

Szegedi Tudományegyetem
Informatikai Tanszékcsoporth

**h_v-konvexitáson alapuló bináris tomográfiái
algoritmusok tervezése és elemzése**

Diplomamunka

Készítette:

Ozsvár Zoltán

programtervező informatikus Msc
szakos hallgató

Témavezető:

Dr. Balázs Péter

egyetemi adjunktus

Szeged
2014

Tartalomjegyzék

Feladatkiírás	3
Tartalmi összefoglaló	4
Bevezetés	5
1. Alapfogalmak	7
2. Algoritmusok hv-konvex bináris képek rekonstruálására	10
2.1. Heurisztikák a hv -konvex képek rekonstruálására	10
2.2. H-TILES: h -konvex diszjunkt komponenseken alapuló rekonstrukció . .	12
2.3. H-TILES_SA: h -konvex diszjunkt komponenseken alapuló rekonstrukció szimulált hűtéssel	17
2.4. REGION_GROWTH: Rekonstrukció 4-szomszédokon való régió növelés segítségével	21
3. Eredmények	26
3.1. h -konvex diszjunkt halmazokon alapuló rekonstrukciós eljárás vizsgálata	26
3.2. h -konvex diszjunkt komponenseken alapuló szimulált hűtéses rekonstrukció vizsgálata	28
3.3. 4-szomszédokon való régió növeléses rekonstrukciós eljárás vizsgálata . .	30
4. Összegzés	36
4.1. További tervek	36
Nyilatkozat	38
Köszönetnyilvánítás	39
Irodalomjegyzék	40

Feladatkiírás

A bináris tomográfiában a kevés vetület eredményezte nagyméretű megoldáshalmaz méretének csökkentése érdekében kiegészítő korlátokat vezetünk be, melyek a kép geometriai jellemzőit írják le, mint az összefüggőség vagy a hv -konvexitás. A dolgozat célja egy konkrét geometriai kényszerfeltétel, a hv -konvexitás, rekonstrukcióra gyakorolt hatásának vizsgálata. A probléma elméleti szempontból is érdekes, ugyanis a hv -konvex bináris képek rekonstrukciójára a további feltételektől függően egyes esetekben létezhet polinom idejű algoritmus, míg más esetekben a rekonstrukció NP-teljes is lehet. A két vetületből történő rekonstrukció az általános hv -konvex mátrixosztályban NP-teljes. Egy korábbi munkánkban két heurisztikus és egy optimalizációs eljárás alapján rekonstrukciós algoritmust implementáltuk és ezek hatékonyságát vizsgáltuk meg a fenti osztály esetében. A kapott adatok alapján feltérképeztük, hogy a különböző algoritmusok esetén miben rejlik a probléma nehézsége a rekonstruálás során. A feladat következő lépéseként, további célunk az eddigi ismereteink felhasználása új, hatékonyabb bináris képeket rekonstruáló algoritmusok tervezéséhez. A dolgozatban megvizsgálandó probléma az, hogy a vetületekből, hogyan alkotható meg a kép, ha azt nem pixelenként, hanem nagyobb alakzatok (úgy nevezett csempék) diszjunkt uniójaként kívánjuk létrehozni.

Tartalmi összefoglaló

A dolgozatban célunk hv -konvex bináris képek rekonstruálása két vetületből. Általános hv -konvex bináris mátrixok esetén a rekonstrukció ismerten NP-teljes. A kevés vetület eredményezte nagyméretű megoldáshalmaz méretének csökkentése érdekében kiegészítő feltételeket vezetünk be, melyek a kép geometriai jellemzőit írják le. Az alábbi munkában kidolgozunk három új algoritmust, melyek a fenti rekonstrukciós feladat megoldásait keresik. Az egyik a rekonstrukciót diszjunkt h -konvex halmazok segítségével oldja meg, a másik ennek egy változata, mely szimulált hűtést alkalmaz a probléma megoldására. Bemutatunk továbbá egy olyan algoritmust is, mely 4-szomszédokkal való régiónövelés segítségével próbálja meg rekonstruálni a hv -konvex képeket. A kidolgozott algoritmusok teljesítményét egy már közölt algoritmussal hasonlítjuk össze.

Kulcsszavak: bináris tomográfia; rekonstrukció; hv -konvexitás; diszjunkt régiók; 4-összefüggőség.

Bevezetés

A *Tomográfiai* eljárások során célunk tárgyak belsejének háromdimenziós modelljének megalkotása a vizsgált objektum roncsolása nélkül. A megfigyelt tárgy modelljét annak keresztmetszeti képeiből építjük fel, melyeket vetületeik alapján határozzunk meg. Egy keresztmetszet tekinthető egy kétváltozós függvénynek, míg a vetületek az adott függvény különböző irányokból vett vonalmenti integráljaiból állnak elő. A vizsgálat során sugárzással (például röntgen-, neutron- vagy elektronsugárzással) átvilágítjuk a testet. Ez a sugárzás különböző anyagi minőségű és/vagy eltérő vastagságú részeken eltérő mértékben hatol át. Az áthatolt sugárzást detektorokkal mérjük. Az így kapott adatok meghatározzák az objektum vetületeit, melyekből információhoz juthatunk a tárgy belsejével kapcsolatban. A tomográfiának számos alkalmazási területe ismert, mint például az orvosi képalkotás, nem-roncsoló vagy elektron mikroszkópos vizsgálatok. Az úgy nevezett szűrt visszavetítés (*FBP* - Filtered Backprojektion) a leggyakrabban alkalmazott rekonstrukciós eljárás, mely azonban a vizsgált objektumról több száz vetület készítését igényli. A vetületképzés során alkalmazott sugarak viszont károsíthatják a megfigyelt testet, emellett időigényes és költséges is lehet a vizsgálat, így nem mindig adódik lehetőség az *FBP* eljárás használatára.

A *diszkrét* tomográfiát akkor alkalmazhatjuk, ha az objektum ábrázolása során csak néhány előre ismert elnyelődési együttható jelenik meg. Ezzel az információval a szükséges vetületek száma drasztikusan csökkenthető, esetenként akár 10-20 vetület is elegendő lehet a kívánt képminőség eléréséhez [1, 2]. A *Bináris tomográfia* során feltételezzük, hogy a vizsgált objektum homogén. A képpontok csak két értéket vehetnek fel, ahol 0 és 1 rendre a vizsgált anyag hiányát és jelenlétét jelöli az adott pozíciókban. Ezáltal a megkötés által sokszor már valóban kevés (2-10) vetület segítségével is rekonstruálható megfelelő minőségű eredmény. A bináris megszorításnak köszönhetően egyes esetekben akár 2 vetületből is előállítható a kép, viszont a vetületek alacsony száma miatt a probléma gyakran aluldefiniált, vagyis ugyanazon vetületeknek több különböző bináris kép is megfelelhet. A megoldáshalmaz méretének csökkentése érdekében kiegészítő feltételeket szokás meghatározni, amelyek a vizsgált kép további geometriai tulajdonságaira vonatkozhatnak. Ilyenkor a rekonstruált mátrixnak (képnek) ki kell elégítenie az előre megadott

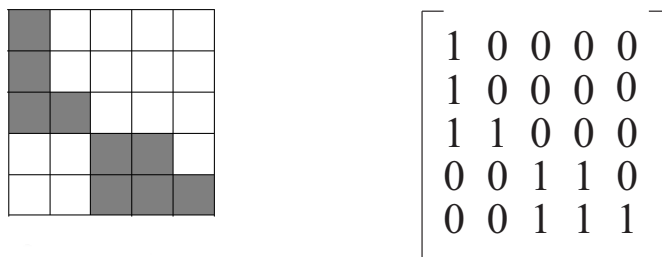
sor és oszlop összegek (vetületek) mellett egyéb geometriai feltételeket is. Bizonyos geometriai tulajdonságok esetén létezik polinomiális idejű rekonstrukciós algoritmus, míg más esetekben a rekonstrukció NP-nehéz.

A dolgozat célja kettő új *h_v-konvexitáson* alapuló, a szakirodalomban még nem közölt rekonstrukciós algoritmus bemutatása és elemzése. A 1. fejezet a dolgozatban leírtak megértéséhez szükséges fogalmakat ismerteti. A 2. fejezet az algoritmusokat mutatja be. Végül a 3. fejezet a két algoritmusnak az elemzését és az abból levont következtetéseket tárgyalja.

1. fejezet

Alapfogalmak

Tetszőleges kétdimenziós kép számszerű leképezését *digitális képnek* nevezzük, aminek legkisebb eleme a pixel (vagy képpont). Egy digitális kép véges számú pixelből áll, és ezen képpontok csak egész értéket tartalmazhatnak. A *bináris kép* egy olyan digitális kép, mely a képpontokban csak kétféle értéket vehet fel. A bináris képeket kétdimenziós bináris mátrixokként is ábrázolhatjuk (1.1. ábra). A következőkben a félreértések elkerülése végett feltételezzük, hogy mindkét reprezentáció esetében az oszlopok számozását balról jobbra végezzük, illetve a sorokét fentről lefelé. Így a bal felső sarokban levő képpont az (1,1) indexű elem.



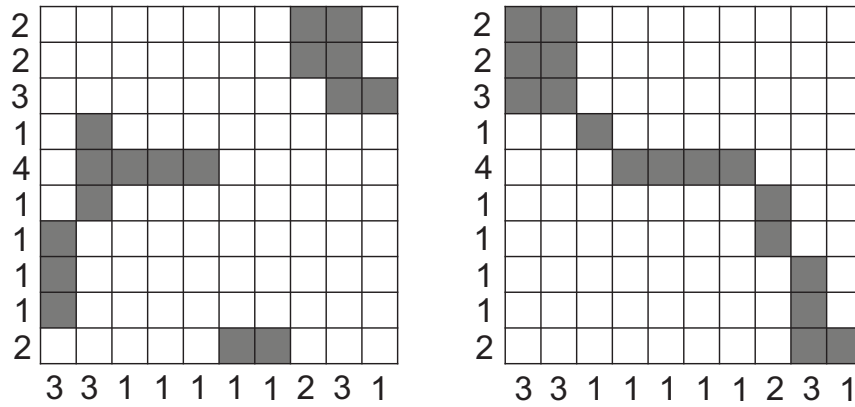
1.1. Ábra. Egy bináris kép két különböző ábrázolása: mint pixelek tömbje (bal oldalt) és mint bináris mátrix (jobb oldalt)

Egy $m \times n$ méretű bináris kép tehát reprezentálható egy $A = (a_{ij})_{m \times n}$ bináris mátrixsal, ahol az egyesek azokat a pontokat jelzik, melyek a vizsgált terület egy objektumpontját tartalmazzák, illetve nullák azokat amelyek háttérpontok. A továbbiakban feltételezzük, hogy az $m \times n$ méretű bináris mátrix minden sora és oszlopa legalább egy egyes értéket tartalmaz. Az A bináris mátrix *horizontális* és *vertikális* vetületén rendre az $R(A) = (h_1, h_2, \dots, h_m)$ és $S(A) = (v_1, v_2, \dots, v_n)$ vektorokat értjük, ahol

$$h_i = \sum_{j=1}^n a_{ij} \quad (i = 1, \dots, m), \quad v_j = \sum_{i=1}^m a_{ij} \quad (j = 1, \dots, n). \quad (1.1)$$

Jelölje $\mathcal{M}$ az $m \times n$ méretű mátrixok egy adott osztályát. A bináris rekonstrukció során adottak az $m, n \in \mathbb{N}$ egész számok és az $R \in \mathbb{N}^m$ és $S \in \mathbb{N}^n$ vektorok, feladatunk pedig egy olyan $A \in \mathcal{M}$ mátrixot konstruálni, melyre $R(A) = R$ és $S(A) = S$.

A rekonstrukció során előfordulhat, hogy több megoldás is kielégíti ugyanazon vetületeket (1.2. ábra). Ennek oka a *kapcsoló komponensek* jelenléte, amik egy bináris mátrix $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ és $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ részmátrixai. A kapcsoló komponensek pozícióiban lévő értékeket invertálva egy olyan bináris mátrixot kapunk, melynek horizontális és vertikális vetületei ugyanazok, mint az eredeti mátrixnak. Tudjuk, hogy az ilyen kapcsoló komponens jelenléte szükséges és elégséges feltétele annak, hogy a megoldás ne legyen egyértelmű. Ismert továbbá, hogy ha az A és B különböző mátrixok ugyanazon horizontális és vertikális vetületekkel rendelkeznek, akkor az A mátrixból el lehet jutni B mátrixba (vagy fordítva) kapcsoló komponensek véges számú átkapcsolásával [10].

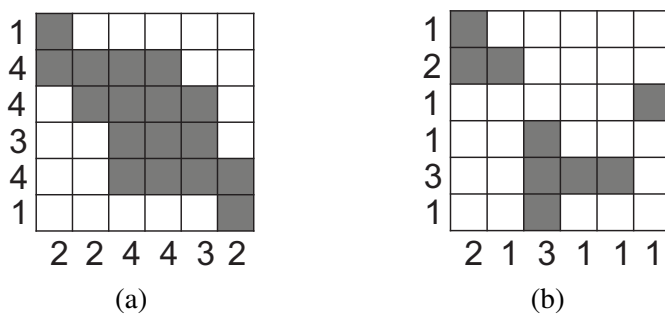


1.2. Ábra. Két különböző kép, melyek vetületei megegyeznek.

A rekonstrukcióhoz tehát szorosan kapcsolódik az egyértelműség eldöntésének kérdése is, azaz hogy egy $A \in \mathcal{M}$ mátrixhoz létezik-e olyan $A' \in \mathcal{M}$ bináris mátrix, melyre $R(A') = R(A)$ és $S(A') = S(A)$, de $A' \neq A$.

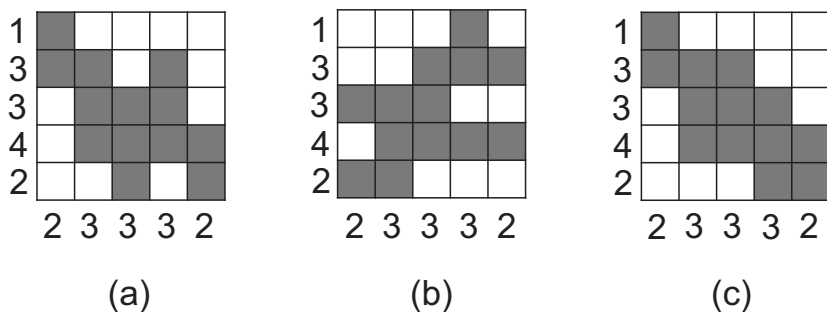
Különböző geometriai megszorításokkal csökkenthetjük a megoldáshalmaz méretét. Ilyen megszorítások lehetnek az *összefüggőség* és a *konvexitás*. Egy A bináris mátrix két pontja, $P = (p_1, p_2)$ és $Q = (q_1, q_2)$, *4-szomszéd*, ha $|p_1 - q_1| + |p_2 - q_2| = 1$. A $P_0, \dots, P_k$ különböző pontokból álló sorozat *4-út* a P_0 pontból P_k pontba, ha a sorozat minden pontjában A értéke 1 és P_l és P_{l-1} *4-szomszédok* minden $l = 1, \dots, k$ -ra. Az A bináris mátrix *4-összefüggő*, ha bármely két 1-es értékű pontja között van A -beli *4-út*. A *4-összefüggő* halmazokat *poliominóknak* is nevezzük. Ha egy bináris mátrix nem *4-összefüggő*, akkor több poliominóból áll. Az ilyen maximális *4-összefüggő* részeképek az A bináris mátrix *komponenseit* adják (1.3. ábra).

Egy bináris mátrix *horizontálisan konvex* (vagy *h-konvex*) ha minden sorában az egyesek egy összefüggő sorozatot alkotnak. Hasonlóan egy bináris mátrix *vertikálisan konvex*



1.3. Ábra. Egy 1 komponenst (a), és egy 3 komponenst (b) tartalmazó kép

(vagy v -konvex), ha minden oszlopában az egyesek egy összefüggő sorozatot alkotnak. Ha egy bináris mátrix horizontálisan és vertikálisan is konvex, akkor hv -konvexnek nevezzük (1.4. ábra).



1.4. Ábra. Egy 4-összefüggő v -konvex bináris kép (a), egy 4-összefüggő h -konvex bináris kép (b), egy 4-összefüggő hv -konvex bináris kép (c) ugyanazon vetületekkel.

2. fejezet

Algoritmusok hv -konvex bináris képek rekonstruálására

A 4-összefüggő hv -konvex bináris mátrixok rekonstrukciója két vetületből polinomiális időben megoldható. Erre vonatkozóan két megközelítést ismerünk. Az egyik egy úgynevezett *mag-burok* eljárás [3, 4], a másik pedig 2SAT problémán alapszik [5]. Az általános hv -konvex bináris mátrixok esetén azonban a feladat NP-teljessé válik, így nem várható, hogy polinomiális idejű algoritmust találunk rá [6]. Számos heurisztikus algoritmus létezik, mely igyekszik hatékony választ adni az általános hv -konvex rekonstrukció kérdésére. Egy előző munkában rámutattunk, hogy ezek teljesítménye nagyon különböző lehet, melyeket befolyásolhat a kép mérete, illetve a rekonstruálni kívánt képen levő komponensek pozíciói, nagysága és darabszáma [7]. Ebben a fejezetben először ismertetjük a szimulált hűtési keretrendszerünket. Majd bemutatunk egy referenciának szánt algoritmust, mely kapcsoló komponensek segítségével próbálja megtalálni a feladat megoldását. Utána ismertetjük az általunk kidolgozott algoritmusokat, melyeknél az első h -konvex diszjunkt régiókat, a másik pedig pixeleket és azokkal való műveleteket alkalmaz a rekonstrukció során.

2.1. Heurisztikák a hv -konvex képek rekonstruálására

A hv -konvex rekonstrukcióra kidolgozott legelső heurisztikus módszer az úgynevezett *mag-burok* algoritmus. Ennek lényege, hogy a vetületek alapján két halmazt különböztet meg, amik alapján elvégzi a rekonstrukciót [8]. Az egyik halmaz a *mag*, ami azokat a pontokat tartalmazza amelyek biztosan a megoldáshoz fognak tartozni. A másik a *burok*, ami azokat a pontokat foglalja magában, amelyek esetlegesen hozzátartozhatnak a megoldást reprezentáló bináris képhez. Az algoritmus első lépésében a *mag* halmaza üres, míg a *burok* az összes képpontot tartalmazza. Minden iterációban megpróbáljuk növelni a *magot*

és utána a horizontális és vertikális vetület segítségével csökkenteni a burkot. Amennyiben egy iterációs során nem növekszik a mag és nem csökken a burok, akkor a veremmemóriában eltároljuk az aktuális állapotot és egy tetszőleges pontot a maghoz adunk, ezzel kilépve ebből az állapotból. Akkor kapunk megoldást, ha a két halmaz megegyezik. Ha a mag nem egyenlő és nem is részhalmaza a buroknak és a veremmemória üres, akkor nincs megoldás az adott vetületekre. Viszont ha van a veremmemóriában elmentett állapot, akkor beolvassuk a veremmemória tetején levő állapotot, a korábban választott pont nem fog sem a maghoz, sem a burokhoz tartozni, és tovább iterálunk.

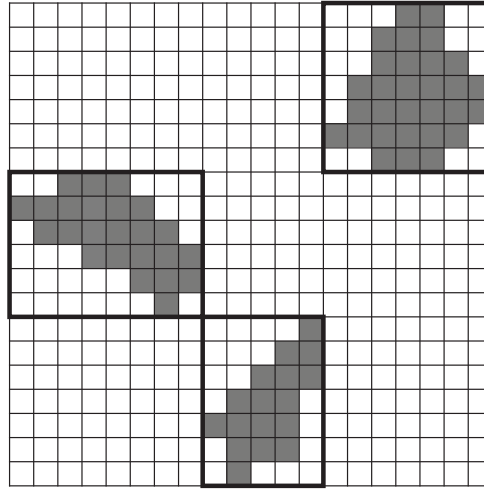
Egy másik algoritmus egy optimalizációs eljárásan alapul [9]. Szimulált hűtést használ, ami a kezdőállapotból indul az optimum felé. A kiindulási állapotot Ryser algoritmus segítségével állítjuk elő [10]. A Ryser algoritmus eredménye általában nem *hv*-konvex, ezért célunk, hogy minél *hv*-konvexebb legyen a végső állapot. A optimalizálás során megengedünk az aktuálistól rosszabb állapotokat is, ezáltal nem ragadunk be lokális optimumokba. A konvexitás mértékét a szomszédos egyesek száma adja meg. A kapcsoló komponensek átkapcsolásával pedig az egyik állapotból a másikba jutunk. Eredményt akkor kapunk, ha hosszabb ideig nem sikerül optimálisabb eredményt elérni, vagy elértünk az kilépési feltételben megadott hőmérsékletet.

A harmadik ismert *hv*-konvex képeket rekonstruáló eljárás az úgynevezett diszjunkt komponensek elhelyezkedésén alapuló algoritmus [11, 12]. A *hv*-konvex képeknél a komponenseket körül lehet rajzolni diszjunkt minimális befoglaló téglalapokkal (2.1. ábra). Ennek az algoritmusnak az alapja, hogy a kapott képet először megpróbálja a komponenseket befoglaló téglalapokra bontani, majd ezeket a diszjunkt régiókat egyesével rekonstruálja (1. Algoritmus). Az így kapott befoglaló téglalapokban elhelyezkedő komponensek 4-összefüggő *hv*-konvex alakzatok, amiket polinomiális idő alatt lehet rekonstruálni.

Legyen az A bináris képnek k darab diszjunkt komponense, ahol $I_l \times J_l = [i_l, i'_l] \times [j_l, j'_l]$ az l -edik komponens minimális befoglaló téglalapja ($l = 1, \dots, k$). A komponensek akkor diszjunktak, ha bármely $l \neq l'$ (ahol $1 \leq l, l' \leq k$) esetén $I_l \cap I_{l'} = \emptyset$ és $J_l \cap J_{l'} = \emptyset$.

2.1. Definíció. Legyen $\mathcal{S}$ poliomínók osztálya, $R \in \mathbb{N}^m$ és $S \in \mathbb{N}^n$. Az $[i_l, i'_l] \times [j_l, j'_l]$ téglalap kompatibilis az $\mathcal{S}$ osztályra nézve, ha létezik olyan $P \in \mathcal{S}$ poliomínó, melyre $R(P) = (h_{i_1}, \dots, h_{i_2})$ és $S(P) = (v_{j_1}, \dots, v_{j_2})$.

Ezen algoritmusok vizsgálata során, ahol rekonstrukció pontosságát és futásidejét mértük a diszjunkt komponenseken alapuló algoritmus teljesített a legjobban [7]. Ezért referencia algoritmusként ezt az eljárást fogjuk a későbbiekben használni.



2.1. Ábra. Bináris kép három diszjunkt komponenssel. A minimális befoglaló téglalapokat vastag vonalak ábrázolják

Algoritmus 1: Diszjunkt komponensek elhelyezkedésén alapuló algoritmus

Input: $R \in \mathbb{N}^m$ és $S \in \mathbb{N}^n$ két vektor;

Output: *hv*-konvex bináris kép az R sor- és S oszlopösszegekkel, vagy HIBA (ha nincs ilyen).

- 1 Keressük meg és tároljuk el az összes kompatibilis intervallum párt $[i_1, i_2]$ H -ra és $[j_1, j_2]$ V -re;
 - 2 Az eltárolt intervallum párok között keressük meg azt az összeállítást, ahol minden komponensre érvényes, hogy
 - a) sor intervallumok egymást követők és páronként diszjunktak, és
 - b) oszlop intervallumok páronként diszjunktak;
 - 3 Ha a két intervallum sorozat nem fedi le a teljes H és V vektorokat, akkor HIBA. Egyébként rekonstruáljuk a befoglaló téglalapokban a komponenseket.
-

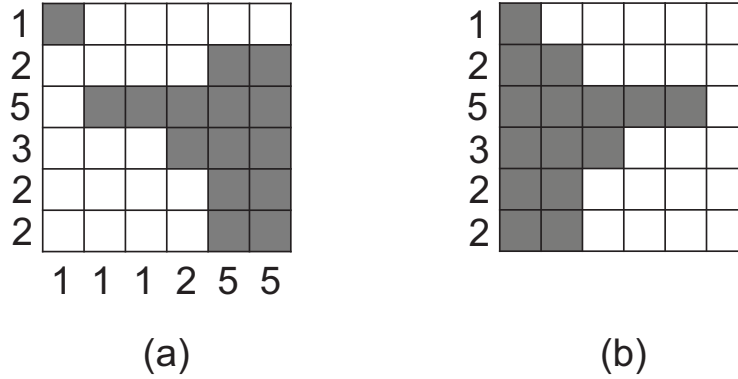
2.2. H-TILES: *h*-konvex diszjunkt komponenseken alapuló rekonstrukció

A szakirodalomban található bináris képeket rekonstruáló algoritmusok gyakran pixelalapúak, ami azt jelenti, hogy a rekonstrukció során alkalmazott lépeseknél egy-egy képpont változtatásával közelítenek a megoldás felé. A most ismertetésre kerülő algoritmus pixelek helyett nagyobb alakzatok, úgy nevezett csempék, diszjunkt uniója segítségével próbálja meghatározni a kapott vetületekből a képet [13]. A továbbiakban erre az algoritmusra H-TILES néven fogunk hivatkozni.

Az algoritmus alapötlete a következő. Mindegyik sorban elhelyezünk egy összefüggő egyesekből álló „csíkot” (csempét), úgy hogy az adott sor vetületi értéke határozza meg a csempe hosszát. Természetesen az így elhelyezett elemek *h*-konvexek, továbbá diszjunktak. Ezen elemeket a függőleges vetületek alapján is létre lehetne hozni, mi azonban

az egyszerűség kedvéért csak a vízszintes vetületek alapján határozzuk meg a kiindulási képet.

Például ha van egy képünk (2.2 (a). ábra), melynek a vetületei $R = (1, 2, 5, 3, 2, 2)$ és $S = (1, 1, 1, 2, 5, 5)$, akkor 6 darab h -konvex diszjunkt elemmel fogunk foglalkozni, melyek mérete az első sortól az utolsóig: 1, 2, 5, 3, 2, 2.



2.2. Ábra. Bináris kép, melynek vetületei: $R = (1, 2, 5, 3, 2, 2)$ és $S = (1, 1, 1, 2, 5, 5)$ (a), az adott képből kapható h -konvex elemek, ahol minden elem az első oszloptól kezdődik (b)

A rekonstrukció során a csempeket horizontálisan mozgathatjuk. Célunk, hogy megtaláljuk a csempek pozícióinak azon kombinációját, mely kielégíti a vertikális vetületet. További feltétel, hogy az eredmény hv -konvex legyen. Mivel az említett elemek kivétel nélkül vízszintesen egybefüggőek, ezért a h -konvexitás garantált az algoritmus egész futása alatt, így csak a v -konvexitást kell ellenőriznünk a rekonstrukció során.

Az eljárás pszeudokódját a 2. algoritmus írja le. A rekonstrukciót egy üres (objektum képpont nélküli) mátrixszal kezdjük (*Elemek inicializálása*). A csempeket megpróbáljuk egyesével elhelyezni. A csempek elhelyezésének a sorrendje az eredeti algoritmus szerint a következő [13]. Első elemnek kiválasztjuk a legnagyobbat, melynek a legkisebb a sor indexe. Az ezután következő elemek esetén csak olyan elemet választhatunk, mely szomszédos egy már elhelyezett elemmel (két elem akkor szomszédos, ha sorindexeik különbsége 1). Továbbá igyekszünk az elhelyezhető elemek közül a nagyobb méretűt választani. Ezt az elemválasztási sorrendet az alábbiak alapján módosítottuk. Állítsunk elő egy a rekonstruálni kívánt kép méretével megegyező méretű mátrixot. Az elemei az adott sor vertikális vetületének az értékét fogják tartalmazni. Ezután jelöljük az i -edik sorban levő elem hosszát l_i -vel. Minden sorban kiszámoljuk a mátrixban levő első l_i elemek összegét, majd elosztjuk l_i -vel, jelölje ezt az értéket k_i . Először azt az elemet választjuk, amelyik sorban a legnagyobb ez a k_i érték:

$$k_i = \frac{\sum_{j=1}^{l_i} v_j}{l_i} \quad (i = 1, \dots, m; l_i \geq 1) \quad (2.1)$$

Amennyiben több ilyen elem van, akkor a legkisebb indexűt fogjuk választani (2.3. ábra). Következő elemnek csak olyan elemet választhatunk mely szomszédos egy már elhelyezett elemmel. Ha több ilyen elem van, akkor az választjuk, melynek nagyobb a k_i értéke. Ezt a műveletet az *UjElemetValaszt()* függvény végzi el.

1	3	3	2	3	2
1	3	3	2	3	2
4	3	3	2	3	2
4	3	3	2	3	2
2	3	3	2	3	2
	3	2	2	3	2

2.3. Ábra. k_i értékek kiszámítása egy példán keresztül. Ahol $k_1=3$; $k_2=3$; $k_3=2.25$; $k_4=2.25$; $k_5=2.5$. Tehát k_1 a legnagyobb elem, melynek a legkisebb az indexe, ezért először az első sorban levő elemet választjuk.

Mivel mindig egy elhelyezett elemmel szomszédos elemet helyezünk el, ezért a v -konvexitást elég csak az aktuálisan elhelyezett elemre megvizsgálni. Egy elem elhelyezése során az alábbi szempontokat kell szem előtt tartani. Egy elemet csak abba a pozícióba lehet elhelyezni, ahol az oszlopokban szereplő egyesek összege nem haladja meg az adott vertikális vetületi értékeket, mivel az elhelyezett elemek függőleges vetületei soha nem lehetnek nagyobbak a feladatban megadott vetületeknél. Továbbá az elhelyezett elemmel kapott képnek v -konvexnek kell lennie. Az 2. algoritmusban ezt az ellenőrzést a *EltudtaHelyezni(Elem)* függvény végzi, ami a paraméterben kapott elemre vizsgálja meg a fentebb leírt feltételeket. Minden iteráció során az úgynevezett *aktuális vetületek* alapján dolgozunk, melyek az jelzik, hogy az adott iterációban még hány objektum pont hiányzik a sorokban vagy oszlopokban.

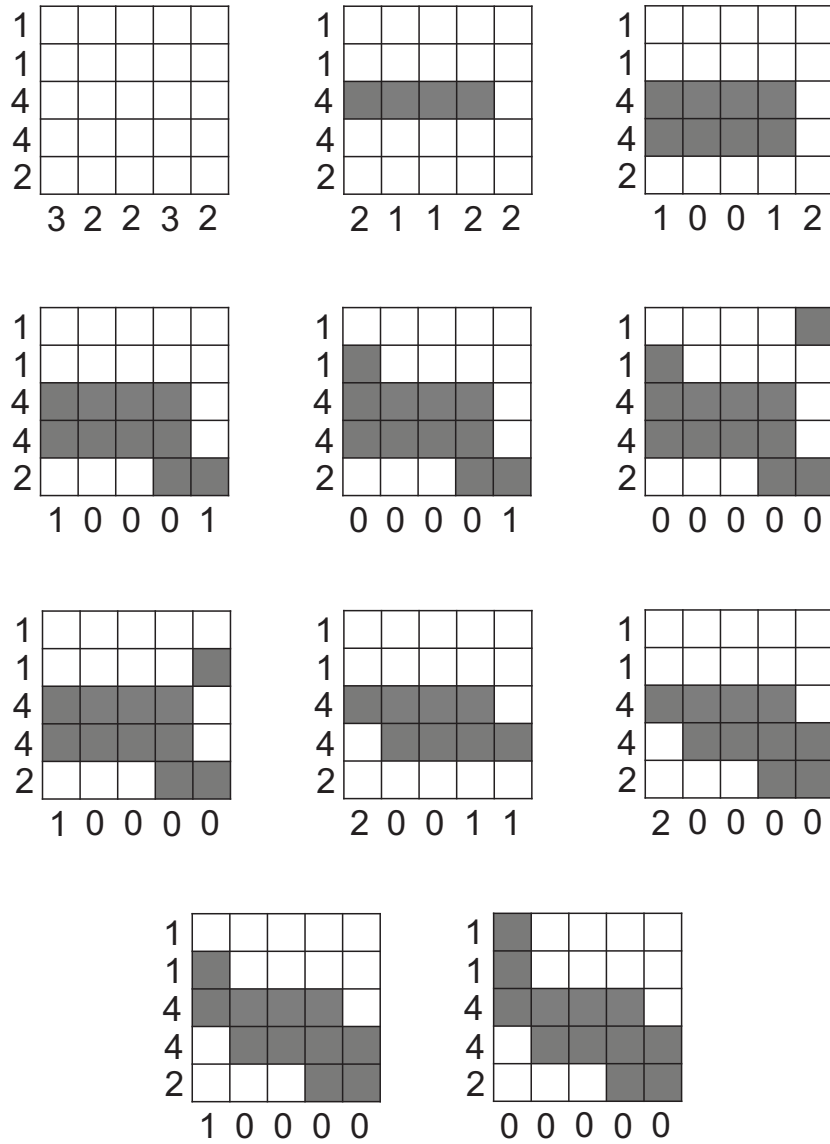
$$h_{akt} = h_j - \sum_{i=1}^n a_{ij} \quad (i = 1, \dots, m); \quad v_{akt} = v_i - \sum_{j=1}^m a_{ij} \quad (j = 1, \dots, n). \quad (2.2)$$

A rekonstrukció során az elhelyezett elemeket veremmemóriában tároljuk. Az elemek elhelyezése során, ha az adott csempe még nincs elhelyezve, akkor a legkisebb oszlop indexű érvényes pozícióba helyezzük el. Ha már el van helyezve, viszont el kell mozdítanunk, akkor a tőle jobbra levő legkisebb oszlop indexű érvényes pozícióba tesszük.

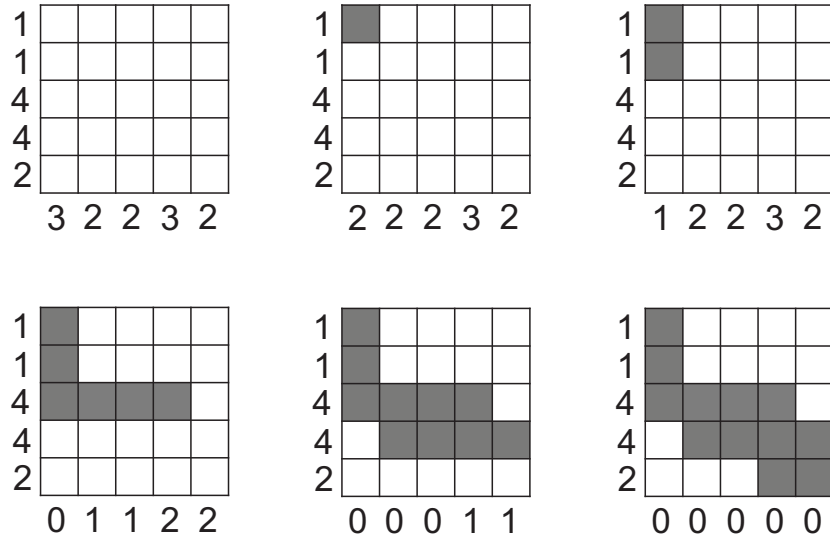
Abban az esetben, ha nem tudunk következő pozíciót választani "felvesszük" az elhelyezett elemet, és a veremmemória segítségével az előtte lerakott elemet próbáljuk meg elmozdítani. Ha egy csempét "felvettünk" a tábláról, akkor az nem fog szerepelni a képen, illetve következő elhelyezésnél megint a legkisebb oszlopindexű érvényes pozícióban fogjuk elhelyezni. Megoldást akkor találunk, ha az összes diszjunkt régiót elhelyeztük és az elhelyezett elemek vertikális vetülete megegyezik az adott feladatra előírt vertikális vetülettel. Ezeket a feltételeket a *NincsMegoldas()* függvény segítségével vizsgáljuk.

A 2.4. ábra egy példát mutat az algoritmus lépéseire az elemek eredeti választási sorrendje szerint. Először a legnagyobb elemet választjuk, ami a harmadik sorban található. Ezt elhelyezzük az első érvényes pozícióban, ami a első oszlop lesz. A következő elem, amit választunk a negyedik sorban levő lesz, amit szintén az első oszlopban helyezünk el. Az utolsó elem elhelyezése során egy nem *v*-konvex alakzatot kapunk, ezért vissza kell lépni a verem memória segítségével. felvesszük a legutolsó elemet és az előtte lerakott elemet jobbra mozgatjuk, ami megint nem *hv*-konvex alakzatot ad. Ezért megint visszalépünk a verem memória segítségével. Felvesszük az elemet, majd az ötödik sornál nem tudunk más érvényes pozíciót biztosítani, ezért abban a sorban lévő csempét is felvesszük és a harmadik sorban levő elemet léptetjük eggyel jobbra. Ezután minden elemet csak egy helyre tudunk helyezni, és az utolsó elem lerakása után megkapjuk az eredményt.

A 2.5. ábra egy példát mutat, ahol a vetületek ugyanazok mint az előző esetben, viszont az algoritmus az elemeket módosított választási sorrend szerint választja. Először az első sorban lévő elemet választjuk, mert k_1 a legnagyobb értékű, majd sorba megyünk az oszlopokon. Az összes elemet az első érvényes pozícióra helyezve megkapjuk a megoldást.



2.4. Ábra. A H-TILES algoritmus lépései az elemek eredeti kiválasztási sorrendje alapján, ahol a sorokban található számok a elemek méreteit, az oszlopokban a szükséges egyesek számát jelzik



2.5. Ábra. A H-TILES algoritmus lépései az elemek módosított kiválasztási sorrendje alapján, ahol a sorokban található számok a elemek méreteit, az oszlopokban a szükséges egyesek számát jelzik

Algoritmus 2: H-TILES

Input: $R \in N^m$ és $S \in N^n$ két vektor; verem $S = \emptyset$; utolsoLerakottElem = NULL;

Output: elhelyezett elemek, melyek a rekonstruált bináris eredmény képet reprezentálják, ahol az eredmény $h\nu$ -konvex. Különben HIBA.

```

1 Elemek Inicializálása;
2 while (NincsMegoldas()) do
3   utolsoLerakottElem = UjElemetValaszt();
4   while (utolsoLerakottElem.Elhelyez()) do
5     if ( $S = \text{üres}$ ) then
6       | utolsoLerakottElem = S.pop();
7     end
8   end
9   if (EltudtaHelyezni(utolsoLerakottElem)) then
10    | utolsoLerakottElem = S.push();
11  end
12 end

```

2.3. H-TILES_SA : h -konvex diszjunkt komponenseken alapuló rekonstrukció szimulált hűtéssel

A H-TILES algoritmus futtatása során ha nagy méretű képet rekonstruálunk, akkor rengeteg kombinációt kell kipróbálni, hogy bejárjuk az egész keresési teret, ami nem célszerű. Ezért optimalizálási feladatként próbáljuk megoldani. Ugyanúgy h -konvex diszjunkt csempéket használunk, melyeknek a megfelelő elhelyezése esetén megkaphatjuk a meg-

oldást. Ennek megtalálásához szimulált hűtést fogunk alkalmazni. Az eljárást a 2.3. algoritmus írja le. A kezdeti állapot kialakításához a *h*-konvex elemeket véletlenszerűen helyezzük el, minden elem egy véletlenszerűen választott oszlopban fog elhelyezkedni, úgy hogy ne lógjon ki egyik vége sem az $m \times n$ -es területről (*Init()*). Az optimalizálás során a kezdő hőmérséklet 50. Az eljárás terminál, ha a hőmérséklet eléri a kilépési feltételben megszabott 0.005 értéket vagy ha 500 iteráció során nem talált jobb megoldást. Minden iterációban véletlenszerűen (egyenletes eloszlás alapján) választunk egy elemet, amit egy szintén véletlenszerűen választott pozícióba fogunk áthelyezni úgy, hogy az $m \times n$ -es területről ne lógjon ki egyik oldala sem (*ElemMozgat()*). Az *f* célfüggvény értékét az alábbi módon határozzuk meg. Először számítsuk ki a vetületi hibát a vertikális vetületre. Ehhez az eredeti vetületek és az aktuális vetület abszolút különbségét állapítjuk meg:

$$P = \sum_{j=1}^n |v_j^o - v_j^r| \quad (v_i^o \in R^o; v_i^r \in R^r; j = 1, \dots, n). \quad (2.3)$$

Majd megállapítjuk a *v*-konvexitásnak a mértékét, amit úgy határozunk meg, hogy összeszámoljuk az oszlopokban levő vertikális szomszédokat:

$$K = \sum_{i=1}^{m-1} \sum_{j=1}^n a_{ij} a_{(i+1)j} \quad (i = 1, \dots, m; j = 1, \dots, n). \quad (2.4)$$

Ezután az *f* függvény értéket a *v*-konvexitásnak és a vetületi hiba értékének a különbsége fogja meghatározni, ahol a γ paraméter elősegíti, hogy azonos nagyságrendű legyen ez a két érték. A γ paraméter értéke egy becslés, ami az maximális *v*-konvexitás és maximális vetületi hiba hányadosa. Ez általában 1.5 körül szokott lenni, ezért a rekonstrukciók során ezt az értéket fogjuk használni. Tehát az alábbi képletet alkalmazzuk:

$$f = K - (\gamma \cdot P). \quad (2.5)$$

Egy állapotot akkor fogadunk el, ha jobb mint az előző állapot. Amennyiben ez a feltétel nem teljesül, akkor is adunk esélyt, hogy elfogadjunk a rosszabb állapotot. Így jó eséllyel nem ragad be az algoritmus a lokális maximumokba. Az eljárás során folyamatosan csökken annak a valószínűsége, hogy elfogadjuk a rosszabb állapotokat a $p < \exp(\frac{-|d|}{T_{emp}})$ feltétel segítségével, ahol $0 < p < 1$ egy véletlen szám és $d = f(elozo) - f(aktualis)$. Amennyiben elfogadunk egy állapot, akkor elmentjük azt, csökkentjük a hőmérsékletet, és lenullázzuk a számlálót, ami számon tartja, hogy hány iteráció óta nincs jobb állapot. Ellenkező esetben növeljük a számlálót. Az eljárás pár iterációja a 2.6. ábrán figyelhető meg. Amennyiben elfogadunk egy állapot, akkor elmentjük azt, csökkentjük a hőmérsékletet, és lenullázzuk a számlálót, ami számon tartja, hogy hány iteráció óta nincs jobb állapot. Ellenkező esetben növeljük a számlálót. Ezt az eljárást addig folytatjuk míg

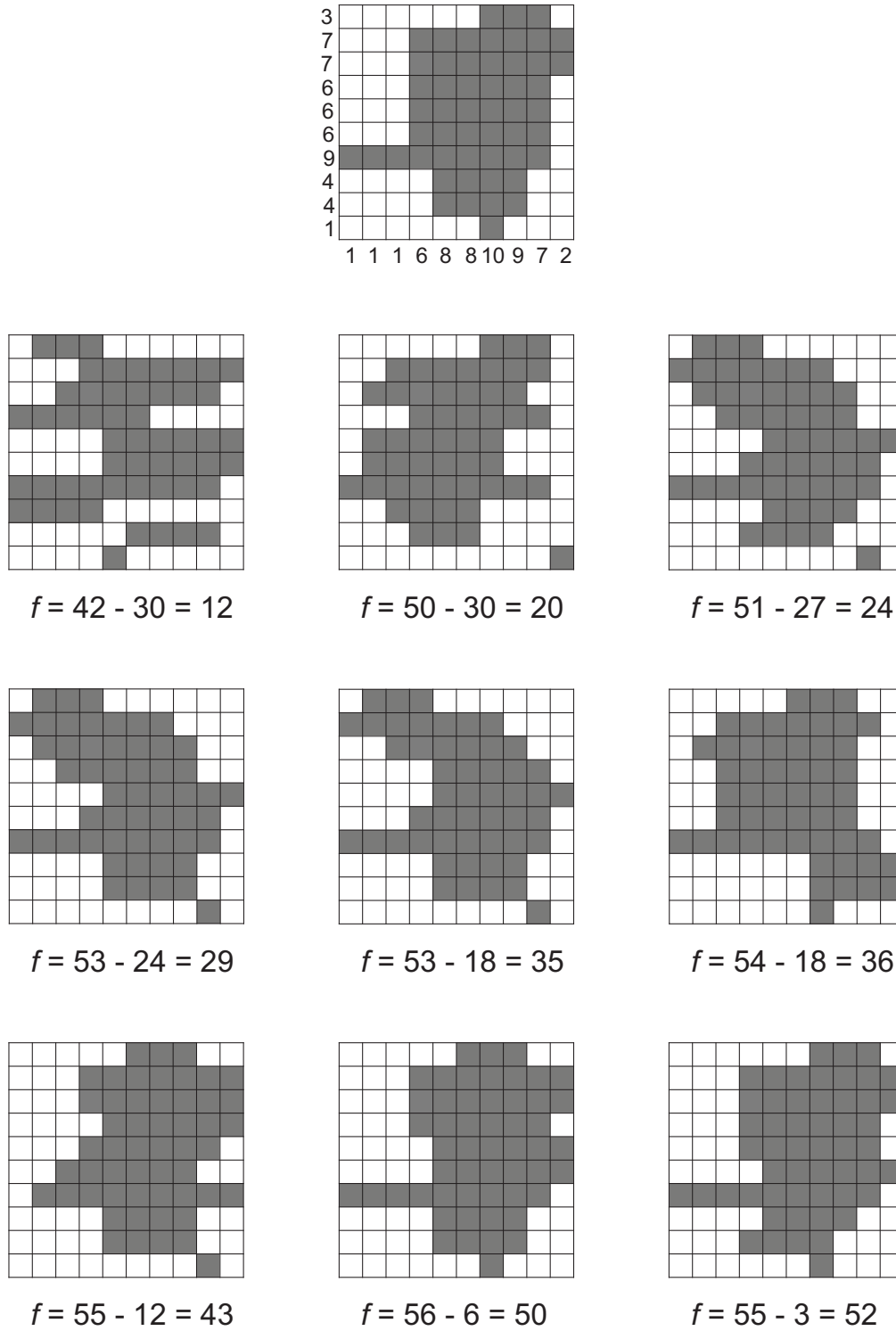
a hőmérséklet el nem éri az általunk kívánt értéket, vagy az előre megadott értéktől több iteráció óta nem találtunk jobb megoldást (3. algoritmus).

Algoritmus 3: H-TILES_SA

Input: $R \in N^m$ és $S \in N^n$ két vektor; Temp = 50; Nbr = 0;

Output: elhelyezett elemek, melyek a rekonstruált bináris eredmény képet reprezentálják.

```
1 Init();
2 while (Temp < 0.005 és Nbr < 500) do
3   ElemMozgat();
4   if ((f(elozo)-f(aktualis) < 0) vagy ( $p < \exp(\frac{-|d|}{Temp})$ )) then
5     elozo=aktualis;
6     Temp=Temp · 0.9999;
7     Nbr=0;
8   else
9     | Nbr++;
10  end
11 end
12 end
```



2.6. Ábra. A H-TILES_SA algoritmus pár lépése. Az első kép az eredeti képet ábrázolja. Alatta a rekonstrukció előrehaladása során előfordult pár állapot, ahol az ábrák alatt az $f = K - (\gamma \cdot P)$ jószágértékek találhatók.

2.4. REGION_GROWTH: Rekonstrukció 4-szomszédokon való régiónövelés segítségével

A következő algoritmus alapötlete, hogy a legvalószínűbb pixelekhez rendeljük az objektum pontjait, illetve a már lerakott pixelek mentén megpróbáljuk a projekciók alapján növelni a régiót, úgy hogy az *hv*-konvex maradjon és kielégítse az adott vetületeket. A továbbiakban REGION_GROWTH néven fogunk hivatkozni az algoritmusra, melynek pszeudokódját a 4. algoritmus írja le.

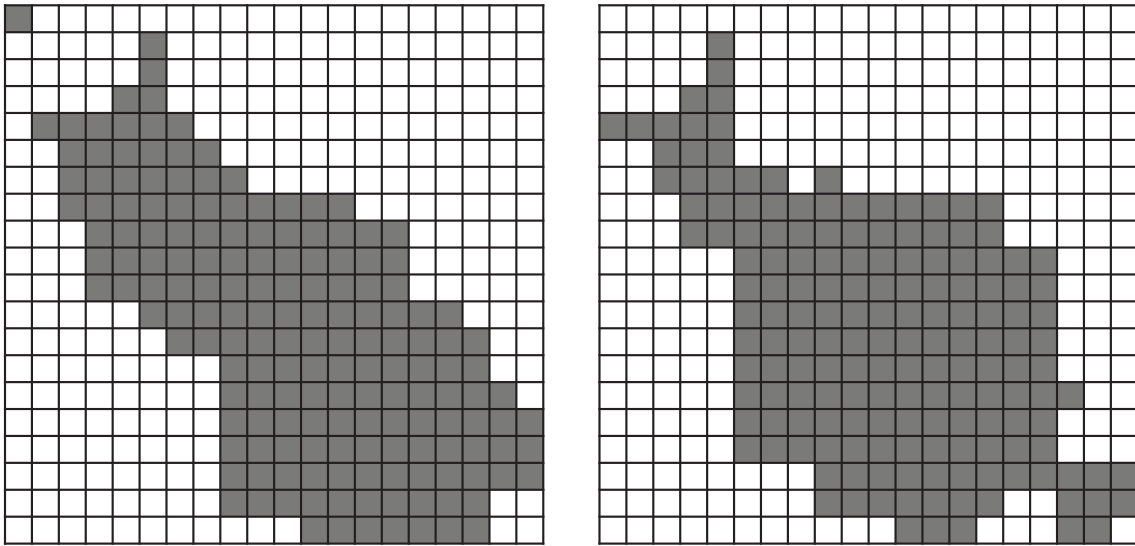
Minden pixelhez két attribútumot rendelünk, az egyik meghatározza, hogy az adott képpont foglalt-e vagy sem, vagyis hogy az adott pixelnél megtalálható-e az objektumnak képpontja. A másik attribútum arra ad mérőszámot, hogy az adott pontban van-e a rekonstruált objektumnak képpontja. A nagyobb szám azt jelöli, hogy nagyobb valószínűséggel (nem matematikai értelemben) van objektumpont az adott pontban. Ha ez az érték 0, akkor az adott pixelben biztos nem lehet objektum pontot elhelyezni. Adott egy w ($1 \leq w, w < \min\{m, n\}$, és páratlan) méretű ablak, amely sorainak és oszlopainak a száma w -vel megegyezik. A mérőszám értéket az adott képpont sorában és oszlopában szereplő vetületek összege, és a képpont 4-szomszédságában levő foglalt képpontok száma határozza meg, amit elosztunk az ablakban található szabad pontok számával (jele: f). Szabad pontok olyan pixelek, melyre $p > 0$ értéke nagyobb mint 0 és nem reprezentálnak objektumpontot. A 4-szomszédságban levő foglalt pixelek számát megszorozzuk egy általunk definiált paraméterezhető konstans értékkel (jele: k), aminek hatására az algoritmus valószínűbbnek fogja tartani a már lerakott elemek 4-szomszédságában levő képpontokat. $A = (a_{ij})_{m \times n}$ egy két dimenziós bináris mátrix, ahol a_{ij} az i . sor j . oszlopában levő elemet reprezentálja, ahol egyes a foglalt, még a nullák azokat a képpontokat jelzik, ahol nem lehet elem (2.6. képlet).

$$p_{ij} = \frac{h_i + v_j + \sum_{j=i-1}^{i+1} a_{ij}k + \sum_{i=j-1}^{j+1} a_{ij}k}{f} \quad (i = 1, \dots, m; j = 1, \dots, n; p \geq 0; k \geq 0; f > 0) \quad (2.6)$$

A rekonstrukció során meghatározzuk az összes nem foglalt képpont értéket, majd kiválasztunk egyet a *TablaKiszamolasa()* függvénnyel a következő módon. algoritmusnál. Az ablakot a vizsgált képen minden pozícióban elhelyezzük és összegezzük az ablakban található képpontok valószínűségeit. Azt a képpontot választjuk, amelyik a legnagyobb értékű ablak középpontjában található meg. Ezt a műveletet a 4. algoritmusnál az *ElemKivalasztasa()* függvény végzi el. Amennyiben az ablak „lelóg” a képről, akkor aktuálisan lelógó pixeleknél az értéke nulla.

Itt is, hasonlóan mint az előző algoritmusnál *aktuális* vetületek alapján dolgozunk, melyek azt jelzik, hogy az adott iterációban még hány objektum pont hiányzik a sorokban és oszlopokban (2.2. képlet).

Az algoritmus futása során ügyelünk, hogy *hv*-konvex legyen a megoldás. Ha egy pozíció foglalt lesz, akkor az aktuális vetülettől nem lehet messzebb objektum képpont az adott sorban és az adott oszlopban. Ezért kinullázzuk azokat az elemeket, melyeknek sorában (vagy oszlopában) mért távolsága nagyobb az adott sorban (vagy oszlopban) levő legközelebbi objektumponttól, mint az aktuális horizontális (vagy vertikális) vetület az adott sorban. Mikor elhelyeztünk egy objektumpontot, az adott sorban és oszlopban levő vetületek eggyel csökkennek. Ezután az aktuális képpont sorát és oszlopát elég újraszámolni. Ezt a lépést a *VetuletekFrissitese()* függvény végzi el.



2.7. Ábra. Eredeti kép (baloldal). Az eredeti kép vetületeiből rekonstruált kép 4-szomszédokon való régió növelés segítségével (jobboldal).

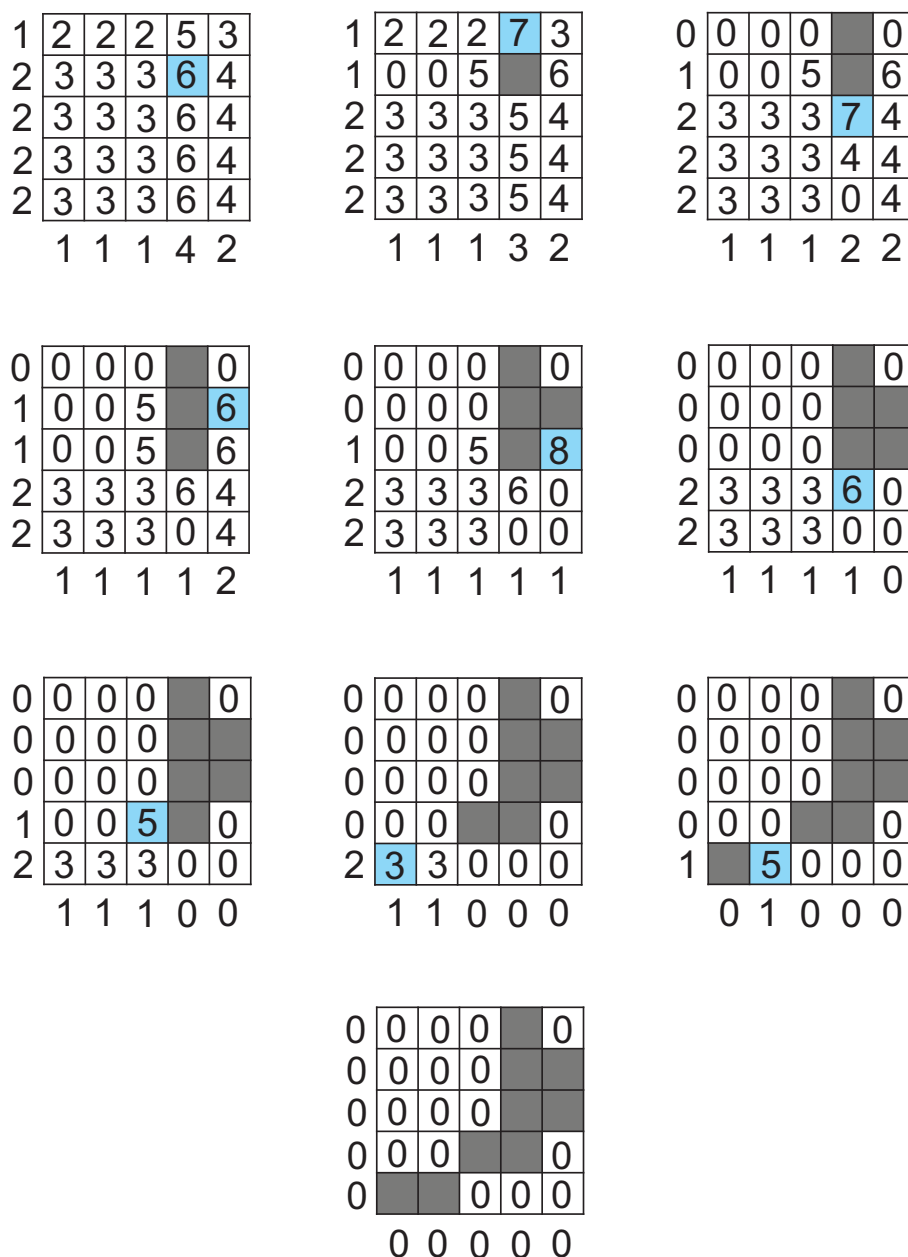
Algoritmus 4: Rekonstrukció 4-szomszédokon való régió növelése

Input: $R \in N^m$ és $S \in N^n$ két vektor;

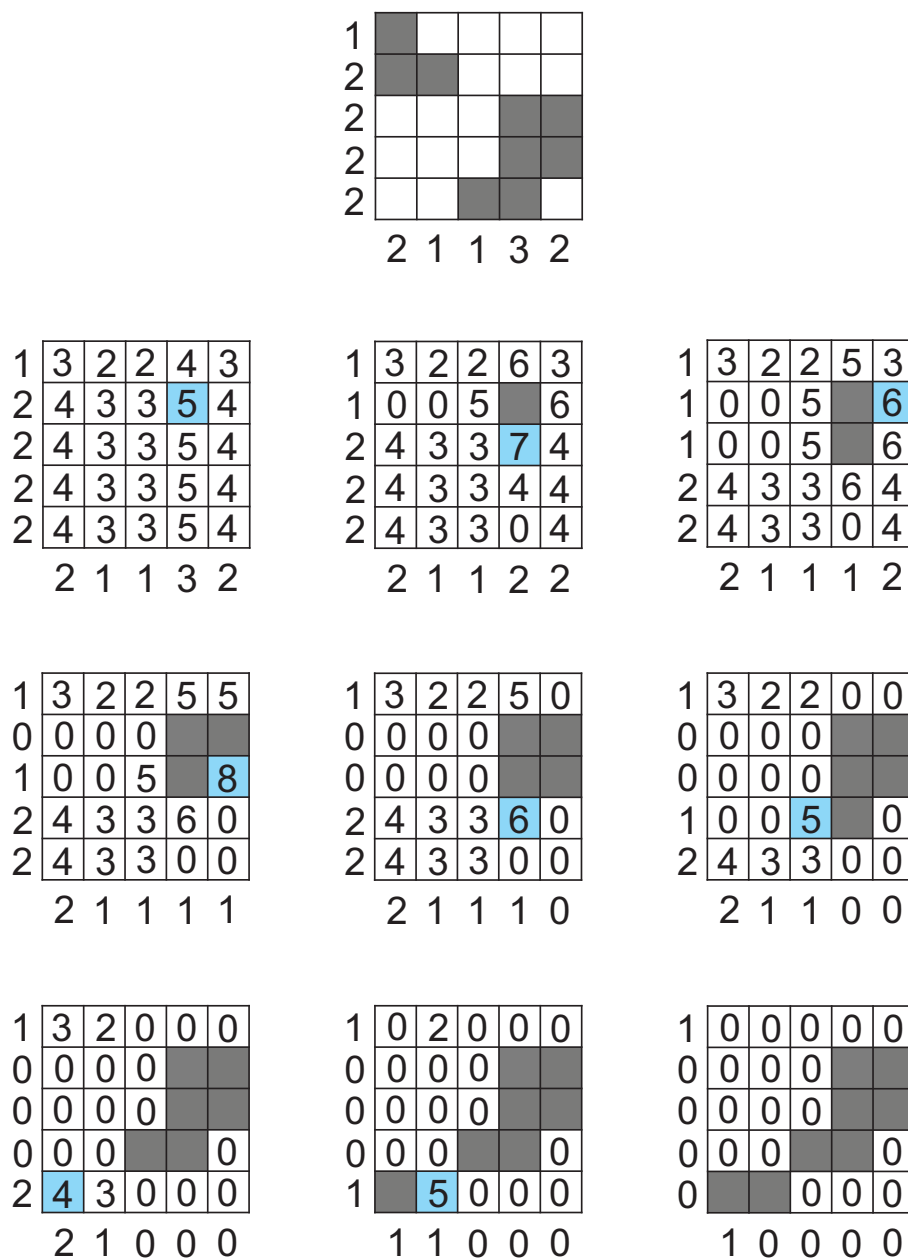
Output: 2 dimenziós mátrix, ahol az egyesek az objektum pontjait reprezentálják, ahol a az eredmény *hv*-konvex. Különben HIBA.

```
1 for (int i=0; i < maxIteracio; i++) do  
2   |   TablaKiszamolasa();  
3   |   ElemValasztasa();  
4   |   VetuletekFrissitese();  
5 end
```

Tökéletes megoldás esetén a rekonstrukció az eredeti képen szereplő pixelek számával megegyező számú iteráció után ad eredményt (*maxIteracio*). A rekonstrukció során



2.8. Ábra. 4-szomszédokon való régió növelés rekonstrukciós lépései, ahol a rekonstruálni kívánt kép vetületei $R = (1, 2, 2, 2, 2)$ és $S = (1, 1, 1, 4, 2)$. A képek melletti számok az aktuális állapotban lévő sorok és oszlopokban levő még le nem rakott objektum pontokat jelzik. A kék mező az adott állapotban kiválasztott elemet, míg a fekete színnel jelölt elemek a lerakott elemeket ábrázolják. A táblázatban található értékek a valószínűségi értékeket tartalmazzák. A fenti példán, amennyiben egy pixel rendelkezik olyan 4-szomszéddal, mely objektum képpont, akkor +3-at adunk hozzá a valószínűségéhez hozzá.



2.9. Ábra. 4-szomszédokon való régió növelés rekonstrukciós lépései, ahol a rekonstruálni kívánt kép vetületei $R = (1, 2, 2, 2, 2)$ és $S = (2, 1, 1, 3, 2)$ (legelső kép). A program paramétereit és a jelöléseket ugyanazok, mint az előző esetben. Ebben az esetben nem született tökéletes megoldás, mivel nem lehet elhelyezni az utolsó pixelt

a megoldás akkor tökéletes, ha az aktuális vetített vektorok mindegyik eleme nulla (2.8. ábra). Viszont az algoritmus nem garantálja a *hv*-konvexitást és projekciós hibák is előfordulhatnak (2.9. ábra). Az utóbbi probléma oka, hogy az algoritmus rossz irányban tölti fel a régiót miközben ügyel a *hv*-konvexitásra. Így olyan pontokat zár (nulláz) ki amelyek a projekció szempontjából jók lennének, viszont megszüntetnék az aktuális állapot *hv*-konvexitását (2.7. ábra).

3. fejezet

Eredmények

Az algoritmusok teljesítményének vizsgálatát több szempont alapján végeztük. A teszt-környezet az alábbi paraméterekkel rendelkezett: Processzor Intel[®] Core[™] i7-2600 CPU @ 3.40 GHz $\times$ 8; Memória 7,8 GiB; operációs rendszer: Ubuntu 13.04 (64 bit). Az algoritmusokat C++-ban implementáltuk. A vizsgálat 2000 $h\nu$ -konvex képen folyt, melyeknek méretei 10×10 -től 50×50 -ig terjedtek 10 pixel lépésközzel. A képek egyenletes eloszlásból származtak és különböző számú komponenseket tartalmaztak (1-től 4-ig) [14, 11].

3.1. h -konvex diszjunkt halmazokon alapuló rekonstrukciós eljárás vizsgálata

Az algoritmus sebességét tekintve fontos, hogy milyen sorrendben választja ki a diszjunkt elemeket. Megfelelő választási sorrend segítségével gyorsan kiesnek azok az esetek, melyek nem vezetnek megoldáshoz, így kevesebb számításra kerülhet sor. Az eredeti algoritmus szerint az elemek elhelyezése a következő. Először válasszuk a legnagyobbat, melynek a legkisebb a sor indexe. A következő elemek választásánál csak olyan elemet választhatunk, mely szomszédos egy elhelyezett elemmel. Továbbá igyekszünk az elhelyezhető elemek közül a nagyobb méretűt választani. E módszer alapján a rekonstrukció futásideje nagyobb képek esetén elfogadhatatlanul nagy (3.1. táblázat). A túl magas futásidő miatt, csak 10×10 -től 30×30 -as méretű képeken végeztük el a méréseket. A tesztek során megvizsgáltuk, hogy az általunk javasolt elemválasztási sorrend tud-e jobb futásidőket biztosítani (3.2. táblázat). Az eredmények alapján látható, hogy mindkét megvalósítás esetén hasonló arányban növekszik a futásidő. A képek méretének növelése során a futásidők átlaga és szórása hirtelen megugrik. A komponensek számának a növelése esetén ugyancsak növekszik a futásidő (bár nem olyan drasztikusan). Nagyjából az összes esetben alacsonyabb a futásidő az eredeti algoritmus rekonstrukciója során. Összegezve

az eredmények tehát azt támasztják alá, hogy sajnos nem javít az algoritmus futásidején a módosított elemválasztási sorrend az eredeti megvalósításhoz képest.

Futási idők - H-TILES eredeti algoritmus						
	10 × 10		20 × 20		30 × 30	
	átlag	szórás	átlag	szórás	átlag	szórás
1	2.91	4.74	7563.28	45312.55	89743.08	106166.14
2	6.15	10.08	34734.60	243680.42	118723.40	115530.27
3	7.62	15.37	63316.84	415149.10	133040.78	135113.34
4	12.88	30.27	49153.88	199689.14	165720.22	164023.71

3.1. táblázat. Eredeti H-TILES rekonstrukciós algoritmus futásidejei milliszekundumban megadva. A sorok a komponens számokat ábrázolják, az oszlopok a tesztképek méreteit.

Futási idők - H-TILES módosított algoritmus						
	10 × 10		20 × 20		30 × 30	
	átlag	szórás	átlag	szórás	átlag	szórás
1	2.65	3.69	45485.06	102794.91	93150.44	111408.39
2	6.89	11.24	51892.18	114399.06	130943.86	146571.14
3	7.61	15.96	56322.17	118804.09	195137.40	156646.51
4	12.62	31.34	53598.49	106997.67	210230.72	172152.83

3.2. táblázat. Módosított H-TILES rekonstrukciós algoritmus futásidejei milliszekundumban megadva. A sorok a komponens számokat ábrázolják, az oszlopok a tesztképek méreteit.

3.2. h -konvex diszjunkt komponenseken alapuló szimulált hűtési rekonstrukció vizsgálata

A futásidőket vizsgálva (3.4. táblázat) megállapítható, hogy a H-TILES algoritmushoz képest sokkal jobb az eredmények. A rekonstrukció idejét nagymértékben befolyásolja a kép mérete és komponenseinek száma. A kép méretének a növelése során több h -konvex diszjunkt elemet kell elhelyezni, ami lassítja a rekonstrukciót. Több komponens esetén általánosságban kevesebb objektumpontot tartalmaz a kép (3.3. táblázat), aminek eredményeképpen kisebbek a csempék és így több pozícióba lehet őket elhelyezni, ami szintén növeli a futásidőt.

Objektumpontok száma										
	10 × 10		20 × 20		30 × 30		40 × 40		50 × 50	
	átlag	szórás	átlag	szórás	átlag	szórás	átlag	szórás	átlag	szórás
1	48.98	10.62	176.57	41.16	370.69	85.00	638.67	183.29	1032.38	260.58
2	31.76	7.75	116.34	31.48	236.26	64.47	392.38	101.98	607.21	198.08
3	26.69	6.43	92.36	26.79	186.48	55.17	306.45	97.79	443.27	118.73
4	21.20	4.52	69.59	16.54	150.10	39.68	250.15	74.18	392.97	108.89

3.3. táblázat. A tesztelt képek objektumpontjainak átlag és szórása. A sorok a komponensek számát ábrázolják, az oszlopok a tesztképek méreteit.

A következő vizsgálati szempont a rekonstrukció pontossága. A 2 vetülettel való rekonstrukció erősen alulhatározott probléma. Ez azt jelenti, hogy egy probléma során több megoldás is lehetséges, melyeknek ugyanazok a vetületeik (1.2. ábra). Ennek oka a más említett kapcsoló komponensek jelenléte. Mivel az algoritmus némely esetekben vetületi hibás megoldásokat ad, ezért érdemes megvizsgálni, hogy a rekonstruált kép mekkora mértékben tér el az eredeti képtől. Legyen $A^o = (a_{ij}^o)$ az eredeti és $A^r = (a_{ij}^r)$ a rekonstruált kép bináris mátrixa. A rekonstrukció pontosságának a mértékét (relative mean error - RME) az alábbi képlet segítségével határozzuk meg, ahol az eredeti és a rekonstruált kép közötti eltérő pixelek számát normalizáljuk az eredeti képen szereplő képpontok számával (3.1. képlet).

$$RME = \frac{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}^o - a_{ij}^r|}{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^o} \quad (i = 1, \dots, m; j = 1, \dots, n). \quad (3.1)$$

A kapott eredmények alapján a rekonstrukció pontossága nem változik szignifikáns mértékben a kép méretének a változtatása esetén. Viszont a komponensek számának a növelése hatására növekszik ez az érték. Ennek vélhetően megint az az oka, hogy kevesebb a objektumpont, így könnyebb olyan megoldást találni, ami jobban eltér az eredeti

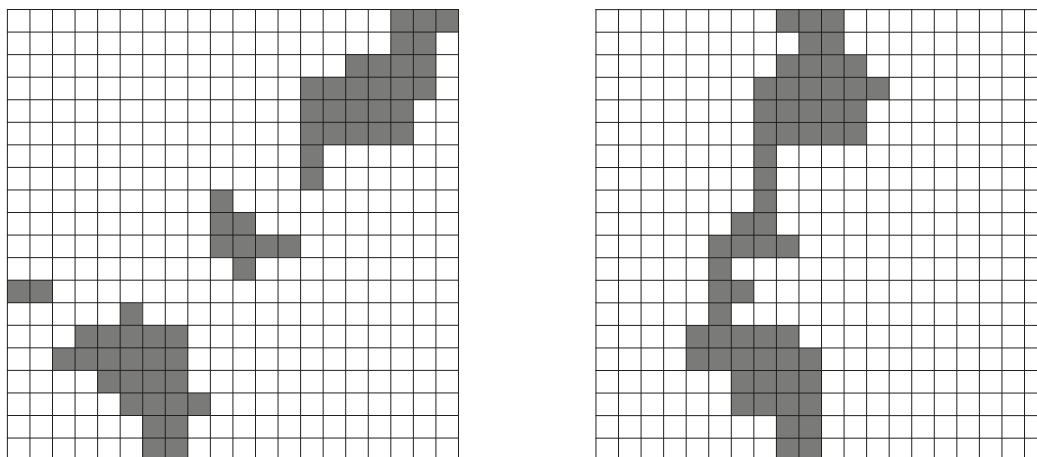
képtől. További probléma, hogy a konvexitás mértékét a vertikális szomszédok száma határozza meg. Ez az érték képméretenként és komponensszámtól függően eltérő lehet, ezért nehéz megadni a pontos arányt a vetületi hiba és a konvexitás között az állapot jóságának becslése során. Így előfordulnak olyan esetek, amikor az algoritmus a szomszédos egységek számát maximalizálva egy oszlopba próbálja rendezni a csempéket, kevésbé ügyelve a projekciós hibára (3.1. ábra).

Futás idő - H-TILES_SA										
	10 × 10		20 × 20		30 × 30		40 × 40		50 × 50	
	átlag	szórás	átlag	szórás	átlag	szórás	átlag	szórás	átlag	szórás
1	1.05	3.33	2.52	7.99	4.50	14.07	7.61	24.03	9.61	30.05
2	1.34	4.13	3.49	10.67	5.89	17.92	10.31	31.31	11.98	36.43
3	1.48	4.54	3.87	11.75	6.53	19.83	11.02	33.44	13.17	39.87
4	1.84	5.65	4.38	13.26	6.89	20.81	11.72	35.48	13.49	41.02

3.4. táblázat. H-TILES_SA algoritmus futásidőjei milliszekundumban megadva. A sorok a komponens számokat ábrázolják, az oszlopok a tesztképek méreteit.

RME - H-TILES_SA										
	10 × 10		20 × 20		30 × 30		40 × 40		50 × 50	
	átlag	szórás	átlag	szórás	átlag	szórás	átlag	szórás	átlag	szórás
1	0.06	0.19	0.06	0.22	0.07	0.22	0.07	0.24	0.06	0.22
2	0.11	0.34	0.11	0.34	0.11	0.36	0.12	0.38	0.13	0.40
3	0.12	0.38	0.12	0.38	0.13	0.41	0.14	0.43	0.14	0.44
4	0.13	0.41	0.15	0.45	0.14	0.45	0.14	0.45	0.15	0.45

3.5. táblázat. H-TILES_SA algoritmus RME értékei. A sorok a komponens számokat ábrázolják, az oszlopok a tesztképek méreteit.



3.1. Ábra. Eredeti kép (bal oldal). Az eredeti kép vetületeiből rekonstruált kép H-TILES_SA eljárás segítségével (jobb oldal).

3.3. 4-szomszédokon való régió növeléses rekonstrukciós eljárás vizsgálata

Első lépésben a 4-szomszédokon való régió növeléses rekonstrukciós eljárásnak a megfelelő paraméterezését vizsgáltuk meg. Az algoritmus paraméterezése során megadható az ablaknak a mérete, illetve a szomszédos foglalt pixelek értéke. Az algoritmus optimális paraméterezésének vizsgálata során arra kerestünk választ, hogy a paraméterezés milyen hatással van az eredményképek konvexitására illetve a projekciós hibára. A konvexitás értékét a szomszédos egyesek száma határozta meg, mely értéket az objektum pontok számának a kétszeresével normalizálunk:

$$konvexitas(A) = \frac{\sum_{i=1}^{m-1} \sum_{j=1}^n a_{ij} a_{(i+1)j} + \sum_{i=1}^m \sum_{j=1}^{n-1} a_{ij} a_{i(j+1)}}{2 \sum_{i=1}^m \sum_{j=1}^n a_{ij}} \quad (3.2)$$

Ez az érték 0 és 1 között lehet, és 1 jelöli a teljes konvexitást. Az eredményeket vizsgálva kiderült, hogy az ablakméret növelése során kezdetben hirtelen romlik a rekonstruált kép konvexitása, majd nem szignifikáns mértékben javul az ablak további növelése során. A szomszédosági érték már kismértékű növekedése egyértelműen javítja az eredményképek konvexitását. Nagyobb értékek esetén egy bizonyos határig növelhető a konvexitás, ezt a határt átlépve viszont már nem tudunk jobb eredményt elérni (3.2. ábra).

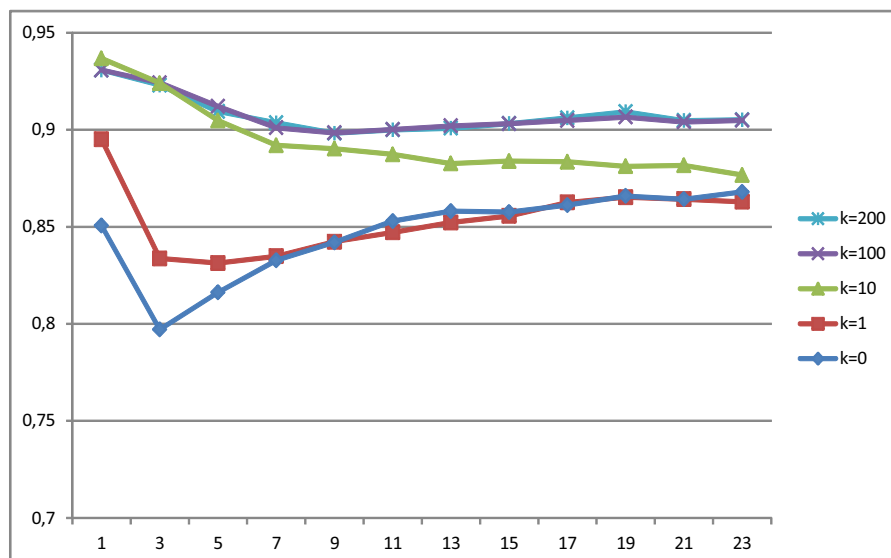
A projekciós hiba meghatározza, hogy mekkora mértékben tér el a rekonstruált kép projekciója (R^r és S^r) az eredeti kép vetületétől (R^o és S^o). Ezt az értéket normalizáljuk a vetületek hosszának az összegével.

$$projekciosHiba = \frac{\sum_{i=1}^m (h_i^o - h_i^r) + \sum_{j=1}^n (v_j^o - v_j^r)}{m + n} \quad (3.3)$$

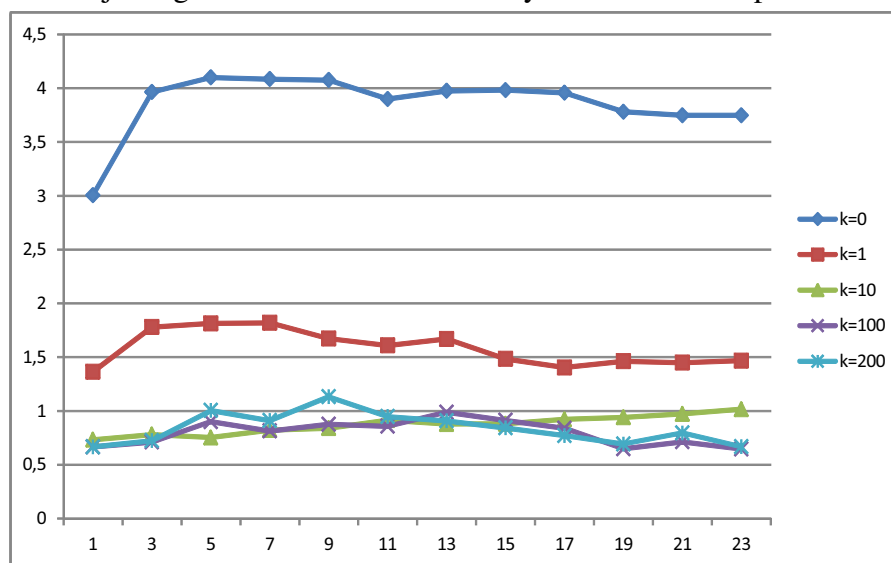
$(v_i^o \in R^o; h_i^o \in S^o; v_i^r \in R^r; h_i^r \in S^r; i = 1, \dots, m; j = 1, \dots, n).$

A képletben szereplő v_j^o és h_i^o az eredeti kép vetületének j -edik oszlopát és i -edik sorát, továbbá v_j^r és h_i^r a rekonstruált kép vetületének j -edik oszlopát és i -edik sorát ábrázolják.

A kapott eredmények alapján megállapítható, hogy a projekciós hiba és a 4-szomszédok értéke között fordított arányosság van. Itt is látható, hogy a 4-szomszédok már kismértékű növelése jelentősen javítja a projekciós hibákat. Az ablak méretének növelése során a projekciós hiba mértéke hirtelen megugrik, majd kis mértékben csökken. Viszont az ablak méretének a változása csak kis mértékben befolyásolja a hiba mértékét, főleg ha a

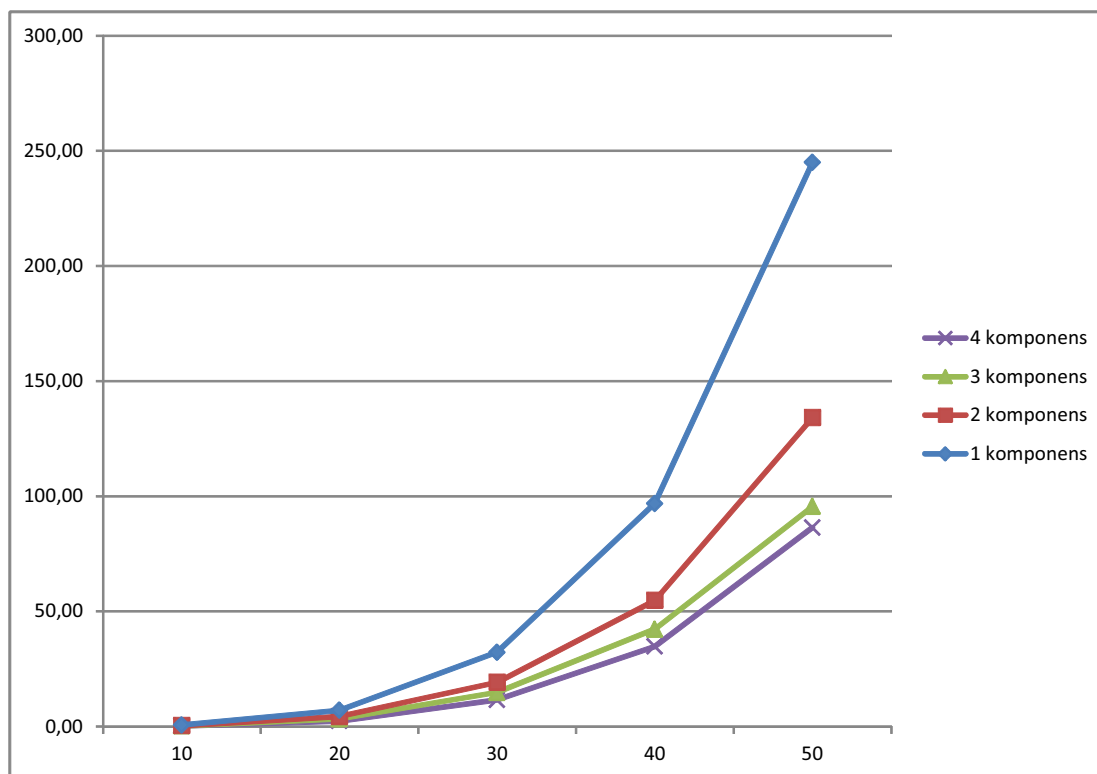


3.2. Ábra. REGION_GROWTH algoritmus eredményképeinek konvexitásának változása az ablak méret (vízszintes tengely) és a 4-szomszédok értéke által ($k = 0$ (◆), $k = 1$ (■), $k = 10$ (▲), $k = 100$ (×), $k = 200$ (*)). A függőleges tengely a konvexitás mértékét ábrázolja. A grafikonon látható eredmények 50×50 -es képeken készültek.



3.3. Ábra. REGION_GROWTH algoritmus projekciós hibájának változása az ablak méret (vízszintes tengely) és a 4-szomszédok értéke által ($k = 0$ (◆), $k = 1$ (■), $k = 10$ (▲), $k = 100$ (×), $k = 200$ (*)). A függőleges tengely a projekciós hibát ábrázolja. A grafikonon látható eredmények 50×50 -es képeken készültek.

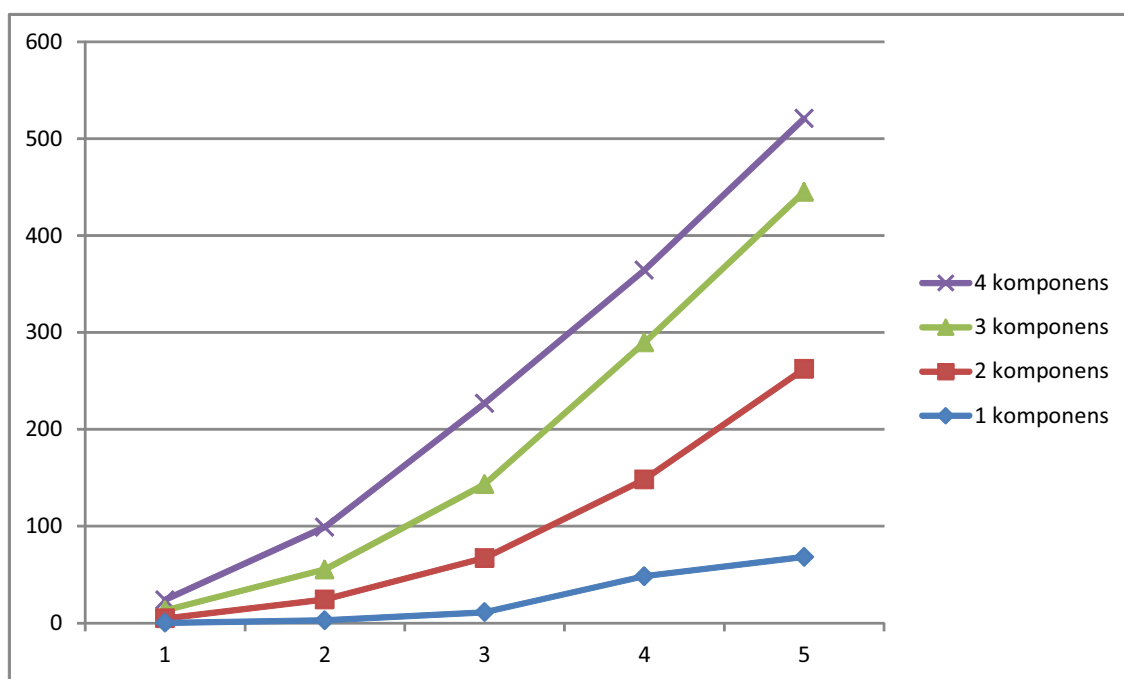
szomszédok értéke alacsony (3.3. ábra). Az eredmények alapján az algoritmus paraméterezése azokban az esetekben optimális, ha az ablak mérete kicsi és az szomszédok értéke kiugróan nagy. Ezért a továbbiakban az ablak mérete 1, és a 4-szomszédok értéke a kép méretének négyszerese lesz.



3.4. Ábra. REGION_GROWTH algoritmus futásidejének változása a képméret (vízszintes tengely), és a komponensek számától (1 komponens (◆), 2 komponens (■), 3 komponens (▲), 4 komponens (×)) függően. A függőleges tengely a futásidőt ábrázolja milliszekundumban.

Általánosságban, ha egy kép több komponenst tartalmaz, akkor kevesebb objektumot reprezentáló képpontot foglal magába (3.3. táblázat). Az 4-szomszédokon való régió növeléses rekonstrukciós algoritmus legfeljebb az objektumpontok számával megegyező számú iterációt hajthat végre. Ezért a futásidők átlaga és szórása is csökken fog a komponensek számának a növelésével. A 3.4. és a 3.5 ábra esetén megállapítható, hogy a REGION_GROWTH algoritmus esetén a legnagyobb futásidő különbség az egy és két komponenses képek között van. Ennek oka, hogy az egy és két komponenses képek között nagyobb eltérés van az objektumpontok számában, mint nagyobb komponens szám esetén. Ezért itt hirtelen csökken az objektumpontok száma, ami nagyobb mértékben javítja a futásidőt.

Összehasonlítva a 4-szomszédokon való régió növelést és a diszjunkt komponenseken alapuló algoritmust megállapítható, hogy az első algoritmus több időt igényel egy kom-



3.5. Ábra. Diszjunkt komponensek elhelyezkedésén alapuló rekonstrukciós algoritmus futásidejének változása a képméret (vízszintes tengely), és a komponensek számától (1 komponens (♦), 2 komponens (■), 3 komponens (▲), 4 komponens (×)) függően. A függőleges tengely a futásidőt ábrázolja milliszekundumban.

ponens esetén. Ennek oka, hogy az utóbbi algoritmus esetén a rekonstrukció nagy részét a komponenseket befoglaló téglalapok keresése teszi ki. Amennyiben csak egy komponens tartalmaz a kép, akkor gyorsan kiderül, hogy az egész kép maga egy komponenset befoglaló téglalap, amit az eljárás gyorsan tud rekonstruálni. A REGION_GROWTH algoritmus több esetben sokkal jobban teljesít, és míg a másik algoritmusnak a futásideje nő a komponensek számával, addig a 4-szomszédokon való régió növelés esetén a futásidő csökken. A futásidők szórásánál hasonlóak az eredmények. A diszjunkt komponenseken alapuló algoritmus esetén a szórás növekszik, míg a 4-szomszédokon való régió növelés algoritmusnál csökken ez az érték, és nagyságrendekkel alacsonyabb is. Tehát könnyebben megjósolható a rekonstrukció futásidejének a nagysága.

Az RME-t vizsgálva (3.8. táblázat) megállítható, hogy az átlagos hiba mértéke és szórása a komponensek számának a növekedésével egyenes arányban nő. Ennek oka, hogy általában több komponenses képek esetén a vetületek értékei kisebbek, és ezek az alacsony értékek csak növelik a bizonytalanságot a rekonstrukció során, mivel az algoritmus alapja, hogy a vetületi értékekből egy valószínűségi értéket próbál előállítani képpontonként. Néha már három komponens esetén eléri a maximumot az átlag és a szórás, így négy komponens esetén már kisebbek az értékek. Kisebb mértékben a kép nagyságának növekedésével is növekszik az átlagos hibák mértéke. A kép méretének növelése esetén

Futás idők - REGION_GROWTH algoritmus										
	10 × 10		20 × 20		30 × 30		40 × 40		50 × 50	
	átlag	szórás	átlag	szórás	átlag	szórás	átlag	szórás	átlag	szórás
1	0.65	0.55	7.04	2.07	32.25	8.97	96.81	34.30	244.96	74.03
2	0.39	0.49	4.36	1.55	19.20	6.67	54.72	17.76	134.10	55.92
3	0.30	0.46	3.36	1.37	14.80	5.67	42.22	16.73	95.53	30.50
4	0.26	0.44	2.40	0.84	11.58	3.89	34.74	12.53	86.39	28.53

3.6. táblázat. REGION_GROWTH algoritmus futásidejei milliszekundumban megadva. A sorok a komponens számokat ábrázolják, az oszlopok a tesztképek méreteit.

Futás idők - Diszjunkt komponensek elhelyezkedésén alapuló rekonstrukciós algoritmus										
	10 × 10		20 × 20		30 × 30		40 × 40		50 × 50	
	átlag	szórás	átlag	szórás	átlag	szórás	átlag	szórás	átlag	szórás
1	0.20	1.41	2.80	7.92	11.20	25.63	48.20	88.24	68.30	133.41
2	4.80	6.74	24.30	22.03	67.01	43.66	148.20	116.67	262.20	199.05
3	12.90	10.28	55.20	42.13	143.16	76.31	289.50	173.22	444.80	204.99
4	23.70	10.98	98.90	43.41	226.67	111.94	364.19	188.66	520.54	197.49

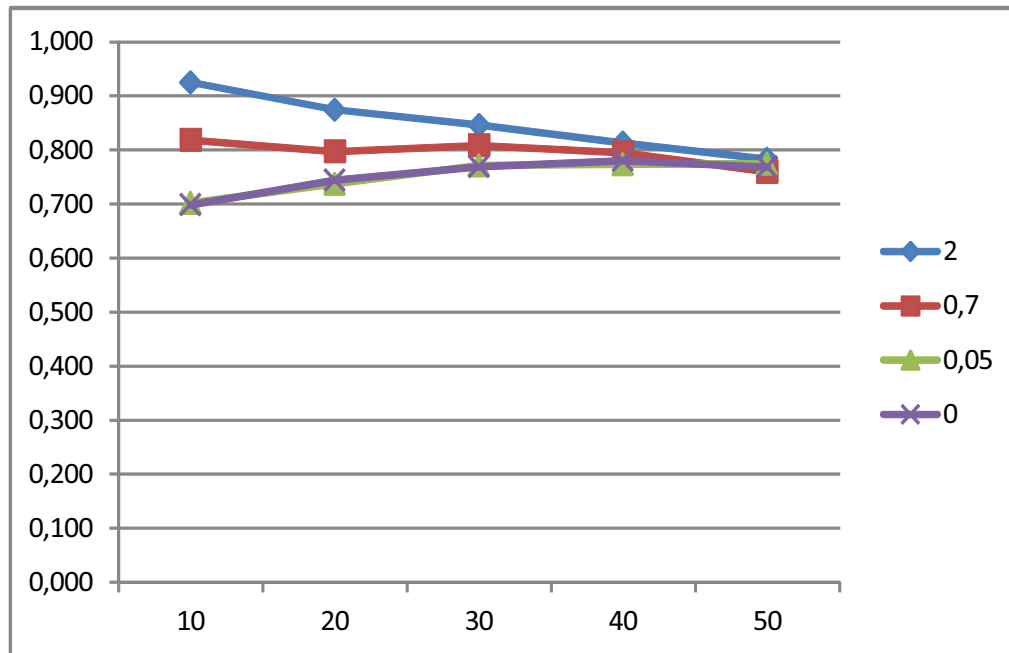
3.7. táblázat. Diszjunkt komponensek elhelyezkedésén alapuló rekonstrukciós algoritmus futásidejei milliszekundumban megadva. A sorok a komponens számokat ábrázolják, az oszlopok a tesztképek méreteit.

nem egyértelmű az RME érték változása. Egy komponensnél az RME értéke növekszik a képmérettel együtt, viszont több komponens esetén nincs szignifikáns változás. A szórás szempontjából sincs számottevő változás a képméretének a növelése során.

A REGION_GROWTH algoritmussal való rekonstruálás során előfordulnak projekciós hibák, illetve a *hv*-konvexitásra sincs garancia. Ezért megvizsgáltuk, hogy különböző mértékben Gauss zajjal terhelt képek esetén milyen RME értékű képeket ad eredményül az algoritmus. A kísérlet során enyhe ($\sigma = 0,05$), közepes ($\sigma = 0,7$) és nagy ($\sigma = 2$) zajjal hajtottam végre teszteket. Az eredmények alapján (3.6. ábra) kiderült, hogy kisebb méretű képek esetén nagyobb az átlagos hiba, ennek oka, hogy kis méretnél kevesebb képpont van, így egy-egy pixel hiba nagyobb értékű. Viszont a kép méretének a növelésével a különböző σ értékű képek átlagos hibái egy irányba konvergálnak.

RME - REGION_GROWTH algoritmus										
	10 × 10		20 × 20		30 × 30		40 × 40		50 × 50	
	átlag	szórás	átlag	szórás	átlag	szórás	átlag	szórás	átlag	szórás
1	0.35	0.34	0.46	0.35	0.52	0.32	0.60	0.45	0.57	0.38
2	0.67	0.60	0.70	0.62	0.70	0.57	0.75	0.56	0.67	0.54
3	0.78	0.66	0.77	0.67	0.87	0.71	0.91	0.66	0.87	0.57
4	0.98	0.64	1.04	0.71	0.98	0.63	0.86	0.55	0.97	0.64

3.8. táblázat. REGION_GROWTH algoritmus RME értékei. Sorok a komponens számokat ábrázolják, az oszlopok a tesztképek méreteit.



3.6. Ábra. REGION_GROWTH algoritmus esetén a rekonstrukció minőségének változása a képméret (vízszintes tengely), és a Gauss zaj függvényében (zaj nélkül (×), $\sigma = 0,05$ (▲), $\sigma = 0,7$ (■), $\sigma = 2$ (♦)). A függőleges tengely az RME értékét ábrázolja.

4. fejezet

Összegzés

A dolgozat célja két új rekonstrukciós eljárás ismertetése és elemzése volt. A kísérletek eredményeit látva megállapítható, hogy a 4-szomszédságon alapuló régió-növeléses rekonstrukciós algoritmus megfelelő, amennyiben alacsony futásidejű eljárásra van szükségünk, és megengedhetünk bizonyos mértékű hibát az eredményképekben. Ez főleg azokban az esetekben fordulhat elő, ha egy optimalizáción alapuló rekonstrukciós algoritmusnak szeretnénk kezdeti megoldást adni, amit tovább lehet javítani. Mindezek mellett, mivel a futásidő csökken a komponensek számának a növekedésével, ezért nagyobb számú komponenseket tartalmazó képek esetén is érdemes lehet használni. A h -konvex diszjunkt halmazokon alapuló rekonstrukciós eljárás sajnos nem hozta a várt eredményeket, mivel a képek méretének növekedésével erősen nő a rekonstrukció ideje. Így csak kisebb képeknél (10×10) lenne érdemes használni. Nagyobb méretű képek esetén könnyen előfordulhat, hogy a rekonstrukció futásideje elfogadhatatlanul nagy lesz. Ez az eljárás viszont teljes mértékben hv -konvex képeket ad eredményül, illetve képes a rekonstrukciós probléma összes lehetséges megoldását is megtalálni.

4.1. További tervek

A leírt algoritmusokat tovább lehet fejleszteni több szempont alapján. A 4-szomszédságon alapuló régió növeléses rekonstrukciós algoritmus esetén érdemes lenne megvizsgálni azokat az eseteket, amikor projekciós hiba merül fel, és plusz feltételek segítségével megelőzni a hiba kialakulását. Célszerű lenne továbbá megvizsgálni azokat az eseteket, amikor az algoritmus előismeretek segítségével ismer pár olyan képpontot, melyek biztosan a megoldáshoz fognak tartozni. Így ezekben esetekben nagy valószínűséggel kisebb projekciós hiba alakulna ki. A valószínűségi értékeket is hatékonyabban meglehetne adni, például olyan súlyozással, ahol nem csak a 4-szomszédos képpontokat vizsgáljuk, hanem távolabbi szomszédok is szerepet játszanának ezen érték meghatározásában. Mindezek

mellett, tudjuk, hogy az algoritmus gyors eredményeket ad, melyek bizonyos mértékű hibát tartalmazhatnak. Ez felhasználható a rekonstrukció során kapott komponenseket befoglaló téglalapok megállapításához.

h-konvex diszjunkt halmazokon alapuló rekonstrukciós eljárás esetén hasznos az algoritmus futása alatt minél korábban kizárni azokat az eseteket, melyek biztosan nem vezetnek megoldáshoz. Ezáltal kevesebb lenne a rekonstrukció során a találgatások száma, így csökkenteni lehetne a futásidőket. Valószínű, akkor is csökkenne a futásidő, ha az eljárás rendelkezne a komponensek elhelyezkedésével kapcsolatban a priori információval.

A *h*-konvex diszjunkt halmazokon alapuló szimulált hűtést alkalmazó algoritmus esetén a konvexitás becslésén lehet még javítani. A [15] cikk alapján a konvexitást meg lehetne becsülni az úgynevezett *irány szerinti* konvexitással, ahol az adott irányban előforduló egyesek és nullák sorozata alapján határozzuk meg a konvexitás mértékét. Viszont ennek a számítása költséges művelet, ezért megfelelő stratégiát is igényel a használata. Mindezen vizsgálatok azonban túlmutatnak a dolgozat témakörén.

Nyilatkozat

Alulírott szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Tanszékcsoport Tanszékén készítettem, diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Informatikai Tanszékcsoport könyvtárában, a helyben olvasható könyvek között helyezik el.

Szeged, 2014. október 6.

.....

aláírás

Alulírott szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Tanszékcsoport Tanszékén készítettem, diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a TVSZ 4. sz. mellékletében leírtak szerint kezelik.

Szeged, 2014. október 6.

.....

aláírás

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani mindenekelőtt témavezetőmnek Balázs Péternek, a lelkes és kitartó támogatásáért.

Jelen kutatási munkát mind a témavezető mint a hallgató esetében támogatta: a Közigazgatási és Igazságügyi Hivatal által a TÁMOP 4.2.4.A/2-11-1-2012-0001 Nemzeti Kiválóság Program - Hazai hallgatói, illetve kutatói személyi támogatást biztosító rendszer kidolgozása és működtetése konvergencia program című kiemelt projekt keretében meghirdetett kutatói ösztöndíj program

Irodalomjegyzék

- [1] G.T. Herman, A. Kuba (Eds.), *Discrete Tomography: Foundations, Algorithms and Applications*, Birkhäuser, Boston, 1999.
- [2] G.T. Herman, A. Kuba (Eds.), *Advances in Discrete Tomography and Its Applications*, Birkhäuser, Boston, 2007.
- [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] S. Brunetti, A. Del Lungo, F. Del Ristoro, A. Kuba, M. Nivat, Reconstruction of 4- and 8-connected convex discrete sets from row and column projections, *Lin. Algebra Appl.* **339** (2001) 37–57.
- [5] M. Chrobak, C. Dürr, Reconstructing hv-convex polyominoes from orthogonal projections, *Inform. Process. Lett.* **69** (1999) 283–289.
- [6] G.J. Woeginger, The reconstruction of polyominoes from their orthogonal projections, *Inform. Process. Lett.* **77** (2001) 225–229.
- [7] Z. Ozsvár, P. Balázs, An empirical study of reconstructing hv-convex binary matrices from horizontal and vertical projections. *Acta Cybern.* **21(1)** (2013) 149-163.
- [8] A. Kuba, Reconstruction of two-directionally connected binary patterns from their two orthogonal projections, *Comp. Vision, Graphics, and Image Proc.* **27** (1984) 249-265.
- [9] F. Jarray, G. Tlig, A simulated annealing for reconstructing hv-convex binary matrices, *Discrete Applied Mathematics* **36** (2010) 447-454
- [10] H. J. Ryser, Combinatorial properties of matrices of zeros and ones, *Canad. J. Math.* **9** (1957) 371-377

- [11] P. Balázs, A framework for generating some discrete sets with disjoint components by using uniform distributions, *Theoret. Comput. Sci.* **406** (2008) 15–23.
- [12] P. Balázs, Discrete tomographic reconstruction of binary images with disjoint components using shape information, *International Journal of Shape Modeling* **14:2** (2008) 189–207.
- [13] M. Póth, Branch and bound method for discrete tomography reconstruction of *hv*-convex binary matrices, *9th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics* (2008) 293–301.
- [14] P. Balázs, A benchmark set for the reconstruction of *hv*-convex discrete sets, *Discrete Appl. Math.* **157** (2009) 3447–3456.
- [15] T.S. Tasi, L.G. Nyúl, P. Balázs, Directional convexity measure for binary tomography. *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, Lecture Notes in Computer Science* **8259** (2013) 9-16.