Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Vékonycsiszolati képek szegmentálása Monte Carlo alapú módszerrel

TDK DOLGOZAT

Készítette
Budai Ádám

Konzulens
Dr. Csorba Kristóf

2016. október 27.

Tartalomjegyzék

Bevezető

A régészeti kutatások során előkerülő márványokról gyakran fontos megállapítani azok származási helyét. Szintén felmerül a restaurációs munkálatok során, hogy a hiányzó építőelemeket hogyan pótolják megfelelő minőségű homokkövekkel, amik képesek az időjárás viszontagságainak ellenállni. Az ilyen jellegű problémák megoldására a hagyományos módszer az, hogy vékonycsiszolati képeket készítenek a márvány, vagy a homokkő valamely mintájának keresztmetszetéről. A képen megkeresik és körbe rajzolják a szemcséket, majd meghatározzák annak méreteloszlását és szomszédossági jellemzőit. Mindezt persze kézzel és szabad szemmel. A dolgozat a GrainAutLine projekt keretein belül született, amelynek a célja, hogy a fent vázolt eljárást automatizálja. A projekt több automatizálási eszközt is integrál, ezek közül az egyik a dolgozat tárgyát képező RJMCMC algoritmus, amely speciális kristályok, ikerkristályok, azonosítását teszi lehetővé. Az ikerkristályok megfelelő fizikai körülmények során keletkezett márványszemcsék, amelyek több kisebb kristály összenövésével jöttek létre. Az algoritmus célja, hogy megtalálja a valódi szemcsehatárokat , még az ikerkristályok jelenléte esetén is, amelyek egyébként a hagyományos, intenzitás gradienseken alapuló algoritmusok számára kezelhetetlenek. Ennek oka, hogy az ikerkristályokat alkotó összenőtt szemcsék határai is még látszanak, pedig azok nem részei az ikerkristályt szegélyező határvonalnak. Dolgozatomban bemutatok egy monte carlo alapú eljárást (RJMCMC: reversible-jump markov chain monte carlo), ami meg tud küzdeni a szegmentálási feladattal, ikerkristályok jelenléte esetén is. A módszernek két alap pillére van. Az első, hogy definiálni kellett egy alkalmas energiafüggvényt, ami jellemezni képes az aktuális szemcsehatárok jóságát. Az energiafüggvény minimuma az a hely, ahol az algoritmus által javasolt szemcsehatár megegyezik a valódiakkal. Másodszor, a jól definiált energiafüggvény minimumát az RJMCMC keresi meg oly módon, hogy változtatja a javasolt szemcsehatárokat (konfigurációt) valószínűségi alapon. A változtatási lehetőségek közül annak a nagyobb a valószínűsége, amelyik esetén az energia értéke a legtöbbet csökken. A kutatás során az algoritmust megvalósítottam és teszteltem mesterséges- és valós ábrákon egyaránt. A mesterséges ábrák kifejezetten az energiafüggvény kifejező erejét voltak hivatottak próbára tenni, míg a valós képek, főleg a konvergencia sebesség és a futás idő miatt mutattak érdekes eredményeket. Összefoglalva, megvalósítottam az rjmcmc algoritmus család egy olyan tagját, ami minimalizálja a konfigurációsenergia-függvényt. A konfigurációs energia a szemcsehatárok helyességét írja le. Az energiafüggvényt megkonstruáltam és teszteltem mesterséges és valódi képeken is. A teljes algoritmus valós képeken is meggyőző eredményeket mutatott.

1. fejezet

A feladat ismertetése

Az alábbi három alfejezetben bemutatom a CV4Sensorhub nevű projektet, annak egy alprojektjét, ami márványszemcsék analízisével foglalkozik és a célkitűzést. A cél a márványokról készült vékonycsiszolati képek (1.1. ábra) félautomata szegmentálása. A félautomata azt jelenti jelen kontextusban, hogy a felhasználótól bizonyos beavatkozások szükségesek, viszont ezek lényegesen kevesebb erőfeszítést igényelnek a részükről a teljesen kézi megoldásokhoz képest.

1.1. A CV4Sensorhub projekt

A CV4Sensorhub egy felhasználó felülettel (1.3. ábra) is rendelkező képfeldolgozási keretrendszer. A projekt célja, hogy lehetővé tegye képfeldolgozási algoritmusok futtatását raszteres képen és címkézett poligonokon alapuló adatszerkezettel, mind szerver és kliens oldalon. A kliens oldalon lehetőség van a felhasználótól segítséget kérni (módosításokat tehet, kiegészítő információk kérhetünk tőlük), így az algoritmusoknak nem szükséges teljesen autonóm módon végezni a képfeldolgozást. Ez egyes esetekben előnyt jelenthet a feladat nehézsége miatt. A keretrendszer az alábbi három főbb komponensből áll:

- közös, portábilis, JSON alapú adatreprezentáció képfeldolgozási módszerekhez
- .NET WPF (és később párhuzamosan HTML5) alatt fejlesztett felhasználói felület
- kliens vagy szerver oldalon is futtatható képfeldolgozási műveletek

Maga a keretrendszernek jelenleg 3 alkalmazási iránya van, melyek korábban önálló független projektek voltak, de közös vonásaik miatt, használják a CV4sensorhub keretrendszer szolgáltatásait. Ezen alkalmazási irányok:

- GrainAutLine: márvány vékonycsiszolatok elemzését szolgáló rendszer.
- ChemoTracker: fehérvértestek mozgását követő alkalmazás immunológiai kutatások támogatására.
- Játékok: a cv4s keretrendszer bizonyos játékok elkészítésére is alkalmas. Ezek elkészítése során számos olyan funkcióval bővül a keretrendszer, amit a többi alkalmazás is fel tud majd használni.

A dolgozatban a *márványszemcsék vékonycsiszolati képeinek szegmentálását* segítő speciális algoritmusról lesz szó.

1.2. A GrainAutLine projekt

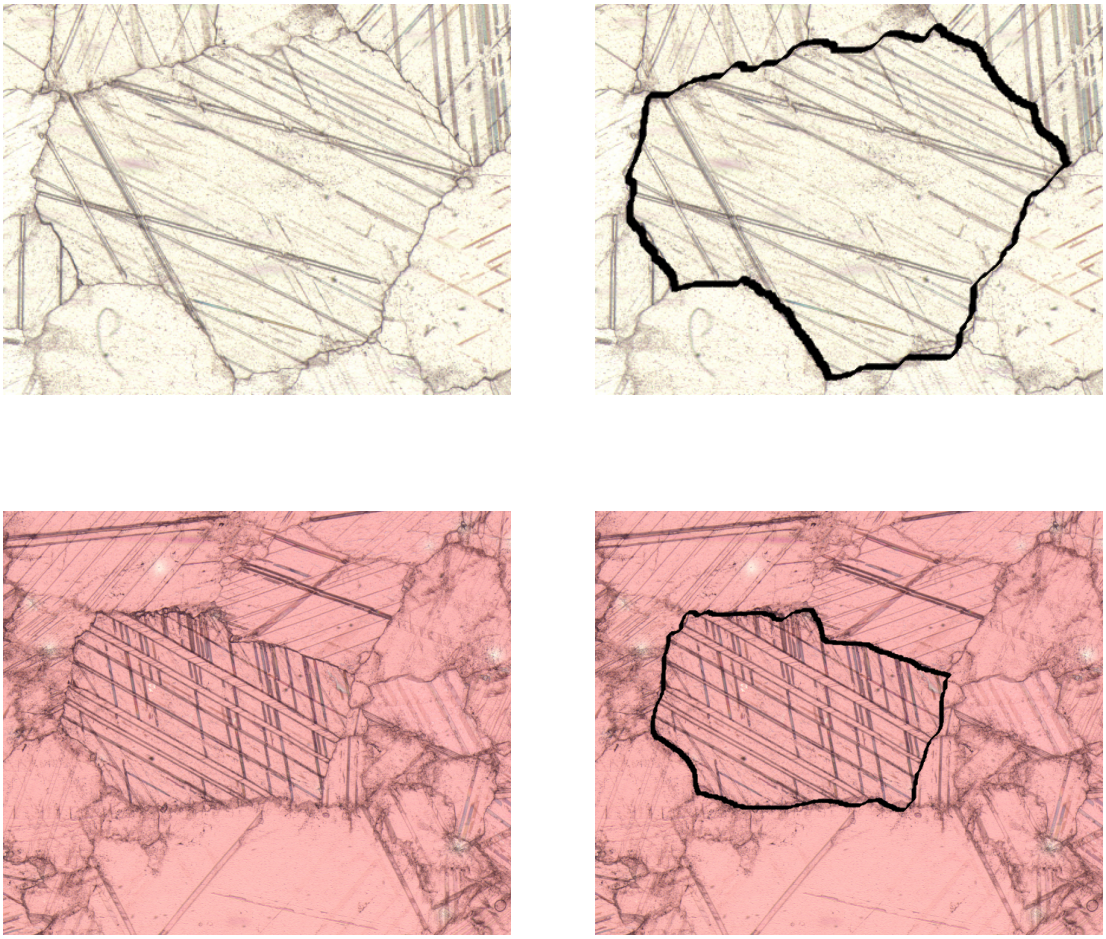
1.1. ábra. *Példa márvány vékonycsiszolati képekre.*

A GrainAutLine projekt célja ([6] és `http://bmeaut.github.io/grainautline/`), hogy megkönnyítse a geológusok, régészek és más, kőzetekkel foglalkozó szakemberek munkáját. A márványok, illetve homokkövek szemcséinek egymáshoz viszonyított helyzete, méreteloszlása, szomszédossági gráfja fontos jellemzők megállapítását teszik lehetővé, mint például a márvány származási helye vagy a homokkövek vízzel szemben tanúsított ellenállását, ami restaurációs munkálatoknál releváns kérdés. A márvány jellemzőinek meghatározásához a kulcs a szemcsehatárok pontos ismerete. Hagyományosan e határokat a szakemberek szemmel keresik meg és egyenként kézzel rajzolják meg a határokat. Egy kép több száz szemcsét is tartalmazhat. Becslések szerint körülbelül 300 szemcse szükséges, hogy statisztikailag megalapozott legyen egy következtetés. A határvonalak megtalálását követően a szemcseméreteket is kézzel határozzák meg. Egy ilyen folyamat rengeteg időt és energiát követel egy szakembertől. A GrainAutLine több olyan megoldást is kínál, amely ezen igyekszik könnyíteni. Ezeket felsorolás jelleggel vázlatosan ismertetem:

- Egyszerű szerkesztő funkciók: lehetséges a határvonalak kézi rajzolása és utólagos szerkesztése. Ez máris kellemesebb, mint ha papírra történne a rajzolás.
- Szegmentáló algoritmusok: A határvonalak meghatározhatók edge detektorokkal, mint a canny edge detektor, adaptive thresholding szegmenter stb. Ezek pontossága esetenként kielégítő, de a hibák a többi szerkesztő funkcióval könnyen javíthatók.
- Statisztikai adatok kinyerése: Ha már elkészült a szegmentálás, akkor lehetőség van a méreteloszlás, szomszédossági gráf meghatározására beépített algoritmusokat használva. Ez is jelentősen csökkenti az elemzéssel töltött időt.

A dolgozatban bemutatandó algoritmus egy Monte Carlo alapú szegmentáló algoritmus, ami speciális márványszemcsékre (ikerkristályokra) lett kifejlesztve. Az *ikerkristályokat* tartalmazó mintákat különösen nehéz szegmentálni.

1.3. Az ikerkristályok



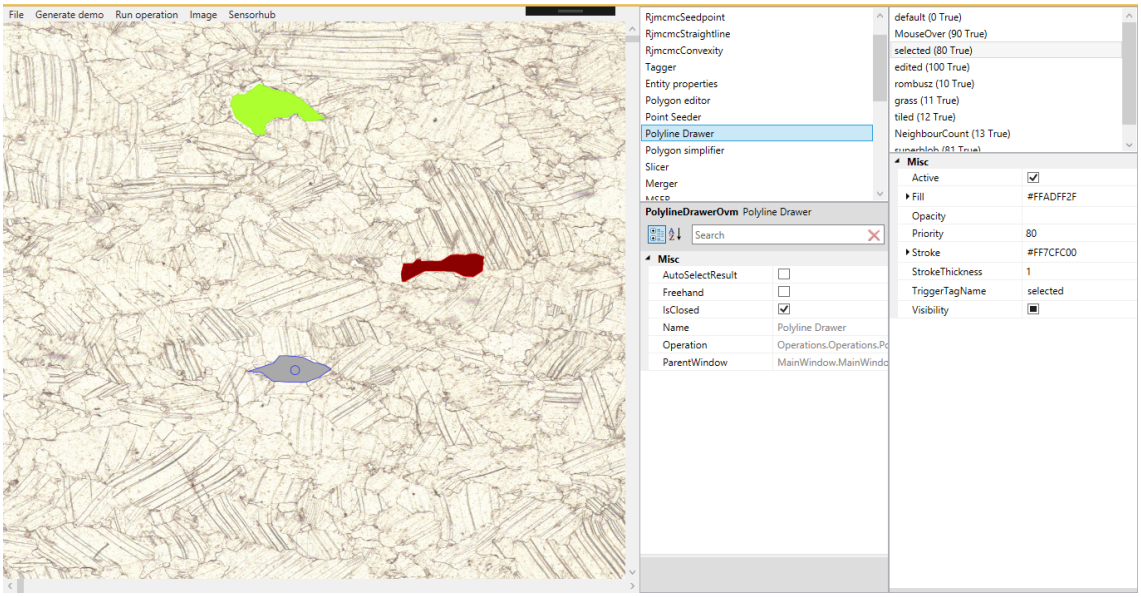
1.2. ábra. *Példák ikerkristályokra. Bal oldalt a tiszta kép látható, jobb oldalt az ikerkristály határa berajzolva.*

Az ikerkristályok alatt olyak márványszemcséket értenek, amelyek valamilyen fizikai körülmények során több kisebb szemcse összenövésével keletkeztek. Ennek köszönhető a (1.2.) ábrán látható sok vonal egy szemcsén belül. Azok az összeolvadás mentén jöttek létre. Azt is észre lehet venni, hogy bár többnyire egyenesek, de nem tökéletesen azok. Sokszor megtörnek, megszűnnek mielőtt elérnék a határt vagy túl sok vonal egymás mellett egy nagyobbnak tűnik. Ezek mind megnehezítik a felismerést.

Azok a szegmentáló algoritmusok, amik a képpontok intenzitás viszonyaiból próbálják meghatározni a szemcsehatárokat, az összeolvadás vonalait is határnak tekintik. Az ikerkristály kiszűrésének alapja, az, hogy ezek a vonalak körülbelül egyenesek. Azonban az előbb felsoroltak miatt (megtörnek, megszűnnek stb.) nem triviális a megkeresésük. Az egymást metsző egyenesek esetén ez különösen igaz. Determinisztikus módszerrel nehéz hatékony, gyorsan lefutó algoritmust találni a probléma megoldására.

1.4. Összegzés

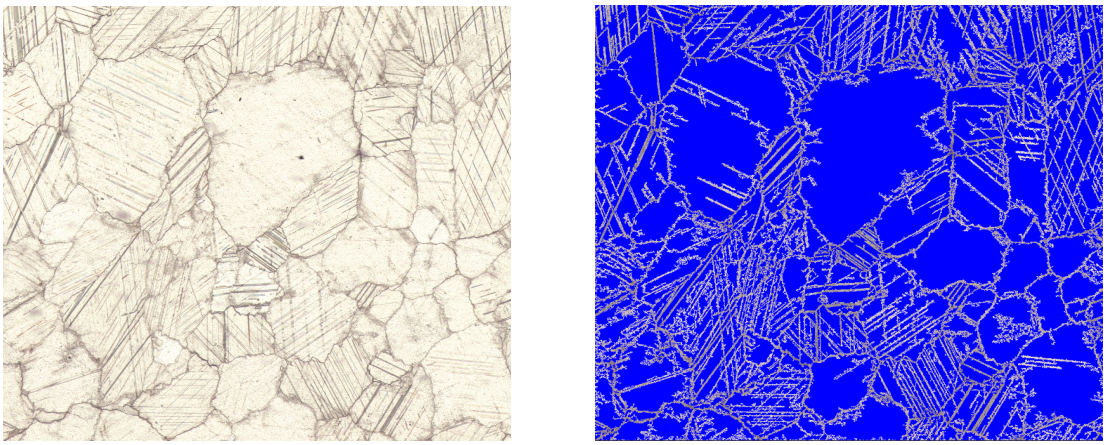
A dolgozatban bemutatott algoritmus, egy olyan szegmentáló algoritmus, ami tudja kezelni az ikerkristályokat. A szegmentálás eredményeként megtalálja a márványszemcsék határait. A módszer egy monta carlo alapú energia módszer lesz, amit rjmcmc algoritmusként fogok nevezni a későbbiek során. Az rjmcmc algoritmus a GrainAutLine eszköztárát gazdagítja, ami a CV4Sensorhub keretrendszerből is elérhető.



1.3. ábra. *Minta a CV4Sensorhub UI felületéről.*

2. fejezet

Az algoritmus elméleti háttere



2.1. ábra. *Példa túlszegmentálásra.*

Az alábbiakban bemutatom az algoritmus koncepcióját és részletesen tárgyalom annak elemeit. Alapvetően egy Monte Carlo alapú energiamódszerről van szó. Az alapötlet az az, hogy a képet hagyjuk túlszegmentálni [2] egy másik szegmentáló algoritmussal (adaptive thresholding, canny edge detektor stb.). Ennek hatására sok kis darabra osztódik fel az eredeti kép, mint ahogyan a (2.1. ábra) is mutatja a jobb oldalon. Az RJMCMC algoritmus feladata, hogy a túlszegmentált kép darabjait (blobokat) úgy rendezze nagyobb halmazokba, csoportokba (szuperblobokba), hogy azok fedjék a valódi szemcséket. Az átrendezés az alapján történik, hogy meghatározunk egy olyan energiafüggvényt, ami jellemzi az aktuális felosztás helyességét. Minél kisebb az energia értéke, annál jobb a felosztás (konfiguráció). A konfiguráció változtatását sorsolás alapján végzi az algoritmus, minden lépésben megvizsgálva egy blobot. Elsőként a konfiguráció módosításának menetéről, majd az energiafüggvényről lesz szó.

2.1. Az RJMCMC módszer

Ha adva volna egy megfelelő energiafüggvény, ami a konfigurációt jól jellemzi, akkor a helyes konfiguráció - amikor is a túlszegmentálás során kapott blobok, szemcsedarabkák olyan csoportba vannak rendezve, ami fedi a valós szemcséket - az energiafüggvény minimumánál

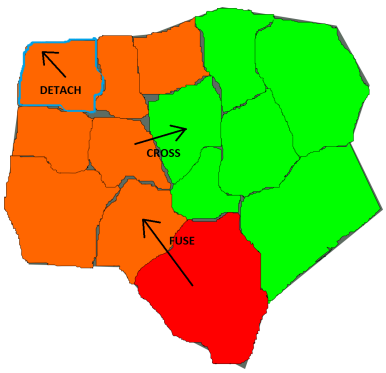
van. Tehát a feladat lényegében egy optimalizálás. Nem várható el, hogy a konfigurációs teren értelmezett függvény olyan mértékben tökéletes legyen, hogy a legkisebb változás a helyes irányba minden esetben energia csökkenést vonjon maga után. Ezen tökéletlenségek és a lokális minimumok elkerülése végett, az algoritmus a következő módosítást egy valószínűségi eloszlásból mintavételezve határozza meg. Így a konfigurációs teret jobban be lehet járni. Az optimalizációt az RJMCMC módszerrel ([1], [5]) valósítottam meg, ami a **R**eversible-**j**umped **M**arkov **c**hain **M**onte **C**arlo-nak a rövidítése.

A módszer többször végig iterál a teljes képen (az összes blobon) és mindegyik blobnál elidőz, melynek során megnézi annak környezetét és számba veszi a lehetséges átmeneteket. Meghatározza az egyes átmenetekhez tartozó energiaváltozásokat, amit a végrehajtásuk okozna. Ezen energiaváltozások alapján generál egy valószínűségi eloszlást, majd az alapján sorsolja a következő lépést.

2.1.1. Lehetséges átmenetek

A konfiguráció megváltoztatása 3 féleképpen lehetséges, attól függően, hogy milyen a blob környezete. A lehetséges átmenetek:

- FUSE: Ha a blob még nem tartozik semmilyen szuperblobhoz (csoporthoz), akkor lehetséges, hogy beolvadjon egy szuperblobba vagy összeolvadjon egy másik magányos blobbal.
- DETACH: Ha egy blob már része egy szuperblobnak, akkor ki is válhat belőle és újra lehet önálló. Itt még feltétel az is, hogy a leváló blob ne hagyjon lyukat az eredeti szuperblobban.
- CROSS: Ha egy blob része egy szuperblobnak, akkor átmehet egy másikba, feltéve, hogy van külső szomszédja.
- *STAY*: Triviális átmenet. Nem történik semmi, viszont ezt is figyelembe kell venni.



2.2. ábra. *A lehetséges átmenetek illusztrálása.*

2.1.2. Energiaváltozás az átmenetek során

Definíció: Az átmenethez tartozó energiaváltozás egyenlő az átmenet végrehajtása utáni– és előtti konfigurációs energia különbségével. Formalizálva:

$$T : \Omega \to \Omega'$$

$$\Delta E(T) = \epsilon(\Omega') - \epsilon(\Omega)$$

Ahol az Ω a konfigurációt, míg T az átmenetet jelöli. A $\Delta E(T)$ az átmenet során létrejövő energiaváltozás.
Könnyebbé teszi a számításokat, ha figyelembe vesszük, hogy a DETACH a FUSE ellentettje, míg a CROSS egy DETACH és egy FUSE egymásutánja. Így az energiaváltozások:

- FUSE: ΔE_{fuse}
- DETACH: $\Delta E_{detach} = -\Delta E_{fuse}$
- CROSS: $\Delta E_{cross} = \Delta E_{fuse}(new\ superblob) + \Delta E_{detach}(old\ superblob)$
- STAY: $\Delta E_{stay} = 0$

A hatékony kiszámításról az (3) részben lesz szó.

2.1.3. Az eloszlásfüggvény

Amikor megvizsgálja az algoritmus, hogy egy blobnak milyen lehetséges átmenetei vannak, akkor hozzárendeli az enegiaváltozásokat is, amik létrejönnének. Legyen adott az átmenetek halmaza:

$$\tau = \{T_i\}_i^r$$

Az r a lehetséges átmenetek számát jelöli az adott blobra.Ekkor az ehhez tartozó energiaváltozások:

$$\Sigma = \{\Delta E(T)\}_{T \in \tau}$$

Az eloszlásfüggvény egy Gibbs–eloszlással adott:

$$P(T_i) = \frac{1}{\sum_{\sigma_i \in \Sigma} e^{-\beta \cdot \sigma_i}} e^{-\beta \sigma_i} \tag{2.1}$$

A képlet lényege, hogy a nagy pozitív energiához ($\sigma_i > 0$), kis valószínűség tartozik, mert $\beta > 0$ (megkötés). Ha az energia negatív, akkor a valószínűség nagy, hiszen ott az exponens kitevőjében pozitív szám áll és mindig van negatív energiájú átmenet, ha van pozitív energiájú is, így az arányok fognak számítani. A β értéke lehet változó is, így szimulált lehűtés valósítható meg. Ez segíthet a konvergencia biztosításában.

2.1.4. Az elnevezés magyarázata

Az algoritmus az RJMCMC családba tartozik.

- **Monte Carlo**: A végrehajtandó átmenetet minden lépésben sorsolja a lehetséges átmenetek valószínűségeinek megfelelően. Az átmenetek valószínűsége a Gibbs–eloszlás alapján számolódik.
- **Markov chain**: A következő állapot a jelen állapot szerint feltételesen független a megelőző állapottól. Ennek megfelelően a konfigurációk elsőrendű markov láncot alkotnak. Az állapot most a konfiguráció és a vizsgált blob együttesét jelenti.
- **Reversible–jump** Ez lényegében arra utal, hogy lehetséges a bloboknak kiválnia a szuperblobokból. A blob is tekinthető egy speciális szuperblobnak, így a szuperblobok száma nőhet és csökkenhet is. Tehát minden folyamat megfordítható, reverzibilis a rendszerben.

2.2. Az energiafüggvény

Az alábbiakban a konfigurációt jellemző energiafüggvényről lesz szó. Az energiafüggvénynek olyannak kell lennie, hogy értéke minimális annál a konfigurációnál, ahol a szuperblobok határai megegyeznek a valós szemcsék határaival. Hasonló megfontolásokra található példa a ([4]) összefoglaló írásban, valamint a ([3]) cikkben. Előbbi általában világítja meg az energia alapú módszerek hatékonyságát és alkalmazhatóságát. Az utóbbiból viszont egy fontos koncepciót merítettünk. A cikk az áramkörök forrasztásainak hibáit detektálni képes képfeldolgozó algoritmust mutat be. A megoldásuk egyik kulcsa, hogy az energiafüggvényt több különálló energiatag összegeként írják fel. Az egyes energiatagok különböző célokat szolgálnak, mint például az alakzatok átlapolódását figyelembe venni stb.
Az itt alkalmazott energiafüggvény 3 energiatag összegéből áll. Mindegyik energiatag megragadja a valós szemcsék valamely fontos tulajdonságát. Minden energiatag tartalmaz hiperparamétereket, amelyek finomhangolásával a függvény a körülményekhez igazítható. Fontos, hogy a cél a megfelelő konfiguráció megtalálása, ezért nem fontos és hatékonyság szempontjából is jobb, ha nem számoljuk ki egy adott konfigurációra az energiát, hanem csak a megváltozásokat számoljuk ki az egyes átmenetekhez. Ezt azért lehet megtenni, mert a konfiguráció változtatásában mindig csak néhány blob érintett. Így az energia számítása lényegesen gyorsabb.

2.2.1. SeedPoint energiatag

Ez az energiatag a felhasználó segítségén alapul. A folyamat legelején a felhasználó minden szemcsében elhelyez egy jelölést (seed pointot). Ezek ismeretéből két dolog következik:

1. A szemcsék száma egyenlő a seed pointok számával.
2. Minden szemcsében pontosan egy seed point van.

Az utolsó állításból az következik, hogy minden szuperblobnak egy seed pointot kell tartalmaznia. Az egyes átmenetek során az érintett blobok és szuperblobok ellenőrzésével az eltérések észlelhetők. Az eltérés mértéke alapján megfogalmazható egy energiatag, amelynek célja, hogy olyan átmeneteket részesítsen előnyben, amik esetén a fenti két feltételnek nem eleget tevő szemcsék száma csökken. Ehhez lényegében elegendő a második feltételt ellenőrizni, mivel, ha az fennáll, akkor az első is teljesül.
A (2.1.2) részben leírtak szerint elég a FUSE esetét megoldani, mivel abból a többi már következik. A fentieket jól kifejezi a következő konstrukció:

$$\Delta E(FUSE) = \begin{cases} 0, \ if \ S_A = S_B = 0 \\ -\lambda, \ if \ S_A + S_B = 1 \\ \Phi_0 \cdot \exp(S_A + S_B), \ if \ S_A + S_B \geq 2 \end{cases} \tag{2.2}$$

A fenti formula arra érvényes, hogy az A blob beolvad egy B szuperblobba. Az S_A jelöli az A-ban levő seed pointok számát, míg az S_B a B-ben levőkét. A λ és a Φ_0 hangolható paraméterek. A képlet mögötti gondolat a következő:

- Ha sem a blobban és sem a szuperblobban nincs seed point, akkor ezt közömbösnek tekinthejük, ezért nem okoz energiaváltozást.
- Ha az egyesülés után pontosan egy szuperblob lesz (egyesülés előtt csak az egyikük tartalmazott seed pointot), akkor ez negatív energiaváltozást okoz, ami csökkenti az energia értékét, vagyis ez egy kedvező átmenet lesz.
- Ha egyesülés után több, mint egy seed point lesz a keletkező szuperblobban, akkor az egy olyan átmenet, amit el kellene kerülni, mert ez a csoportosítás nem lehet helyes. Ezért az energia nőni fog. Az is számít, hogy mennyivel lesz több seed point, ezért egy exponenciális függvényt használunk. Így a növekvő seed point szám gyorsan növekvő pozitív energia változást okoz.

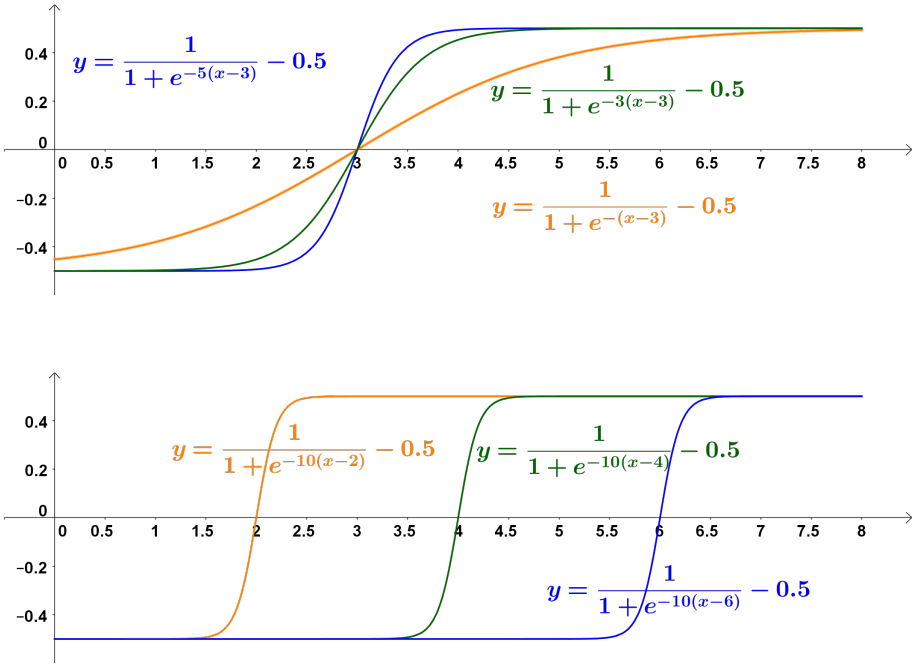
2.2.2. StraightLine energiatag

Ez az energiatag kifejezetten az ikerkristályok kezelését szolgálja. Az ikerkristályok számos egyenes vonalat tartalmaznak belül (1.2. ábra), így az egyenes vonalakra is érdemes egy energiatagot definiálni. Itt is elegendő a FUSE esetét vizsgálni, mert a többi következik belőle a (2.1.2)-ben leírtak szerint. A FUSE során a beolvadó blob és a fogadó szuperblob közös határvonala el fog tűnni. Az energiaváltozás annak a határvonalnak egy speciális jellemzője alapján számolható. Ez a speciális jellemző a „curveness”:

$$\zeta = \min_{\vec{n}} \sqrt{\frac{1}{N} \sum_{i}^{N} \left(d(p_i, \vec{n}) - \sum_{i}^{N} (\vec{r}_i - \vec{r}_0)\, \vec{n} \right)^2} \tag{2.3}$$

Szavakkal: A pontok valamely egyenestől mért távolságainak átlagától való átlagos négyzetes eltérés gyökét kell minimalizálni az egyenes irányának függvényében. A (F.1) függelékben le van vezetve a ζ kiszámítása. A ζ ismeretében az átmenethez tartozó energiaváltozás a következő formulával számolható:

$$\Delta E(FUSE) = E_0 \cdot \left(\frac{1}{1 + e^{-\kappa(\zeta - c)}} - \frac{1}{2} \right) \tag{2.4}$$



2.3. ábra. *Az energiafüggvény szemléltetése. A felső kép a κ, míg az alsó a c hatását szemlélteti.*

Az energiatag lényege, hogy az egyenes vonalakra a curveness (ζ) értéke kicsi, tökéletes egyenesnél 0. Ekkor az energiaváltozás negatív a FUSE esetén, ami megfelelő, hiszen az egyenes közös határvonal arra utal, hogy egy ikerkristály belsejében van az átmenet. A beolvadás tehát előnyös, ezt pedig negatív értékkel lehet támogatni a sorsolás szempontjából.

2.2.3. Konvexitás energiatag

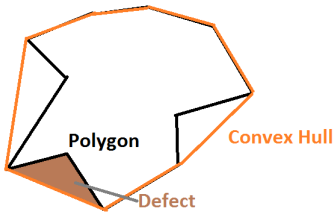
Ez az energiatag arra a megfigyelésre épít, hogy a szemcsék többnyire konvexek, nem jellemző nagy számban a kirívóan konkáv határoló vonallal rendelkező szemcse. Ezért bevezetésre került egy olyan energiatag, ami ezt hivatott kifejezni. Az alakzatokra definiált konkavitás mérték segítségével definiáltam az energiaváltozást. A konvexitásra nehéz jó mértéket találni, mert mind a terület– és kerület alapú megközelítésnek vannak hátrányai, anomáliái. Remek áttekintést ad erről a ([7]) cikk. Az energiatag szempontjából inkább a konkavitást mértem. A konvkavitást ($conv$) jelen esetben az alábbi összefüggés adja:

$$conv = \frac{\sum_i (defect\ area_i)^k}{(total\ area\ of\ polygon)^k} \tag{2.5}$$

Ahol k egy 1-nél kicsit nagyobb szám. A hatványozás célja, hogy a nagyobb méretű horpadások jelentősége nagyobb legyen. A fogalmakat a (2.4.) ábra szemlélteti. Az *energia* egy adott alakzatra, tehát most nem energiaváltozást számolunk(!):

$$E_{convexity} = Threshold \cdot \left(1 - e^{-factor \cdot conv}\right) \tag{2.6}$$

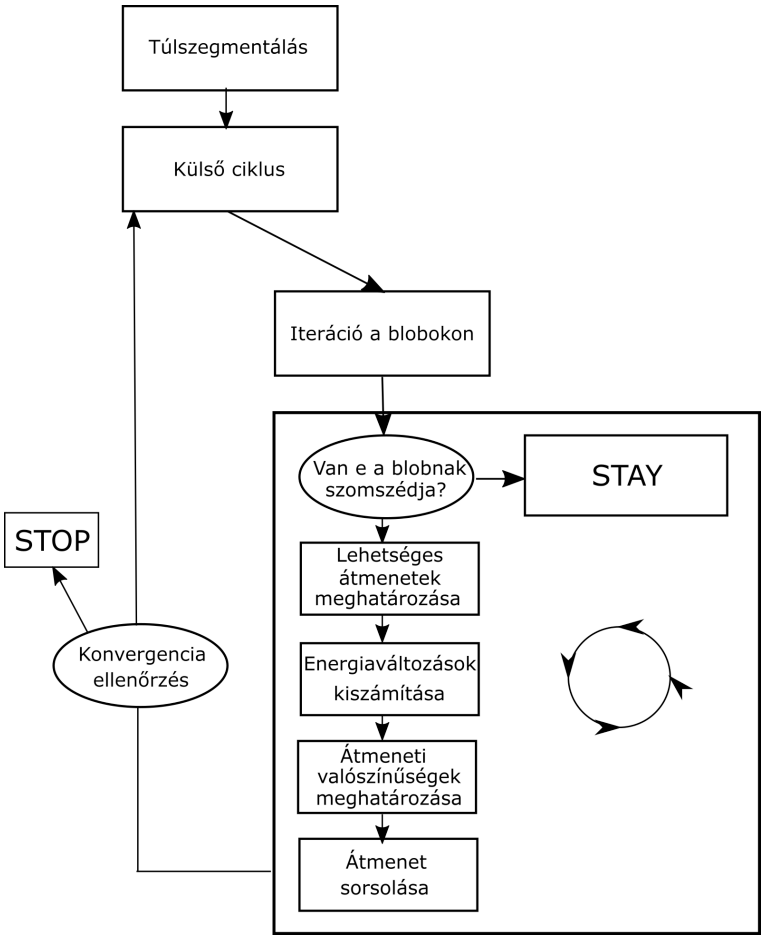
Az átmenetnél az energiaváltozás az átmenet után létrejövő szuperblobok energiáinak összege és a résztvevő szuperblobok energiáinak az összegének a különbsége. Itt nem érdemes mindent visszavezetni a FUSE esetére, mert a CROSS-nál lehet spórolni a konvkavitás számolással, ha tényleg a kezdeti- és végállapotokra számolunk közvetlenül.



2.4. ábra. *A konvkavitás számítás során használt mennyiségek szemléltetése.*

Megjegyzés: Korábban egy un. mutexpárokon alapuló módszer segítségével határoztuk meg a konkavitást egy sokkal bonyolultabb algoritmussal. Ennek futásideje, annyira hosszú volt, hogy egy másik megoldást kellett találni.

2.3. A teljes algoritmus



2.5. ábra. *A teljes program működésének vázlata.*

Az algoritmus működését a (2.5. ábra) mutatja. Látható, hogy két egymásba ágyazott iteráció van. A belső iteráció a blobokon való végig iterálást végzi, míg a külső az iterációkat ismétli addig, amíg konvergencia nem lesz. A belső iteráció végig megy a blobokon és minden lépésben ellenőrzi, hogy a soronkövetkező blobnak milyen lehetséges átmenetei vannak. Ezután az egyes átmenetekre kiszámolja az energiaváltozásokat, amik akkor lennének, ha az átmenetet végrehajtaná. Az energiaváltozás a három energiatag összegeként adódik:

$$\Delta E_{total}(T) = \Delta E_{SeedPoint}(T) + \Delta E_{StraightLine}(T) + \Delta E_{Convexity}(T).$$

Az energia kiszámítását követően meghatározza mindegyik átmenetre a valószínűségeket:

$$P(T_i) = \frac{1}{\sum_{\sigma_i \in \Sigma} e^{-\beta \cdot \sigma_i}} e^{-\beta \sigma_i} \tag{2.7}$$

Az eloszlásfüggvényt figyelembe véve (mindegyik átmenet a valószínűségének megfelelő valószínűséggel kerülhet kiválasztásra) kisorsolja az átmenetet, amit utána végrehajt.

3. fejezet

Az algoritmus implementációja

A CV4Sensorhub keretrendszer *C#* nyelven íródik. A User Interface WPF alapú, amiben könnyen lehet fejleszteni a felhasználó munkáját segítő funkciókat és megjelenítő elemeket. Az rjmcmc algoritmus is *C#*–ban készült. A *C++*–hoz képest lassabb kódot kaphatunk, de a futásidő, mint látni fogjuk, így is kielégítő. A teljesítmény különbség mellesleg nem mindig számottevő a JIT fordító miatt, továbbá a CLI segítségével lehetőség van *C++* kódot hívni, ami a teljesítmény kritikus részeket kezelhetővé teszi. Az algoritmus megvalósítása során felmerültek problémák, amelyeket meg kellett oldani:

- Olyan adatszerkezet kell, amin az egyes átmenetek gyorsan megvalósíthatók és az energia számításokhoz szükséges minden információ könnyen elérhető.
- A konvexitás energiataghoz a defectek (horpadások) megtalálása, területük kiszámítása.
- A szemcsék nem tartalmazhatnak lyukakat, vagy más szemcséket belül. Ezért minden átmenet esetén ellenőrizni kell, hogy keletkezik–e lyuk.
- A számítások során sok exponenciális függvény fordul elő, ráadásul egymásután, lényegében egymásba ágyazva. (Először az energiatag, majd a Gibbs-eloszlás is használja.) Ennek eredményeként könnyű NaN-t kapni.
- A végeredményben könnyen felismerhetővé kell tenni a kialakult szuperblobokat, amit színekkel meg lehet oldani. Azonban a sok szuperblob miatt sok különböző színt (több, mint 100–at) kell generálni.
- A lefutás végén meg kell mérni az eredmény pontosságát. Lehetőleg olyan metrikával, ami a felhasználó szemszögét tükrözi.
- Az algoritmus leállása konvergencia esetén történik. Ezt ellenőrizni kell.

Ebben a fejezetben a felsorolt problémákra adott megoldásokról lesz szó.

3.1. Adatszerkezet

Olyan adatszerkezetre van szükség, amin az átmenetek gyorsan végrehajthatóak. A szuperblob a blobok egy halmaza. A blobot megvalósító adatszerkezet mindenképp tárolja a túlszegmentálás során keletkezett szemcsét határoló vonalat, ami egy poligon. A szuperblob reprezentációjának nem lenne hatékony, ha a szuperblobokat is blobként reprezentálnánk, hiszen az átmenet során rengeteg műveletet jelentene az új szuperblobot határoló poligon meghatározása. Ehelyett a blob adatszerkezete tárolja, hogy az adott blob melyik szuperblobba tartozik. Így mindegyik blob tudja, hogy melyik szuperblobnak a része. A CV4Sensorhub keretrendszer eleve úgy lett kialakítva, hogy erre már fel van készítve. Úgynevezett *tag*–ek használatával lehet különböző tulajdonságokat (most ez egy párt jelöl: tag név és érték, nem a *C#*–os tulajdonságról van szó) adni az ún. entitásokhoz, amelyeknek a leszármazottja a blob is. Az, hogy a blobok tárolják, hogy melyik szuperblobhoz tartoznak, ott jelenthet problémát, mikor a fordított irányba kell lekérdezni. Azaz szeretnénk tudni, hogy a szuperblobhoz, mely blobok tartoznak. Erre is van beépített megoldás, mivel a keretrendszer karban tart egy speciális indexelőt, ami a fordított irányú hozzáférést támogatja. Így a kiolvasás mindkét irányba csak minimális keresést igényel.
Egy ilyen adatszerkezettel az átmenetek végrehajtása gyakorlatilag ingyen van, mivel csak átindexeléseket kell végrehajtani. Ráadásul az átmenetek úgy vannak definiálva, hogy minden átmenetben egy blob és maximum két szuperblob (CROSS esetén) vesz részt, ezért az átindexelés csak a blobon történik, ami csak *egy* átindexelést jelent!

3.2. Konvexitás energiatag

A konvexitás kiszámítása emlékeztetésül az alábbi képlettel történik:

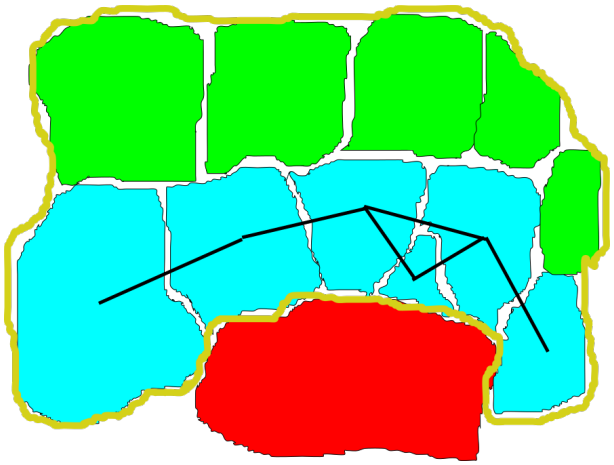
$$cvx = \frac{\sum_i (defect\ area_i)^k}{(total\ area\ of\ polygon)^k} \tag{3.1}$$

A $defect\ area$ meghatározása az OpenCv segítségével történt. Az OpenCv–hez van olyan NuGet package (OpenCvSharp), amivel lehet az OpenCv–s függvényeket használni. A horpadások (defectek) meghatározásához szükség volt a ConvexHull és a ConvexityDefects függvényekre. A horpadások területét ($defect\ area$) közelítőleg egy háromszög területével számoltam. A ConvexityDefect visszaadja a defektek kezdő és végpontjának indexét a konvex burkon, valamint a mélységét. Az előbbiből következik a két pont közti távolság, ami a háromszög alapja lesz, míg az utóbbi a háromszög magasságának tekinthető. Ezekből a háromszög területe számolható. Mivel a blobok midig poligonnal vannak reprezentálva, sok esetben a defekt egy háromszög, így a közelítés valójában sokszor pontos.

3.3. Ellenőrzés, hogy keletkezik–e lyuk az átmenet során

A tapasztalat szerint egy szemcse sosem tartalmaz más szemcséket, illetve a szemcsék zártak, egyszeresen összefüggő tartományt alkotnak, nincsenek bennük lyukak. Ez egy fontos szempont, amit korábban egy másik energiataggal próbáltam kivédeni. Ha lyuk keletkezett

az átmenet során, akkor ahhoz nagy pozitív energiaváltozás tartozott. Az energiatag nem volt túl hatékony, ráadásul a lyukakat ki lehet védeni még a lehetséges átmenetek kiválasztása során azzal, hogy azokat az átmeneteket, amelyek lyuk keletkezéséhez vezethetnek, nem vesszük figyelembe, kizárjuk őket.
A lyukak keletkezésének vizsgálata, úgy történik, hogy elkészítünk a blobokra egy szomszédossági gráfot. Ez az eljárás a keretrendszerben már implementálva volt. Ennek alapján tudni lehet, hogy az egyes bloboknak mely blobok a szomszédai, viszont az, hogy milyen sorrendben vannak egymáshoz viszonyítva azt nem. Vegyük először a FUSE esetét. A blob (ami be fog olvadni) szomszédaiból kiválogatjuk azokat, amik a fogadó szuperblobnak is a részei. Ezután tekintsük azt a gráfot, aminek a csúcsai az előbb kiválogatott blobok és két csúcs közt vezet él, ha a két blob szomszédos. A FUSE után is lyukmentes a szuperblob, ha az így nyert gráf összefüggő. Ezt például BFS–sel lehet ellenőrizni. A DETACH esetén is pontosan ez az eljárás. A különbség a megvalósításban ott van, hogy a DETACH–nél a blob része a szuperblobnak, a FUSE–nál még nem. Ez az indexeknél jelent némi eltérést, de az algoritmus alapja ugyanaz.



3.1. ábra. *A lyuk ellenőrzésére szolgáló algoritmus által használt gráf szemléltetése. A sárga körvonal mutatja a szuperblobot. A piros blob fog beolvadni. A kék blobok a piros blob szuperblobbeli szomszédait jelöli. A fekete szakaszok a létrehozott gráf élei. Akkor nem keletkezik lyuk, ha ez a gráf összefüggő.*

3.4. Overflow elkerülése

Az algoritmus kezdeti próbálgatása során sokszor jött elő a probléma a NaN (not a number) nevű eredménnyel. Ez kivétel nélkül az exponenciális függvény túlzott megnövekedésének, vagy lecsökkenésének lett az eredménye. Ezt elkerülendő a kitevő abszolútértékét lekorlátoztam. Tehát:

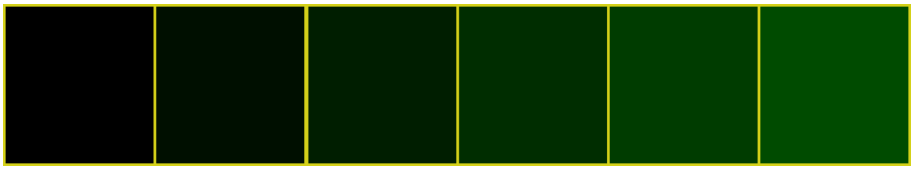
$$k_{safe} = \min(|k_{original}|, k_{max}).$$

Ez nem jelent problémát az algoritmus helyessége szempontjából, mert a nagy energia érték, ha pozitív, azt mutatja, hogy az átmenet kerülendő. Ha több ilyen is van, akkor a

nagyon rosszak közül mindegyiket kerülni akarjuk, ezért ezek valószínűsége kicsi. Mivel a STAY mindig opció, ezért ilyen esetben mindig az fog választódni. Ha az energia nagy, de negatív irányba, akkor az átmeneti valószínűség releváns lesz. Ha több ilyen is van, akkor a nagyon jó esetek közül lényegében bármelyik jó, nem kell szigorúan különbséget tenni köztük. Ez összecseng a Monte Carlo módszerek filozófiájával, miszerint érdemes a rossznak, vagy rosszabbnak tűnő lépéseknek is esélyt adni, hátha végül mégis az lesz a jó (exploration).

3.5. Megjelenítést segítő színgenerátor

Az algoritmus lefutását követően mindegyik szuperblob kap egy saját színt, hogy meg lehessen őket különböztetni. A színeket a szuperblob azonosító száma alapján egy előre definiált színsorból kapják. A színsor determinisztikusan készül. Az R, G és B skálákat 15 egyenlő részre osztottam. Ezután az így kapott értékeket az összes lehető módon számhármasokba állítottam, oly módon, hogy az (R, G, B) hármasban mindig a B értékét növeltem 15–tel, amíg a 255–öt el nem éri. Mikor eléri, akkor a G–t növelem 15–tel, majd újra a B-t. Ha a G és B eléri a 255–öt, akkor az R növekszik 15–tel, majd kezdődik előről a G és B változtatása a nulla értékről. (Olyan mintha egy háromjegyű 2555–ös számrendszerbeli számot léptetnénk 15–ösével.) Az így kapott listának minden 15–dik színét kiválasztva adódik a generált színsor. A szuperblob a saját azonosítójának megfelelő helyről kapja a színét. Ezzel a módszerrel 255 különböző szín kapható. A jelenlegi tesztekhez ez elég. Ha közelebbi színeket választunk, akkor többet is lehet generálni.



3.2. ábra. *A generált színsor első néhány eleme.*

A (3.2. ábra) mutatja az első néhány elemét a színsornak. Vizuálisan elég közel vannak egymáshoz, viszont a blobok, amik közel vannak, közeli azonosító számokkal rendelkeznek. A blobok azonosítójából lesz a szuperblob azonosító, attól függően, hogy melyik blobba olvad bele másik blob (kezdetben minden blob szuperblob és csak FUSE és STAY lehet). Mivel kezdetben az egymás melletti blobok olvadnak össze, így a közeli szuperblob azonosítók kieesnek. Az egymás melletti színek tehát ritkán fordulnak elő egymáshoz közel. Így ez a skála megfelelő. A programnak továbbá van olyan funkciója, ami alapján a *tag* értékei lekérhetők egérkattintással, így a kétes esetek könnyen feloldhatók.

3.6. Az algoritmus pontosságának mérése

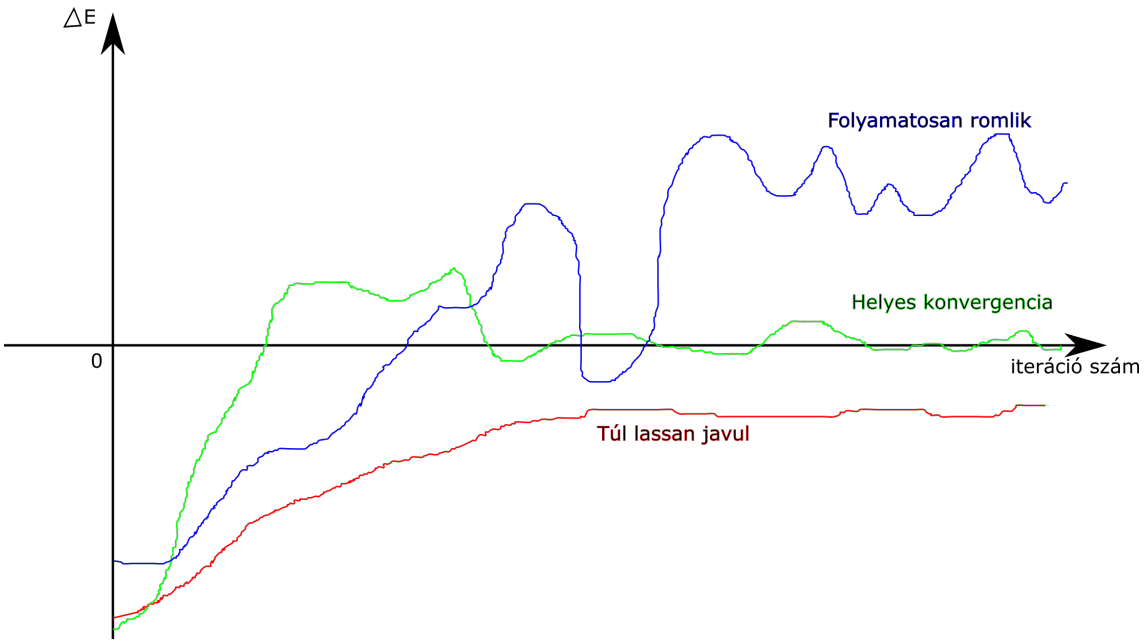
A végeredményt a tesztek során ki kellett tudni értékelni. Ennek érdekében meg kell mérni a pontosságot, egy előre elkészített, helyesen szegmentált képhez viszonyítva. Azonban a két szegmentálás igen változatos eltéréseket tud mutatni. Eltérhetnek a szuperblobok szá-

mában. Lehet, hogy egy szuperblob, amit az RJCMC algoritmus ad, átlapol több helyesen szegmentált szuperblobbal. Tehát a szuperblobonkénti összehasonlítás nem lehetséges. Jó mérték lehetne a helyreállításhoz szükséges minimális lépésszám az algoritmus szempontjából: hány FUSE, DETACH kell minimálisan, hogy a helyes szegmentálást megkapjuk az RJMCMC algoritmus eredményéből. Tekintve, hogy a cél a felhasználó munkájának egyszerűbbé tétele, így mérvadóbb az a mérés, mely a helyreállítást a felhasználó szemszögéből nézi. Azaz mennyi vágást, összeolvasztást kell végeznie átlagosan egy szuperblobon, amik az RJMCMC algoritmus eredményeként adódtak.

3.7. Konvergencia ellenőrzése

A konvergencia ellenőrzése (tágabban, hogy mikor terminálj on az algoritmus) három szintű hierarchikus ellenőrzés eredményeként adódik.

1. Meghaladta e már a külső iterációk (az egész képen való végig söprések) száma a kritikus értéket.
2. Több teljes végig söprés alatt keletkező teljes energiaváltozások átlaga kisebb egy küszöbértéknél, tehát már nem változnak jelentősen a szuperblobok.
3. Több teljes végig söprés alatt keletkező energiaváltozások átlaga tartósan egy adott pozitív érték felett van és már túl vagyunk az algoritmus indulásán egy meghatározott lépésszámmal, ekkor is megáll.



3.3. ábra. *Az ellenőrzés által szűrt különböző esetekre példa.*

A (3.3. ábra) alapján jobban elmagyarázható, hogy mikor érdemes megállítani az algoritmust. Az energiaváltozás minden iterációra *újra számolódik*. Ennek értékeit látjuk az iterációk számában. Ha tartósan negatív, de nagyobb a konvergencia küszöbénél, akkor az

azt jelenti, hogy egy iteráción belül túl sok olyan átmenet van, ami visszarontja a konfigurációt. Összességében javul, de túl lassan (piros görbe).
Ha már kevés átmenet történik (sok a STAY), vagy többnyire csak fluktuáció van oda–vissza a konfigurációban, akkor az energiaváltozás kicsi lesz iterációnként. Ezek átlaga nulla körül kell legyen (zöld görbe).
Ha az energiaváltozás folyamatosan pozitív, akkor az azt jelenti, hogy a rossz hatású átmenetek az uralkodóak, így a konfiguráció folyamatosan romlik (kék görbe).

4. fejezet

Az algoritmus tesztelése, értékelése

Ebben a fejezetben az algoritmus kirpóbálásának eredményeiről lesz szó. A tesztelés az alfejezet címeknek megfelelően három részre osztható. Elsőként megnéztem, hogy külön–külön az egyes energiatagok mit csinálnak rajzolt ábrákon. Mindegyikhez készítettem ábrákat, amelyeknek a kimenetele triviális. Így egyértelműen látható az adott energiatag hatása. A második részben megnéztem, hogy a kombinációjuk milyen hatást gyakorol igazi bemeneten. A túlszegmentálást kézzel végeztem az ábrán, hogy csak ennek az algoritmusnak látszódjon a hatása és az előszegmentálás hibája ne okozzon félrevezetést (ceiling analysis). Végül a konvergencia tulajdonságait vizsgáltam az algoritmusnak.

4.1. Futtatási eredmények energiatagonként

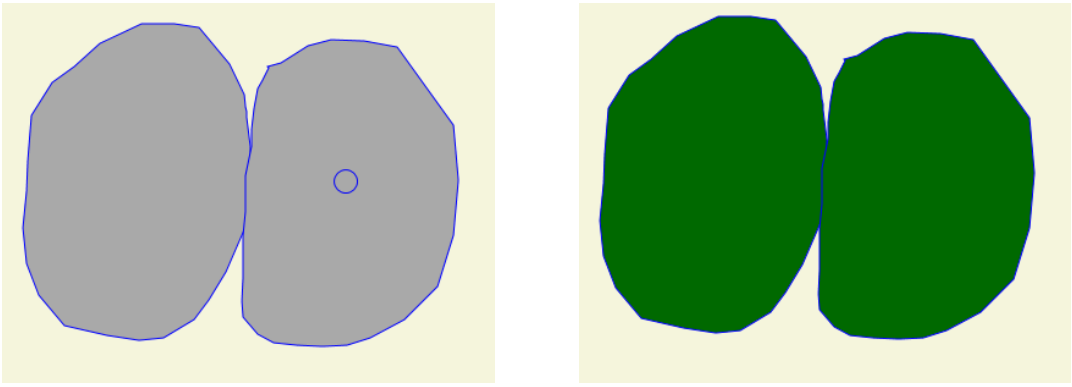
Ebben a részben az egyes energiatagok egyenként kerülnek kipróbálásra. Az ábrák baloldalán a kiindulási állapot, míg a jobb oldalt az algoritmus futásának eredménye látható.

4.1.1. SeedPoint energiatag

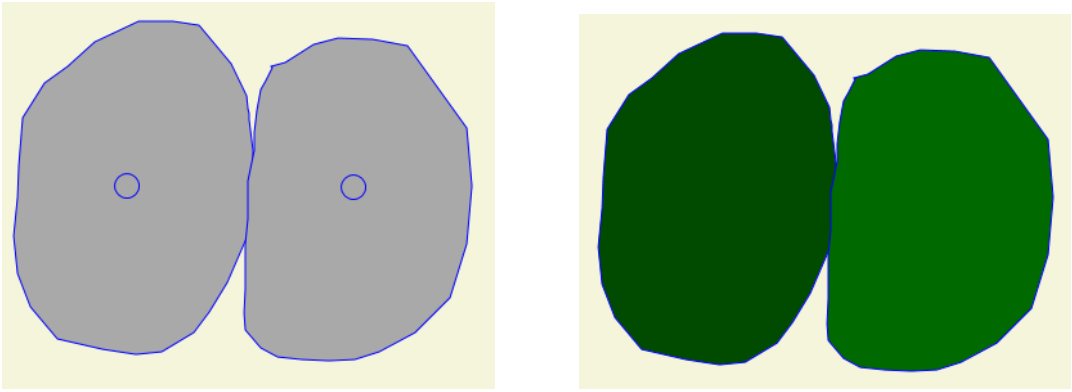
Ebben az esetben a hangolható paraméterek:

- λ, Φ_0: A SeedPoint energiatag hiperparaméterei (2.2.1).
- β: A Gibbs–eloszlás paramétere (2.1.3). Mindig pozitív.

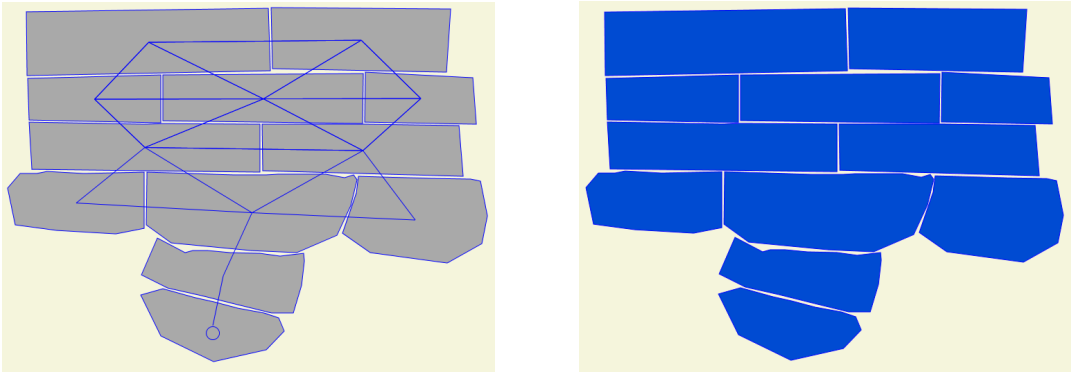
Az itt látható eredmények esetén a paraméterek értéke: $\lambda = -10$, $\Phi_0 = 3$ és $\beta = 1$. A futtatásoknál használt iterációk száma most fixen 10 volt minden esetben. Valójában 2 iteráció is elegendő bármely ábra reprodukálására. A 10 iteráció miatt látható, hogy a végeredmény stabil marad, nem csak egy jó pillanatot sikerült el fogni. Az első két eset (4.1. ábra, 4.2. ábra) a legegyszerűbb szituációkon ellenőrzi a helyes működést. Az utolsó három bonyolultabb esteket vizsgál.
A (4.1. ábra) szegmentációja során kétféle eset jöhet szóba FUSE és STAY. Az előbbire az energiaváltozás –10, míg az utóbbira 0. Az ezekhez tartozó valószínűségek: 0.9995 illetve 0.0005. Tehát fölényesen a FUSE nyer. A (4.2. ábra) esetén is a FUSE illetve a STAY lehetséges. Az energiaváltozás FUSE esetén 403, STAY esetén 0. A megfelelő valószínűségek: 3.77E–44, ami lényegében nulla, a másiké gyakorlatilag 1.



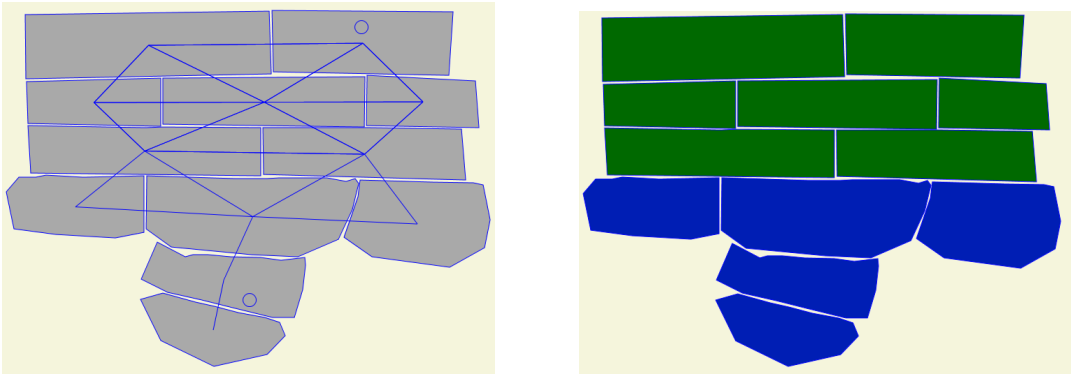
4.1. ábra. *A kiindulási állapot az, hogy egy seed point van a jobb oldali blobban. Mivel minden szuperblobban pontosan egy seed point lehet, így a két blobnak össze kell olvadnia. A jobb oldali ábra mutatja, hogy az algoritmusnak valóban ez az eredménye.*



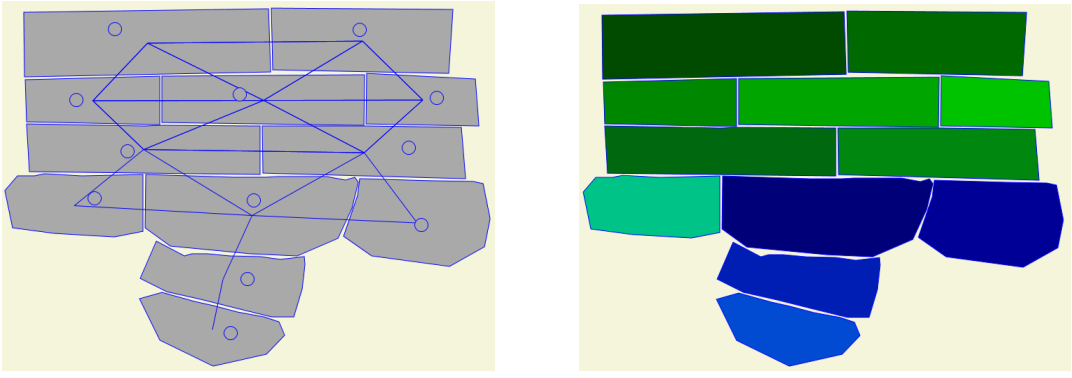
4.2. ábra. *A kiindulási állapot az, hogy mind a két blobban van egy seed point. Mivel minden szuperblobban pontosan egy seed point lehet, így a két blobnak nem szabad összeolvadnia. A jobb oldali ábra mutatja, hogy az algoritmusnak valóban ez az eredménye.*



4.3. ábra. *A kiinduló állapot az, hogy több blob van, de csak az egyik szélsőben van seed point. Ekkor is az egésznek össze kell olvadnia. Ez a példa nehezebb az eddigieknél, hiszen most csak több lépés után tudja az egészet egy nagy szuperblobbá olvasztani. Ez sikerült is az algoritmusnak.*



4.4. ábra. *Ebben az esetben két távoli blobban van seed point. Az elvárt eredmény az, hogy a blobok egy része az egyik szuperblobban, míg a másik része a másik szuperblobban van. A végeredmény tényleg ez lett és nincs lyuk sem, ami szintén fontos.*



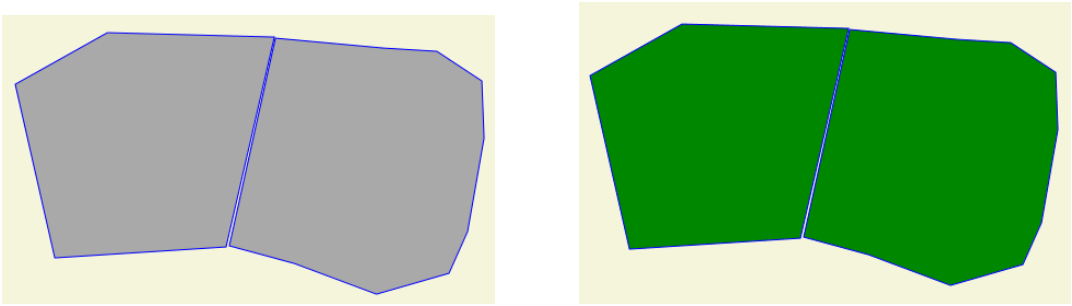
4.5. ábra. *Itt minden blobban pontosan egy seed point van. Itt mindegyik blobnak külön szuperblobnak kell lennie. A végeredmény valóban ez lett, amit a különböző színek jeleznek.*

4.1.2. StraightLine energiatag

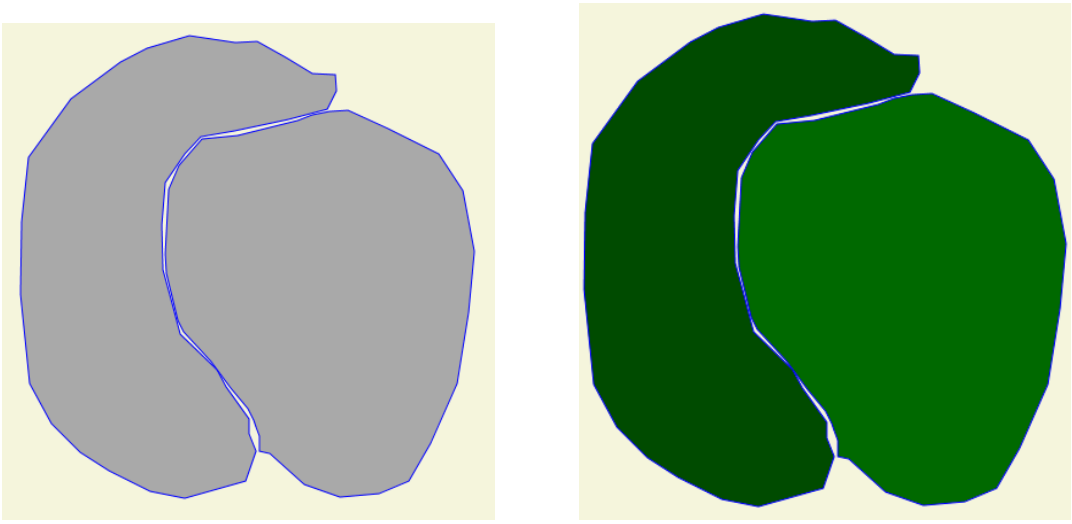
A beállított hiperparaméterek:

- δ: A curveness mekkora értéke alatt tekintjük egyenesnek a vonalat (2.3.).
- γ: Mennyire tekintjük élesnek a görbe és az egyenes közti átmenetet.
- E_0: Az energiaváltozás lehető legnagyobb értéke.
- β: A Gibbs–eloszlás paramétere.

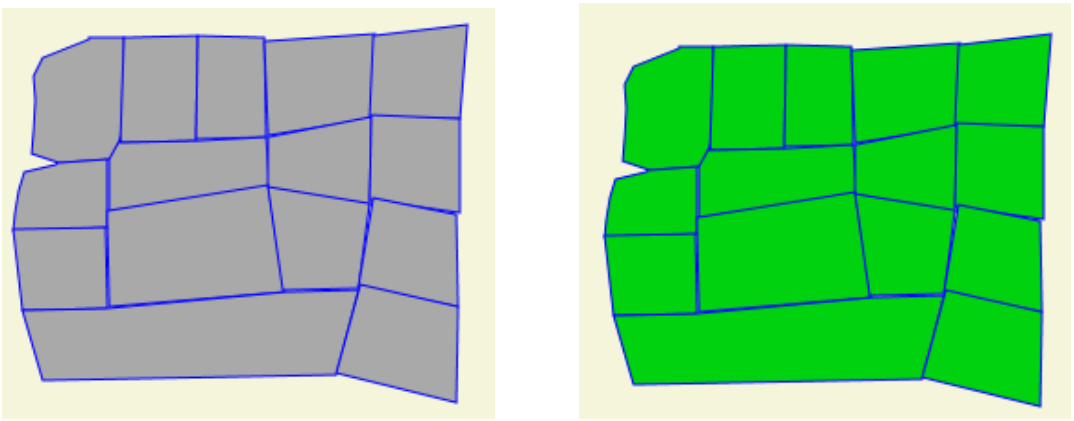
A konkrét beállított értékek: $\delta = 5$, $\gamma = 1$, $E_0 = 10$ és $\beta = 1$.
Az első esetben (4.6. ábra) két átmenet lehetséges: FUSE és STAY. A létrejövő energiaváltozások: –2.51 és 0. A valószínűségek: 0.925 és 0.075. Tehát tényleg a FUSE a valószínűbb.



4.6. ábra. *A két blobot egy egyenes vonal választja el. Ez azt jelenti, hogy a két blobnak össze kell olvadni. Az eredmény ezt igazolja.*



4.7. ábra. *A két blob határa egy görbe vonal. Itt nem lehet összeolvadás. Az eredményt a jobb oldal mutatja.*



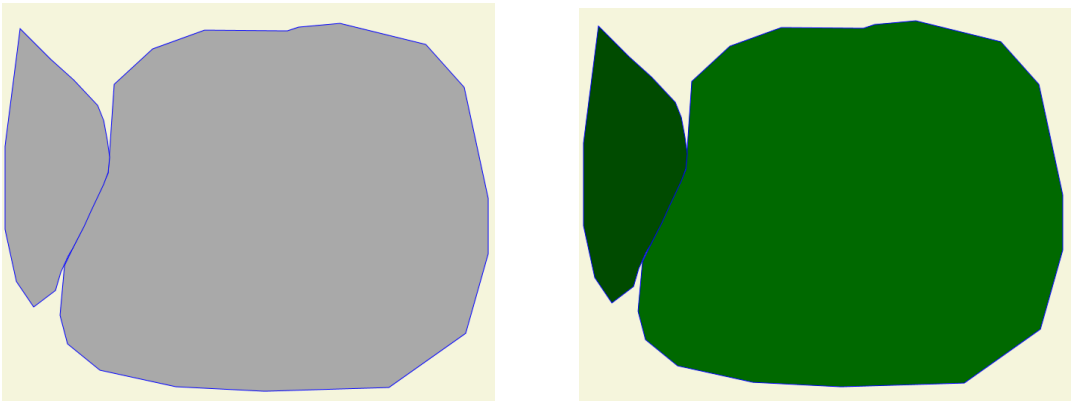
4.8. ábra. *A képen olyan blobok vannak, amiket egyenes vonalak választanak el. Itt is az elvárt az, hogy képesek legyenek egy nagy szuperblobbá összeolvadni.*

4.1.3. Konvexitás energiatag

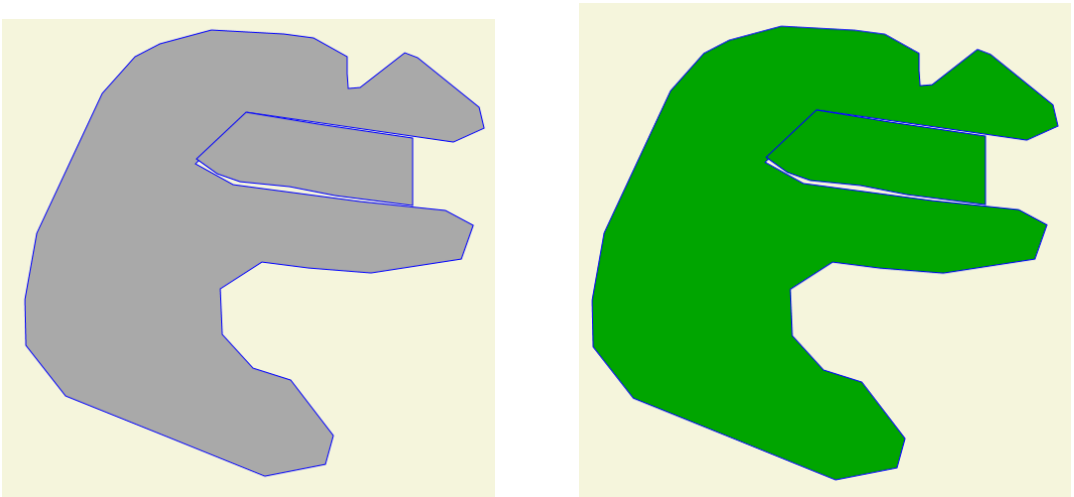
A hangolható hiperparaméterek:

- k: A nagobb horpadások nagyobb súllyal számítanak, ezt lehet vele szabályozni (2.2.3).
- $factor$: Az energiaváltozás szaturálódásának a sebességét szabályozza. Mindig pozitív.
- $threshold$: A maximális energia, ha nagyon konkáv az alakzat.
- β: A Gibbs–eloszlás parmétere.

Az egyes paramétereket a következő módon állítottam be: $k = 1.1$, $factor = 1.5$, $threshold = 20$ és $\beta = 1$.



4.9. ábra. *A két blob külön–külön kis konkavitású, de összeolvadás után egy konkáv alakzat keletkezne. Ezért itt nem történik meg az összeolvadás.*

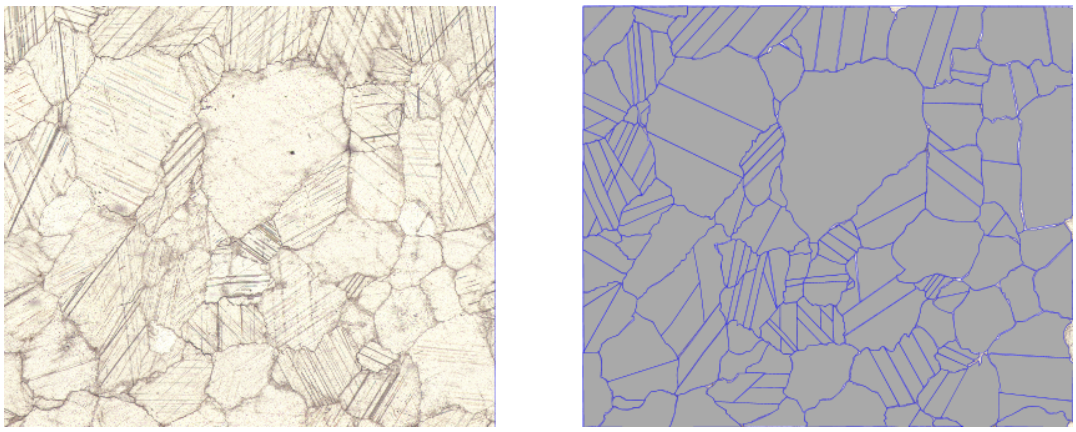


4.10. ábra. *A két blob közül az egyik szemmel láthatóan nagy konkavitású, míg a másik konvex. A konvexebb blob olyan helyen van, ami jelentősen csökkenti a konkavitását az alakzatnak. Ezért összeolvadásnak kell történnie.*

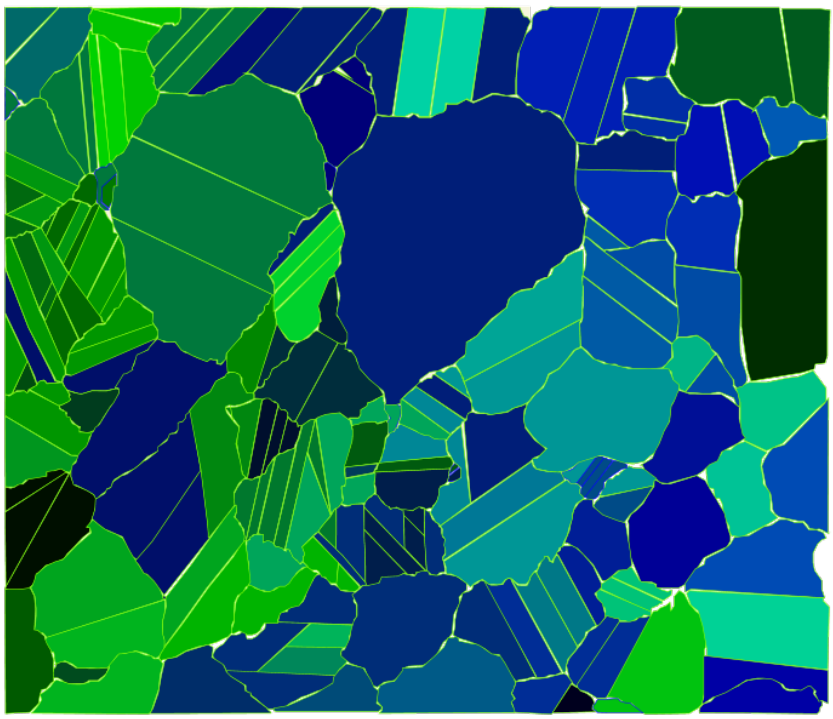
A (4.9.) ábrán látható bal oldali blob konkavitása: 0.009, a nagyobbé: 0.008. A keletkező szuperblobé: 0.078. Az arányok megfelelnek annak, amit szemre várhatunk.

4.2. Futtatási eredmények valós vékonycsiszolati képen

Ebben a fejezetben egy valós vékonycsiszolati képen teszteltem az algoritmust. A kép túlszegmentálását kézzel végeztem. Az eredeti és túlszegmentált kép a (4.11.) képen látható.



4.11. ábra. *Baloldalt az eredeti kép látható. A jobb oldalt pedig a kézi szegmentálás eredménye, ami az algoritmus bemenete.*



4.12. ábra. *A futtatás eredménye valódi képen.*

4.3. Konklúzió

Látható a 4.12. ábrán, hogy több olyan rész is megjelent, ami a megfelelő összeolvadásokat mutatja. A pontosság: 2.11 (3.6). Az algoritmus energiatagjai külön–külön nagyon jól teljesítenek. Viszont együtt nagy méretű ábrán kicsit kevésbé. A valós ábrán is sikerült elég jó eredményt elérni, de az is látszik, hogy bizonyos szituációk még félrevezetik az algoritmust. Ezeknek a kezelése még további fejlesztéseket igényel.

Összefoglalás

A dolgozat célja az volt, hogy bemutasson egy Monte Carlo alapú energiamódszert márványok vékonycsiszolati képeinek szegmentálására ikerkristályok jelenlétében. A szegmentálás eredménye használható különböző hasznos statisztikai következtetések levonására márványok mintákról.
A szegmentálás folyamata két fázisú:

1. Egy másik, intenzitás viszonyokon alapuló algoritmussal túlszegmentáljuk a képet.
2. Az RJMCMC algoritmussal a túlszegmentált kép darabjait csoportokba rendezzük, úgy, hogy a csoportok pontosan fedjék a valódi szemcséket.

Az RJMCMC algoritmusnak (Reversible-jumped Markov Chain Monte Carlo) két alappillére van. Az első egy energiafüggvény, míg a másik az RJMCMC optimalizációs eljárás. Az energiafüggvény szerepe, hogy jellemezze a csoportosítást (konfigurációt). Minnél jobb a konfiguráció, annál alacsonyabb az energia. Az energiafüggvény jól definiált, ha a globális minimuma annál a konfigurációnál van, amelyben a csoportok egyeznek a valós szemcsékkel. Az RJMCMC optimalizációs eljárás a konfigurációt optimalizálja az energiafüggvény alapján.
Az energiafüggvény három energiatag összegeként áll elő. Mindegyik energiatag egy konkrét feladatot lát el:

1. SeedPoint energiatag: A felhasználó jelöléseket (seed pointokat) tesz minden szemcsébe. Minden csoportban csak egy seed point lehet. Az ettől való eltérésre épül az energiatag.
2. StraightLine energiatag: Az ikerkristályok több kisebb szemcse összenövésével keletkeznek. Az összenövések helyén többnyire egyenes vonalak keletkeznek, amik nem részei a szemcse határvonalának. Az egyenes vonalak azonosítására, figyelembe vételére szolgál ez az energiatag.
3. Convexity energiatag: A blobok többnyire konvex formák. Az ettől való eltérést vizsgálja ez az energiatag.

Az RJMCMC algoritmus a konfigurációk változtatását elemi átemenetek segítségével valósítja meg. Minden lépésben megvizsgálja ezek közül a lehetségeseket és végrehajtásukkal járó energiaváltozások szerint egy valószínűségi eloszlást rendel hozzájuk (Gibbs–eloszlás szerint). A végelgesítendő átmenetet ez alapján az eloszlás alapján sorsolja.

Az RJMCMC algoritmus le lett tesztelve mesterséges ábrákon és egy igazi vékonycsiszolati képen. A mesterséges ábrák egyszerű eseteket szimuláltak, amikre a válasz egyértelmű, így az energiatagok hatásai ellenőrizhetők. A mesterséges ábrákon a tesztek sikeresek voltak, az elvárt működést mutatták. A valós ábrán is biztató szegmentálást adott, de látható volt több olyan eset a képen, ahol az algoritmus nem az elvárt hatást érte el. Ezek kisküszöböléséhez további erőfeszítéseket kell tenni.

Köszönetnyilvánítás

Szeretném megköszönni Dr. Csorba Kristófnak a rengeteg segítséget és tanácsot az elmúlt évben, amelyek hozzájárultak a tudományos élet mélyebb megismeréséhez és ezen dolgozat megszületéséhez. Szeretném megköszönni a szüleim támogatását, akik stabil, megbízható hátteret adtak a számomra.

Irodalomjegyzék

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[2] M. Jungmann et al. Segmentation of thin section images for grain size analysis using region competition and edge-weighted region merging. *Computers and Geosciences*, 72:33–48, 2014.

[3] Oliver Krammer et al. Solder paste scooping detection by multi-level visual inspection of printed circuit boards. *IEEE Trans. on Industrial Electronics*, 60(6):2318 – 2331, 2013.

[4] Y. LeCun et al. A tutorial on energy-based learning. *MIT Press*, 2006.

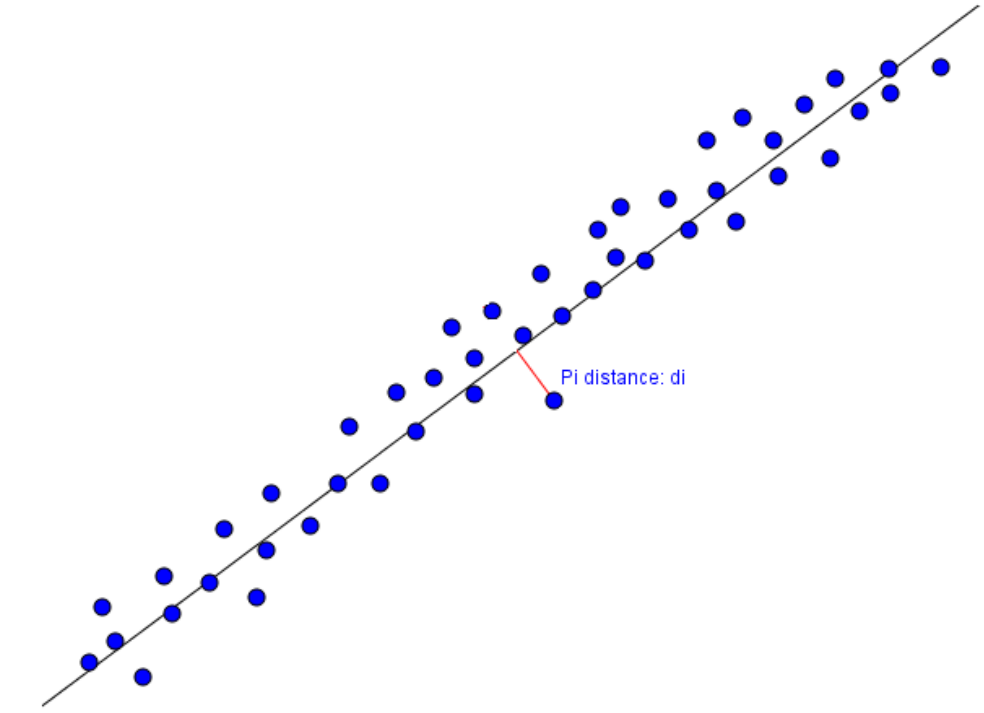
[5] Abdrea Gesmundo. Reversible-jump markov chain monte carlo. *University of Geneva.*

[6] Balazs Szekely Kristof Csorba, Lilla Barancsuk and Judit Zoldfoldi. A supervised grain boundary extraction tool supported by image processing and pattern recognition. *ASMOSIA XI. International Conference*, 2015.

[7] Jovisa Zunic and Paul L. Rosin. A new convexity measure for polygons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(7):923 – 933, 2004.

Függelék

F.1. Az egyenes „curveness” értékének kiszámítási módjáról



F.1.1. ábra. *A pontokra illesztett egyenes és az egyenestől vett távolsága a pontoknak.*

A StraightLine energiatag (2.2.2) számára fontos egy olyan mennyiség, ami jellemezni tudja, hogy egy görbe mennyire görbe (curveness).

Definíció: A curveness (ζ) egyenlő a görbét alkotó pontok (a képen levő görbe pixelekből áll) egy adott egyenestől való négyzetes távolságaik átlagának gyökével, ha az egyenes úgy lett megválasztva, hogy arra ez az érték kisebb bármely más egyenes esetén kapotthoz képest.

Ez a definíció biztosítja, hogy a kapott mennyiség koordináta-rendszer független legyen. Formálisan:

$$\zeta = \min_{\vec{n}} \sqrt{\frac{1}{N} \sum_{i}^{N} \left(d(p_i, \vec{n}) - \tilde{d} \right)^2} \tag{F.1.1}$$

Ahol $d(p_i, \vec{n})$ a p_i pontnak az $\vec{n}$ normálvektorú egyenestől mért távolsága, valamint a $\tilde{d} = \sum_i^N (\vec{r}_i - \vec{r}_0)\, \vec{n}$. Ha az egyenest megfelelő pontba tesszük, ahol a pontok egyenestől való távolságának az átlaga nulla, azaz $\tilde{d} = 0$, akkor a fenti formula egyszerűsödik. Az egyenes vektoros egyenlete alapján ehhez az egyenes pontja kell legyen $\vec{r}_0$, amelyre:

$$\tilde{d} = 0 = \sum_{i}^{N} (\vec{r}_i - \vec{r}_0)\, \vec{n} \Rightarrow_{\forall \vec{n}} \vec{r}_0 = \frac{1}{N} \sum_{i}^{N} \vec{r}_i \tag{F.1.2}$$

A továbbiakhoz a következő jelölések lesznek használva:

$$\vec{n} = (\alpha, \beta),\ \alpha^2 + \beta^2 = 1\ (normalt)$$

$$\vec{r}_i - \vec{r}_0 = (x_i, y_i)$$

A minimum helye szempontjából ekvivalens, ha a következő függvény minimumát keressük α szerint a megadott kényszerfeltétel mellett:

$$F = \sum_i d_i^2 = \sum_i \left[(\vec{r}_i - \vec{r}_0)\, \vec{n} \right]^2 = \sum_i (x_i \alpha + y_i \beta)^2\ valamint\ \alpha^2 + \beta^2 = 1 \tag{F.1.3}$$

A négyzetre emelést elvégezve:

$$F = A\alpha^2 + B\beta^2 + \Theta\alpha\beta \tag{F.1.4}$$

Ahol:

$$A = \sum_i x_i^2,\ B = \sum_i y_i^2,\ \Theta = \sum_i 2x_i y_i$$

Szélsőérték lehet ott, ahol az első derivált nulla:

$$\frac{dF}{d\alpha} = 0 = 2A\alpha + 2B\beta\beta' + \Theta(\beta + \alpha\beta') \tag{F.1.5}$$

A kényszer miatt:

$$\alpha^2 + \beta^2 = 1 \Rightarrow \beta^2 = 1 - \alpha^2 \Rightarrow 2\beta\beta' = -2\alpha$$

Ha a $\beta \neq 0$, akkor igaz a következő is:

$$\frac{dF}{d\alpha} = 2(A - B)\alpha + \Theta \left(\frac{1 - 2\alpha^2}{\beta} \right) = 0$$

Átrendezés és négyzetre emelés után α^2-re másodfokú egyenlet adódik:

$$\underbrace{4(B-A)^2}_{\Gamma^2}\alpha^2(1-\alpha^2) = \Theta^2(1-2\alpha^2)^2 \tag{F.1.6}$$

A megoldóképletet használva:

$$\alpha^2 = \frac{(4\Theta^2+\Gamma^2) \pm \sqrt{(4\Theta^2+\Gamma^2)^2 - 4(4\Theta^2+\Gamma^2)\Theta^2}}{2(4\Theta^2+\Gamma^2)} \tag{F.1.7}$$

Ez jelentősen egyszerűsíthető:

$$\alpha^2 = \frac{1}{2} \pm \frac{1}{2}\sqrt{\frac{\Gamma^2}{4\Theta^2+\Gamma^2}} \tag{F.1.8}$$

A curveness szempontjából a megoldás azon α lesz, amire az F minimális. Az F biztosan nem negatív a definíciója miatt. Az α és a β előjele nem ismert, de mivel a minimumot keressük és A, B biztosan nem negatívak, így a $\Theta\alpha\beta$ szorzatnak negatívnak kell lennie. Tehát α és β szorzata megmondható, ha Θ előjelét megnézzük. Ekkor csak két lényegében különböző eset van, attól függően, hogy melyik megoldást választjuk. A programban egyszerűbb volt megnézni, hogy melyik a kisebb a második derivált vizsgálata helyett. Tehát a két esetet behelyettesítve az F képletébe, a megfelelő F megadható. A hely (α, β) nem szükséges. Végül a keresett érték a megfelelő F ($\tilde{F}$) megtalálását követően:

$$\zeta = \sqrt{\frac{1}{N}\tilde{F}} \tag{F.1.9}$$