéről meglévő előzetes tudást is figyelembe tudnák venni.

Irodalomjegyzék

- [1] P. Balázs, Képrekonstrukció, Budapest, TYPOTEX, 2011
- [2] G. W. Woeginger, The reconstruction of polyominoes from their orthogonal projections, *Inform. Process. Lett.* **77** (2001) 225-229.
- [3] E. Barcucci, A. Del Lungo, M. Nivat, R. Pinzani, Reconstructing convex polyominoes from horizontal and vertical projections, *Theor. Comput. Sci.* **155** (1996) 321-347.
- [4] A. Kuba, Reconstruction of two-directionally connected binary patterns from their two orthogonal projections, *Comp. Vision, graphics, and Image Proc.* **27** (1984) 249-265.
- [5] F. Jarray, G. Tlig, A simulated annealing for reconstructing hv-convex binary matrices, *Discrete Applied Mathematics*, 2010, 447-454
- [6] G. Dahl and T. Flatberg, Optimalization and reconstruction of hv-convex (0,1)-matrices.
- [7] P. Balázs, Reconstruction of canonical hv-convex discrete sets from horizontal and vertical projections, IWCIA 2009, *Lecture Notes in Comput. Sci.* **5852** (2009) 280–288.
- [8] P. Balázs, Complexity results for reconstructing binary images with disjoint components from horizontal and vertical projections
- [9] P. Balázs, Discrete tomographic reconstruction of binary images with disjoint components using shape information, *Journal of Shape Modeling* **14:2** (2008) 189–207.