# Óbudai Egyetem

Doktori (PhD) értekezés

## Adatpárhuzamos sejtmagkeresési eljárás fejlesztése és paramétereinek optimalizálása

Szénási Sándor

Témavezető:
Vámossy Zoltán, PhD

**Alkalmazott Informatikai Doktori Iskola**

Budapest, 2013. május 30.

## Szigorlati bizottság:

Hermann Gyula, egyetemi docens, CSc
Sergyán Szabolcs, egyetemi docens, PhD
Imreh Csanád, egyetemi docens, PhD

## Nyilvános védés teljes bizottsága:

Galántai Aurél, egyetemi tanár, DSc
Sima Dezső, egyetemi tanár, DSc
Fullér Róbert, egyetemi tanár, CSc
Rövid András, egyetemi docens, PhD
Sergyán Szabolcs, egyetemi docens, PhD
Kovács László, egyetemi docens, PhD
Kovács Szilveszter, egyetemi docens, PhD

## Nyilvános védés időpontja:

# TARTALOMJEGYZÉK

# KÖSZÖNETNYILVÁNÍTÁS

Ezúton mondok köszönetet témavezetőmnek, Dr. Vámossy Zoltánnak eredményeim elérésében, értekezésem elkészítésében nyújtott magas színvonalú, folyamatos és áldozatos segítségéért.

Köszönöm az Alkalmazott Informatikai Doktori Iskola tagjainak, különösen vezetőjének, Prof. Dr. Galántai Aurélnak a hasznos szakmai tanácsokat és a kollegiális segítséget.

Külön szeretném megköszönni a segítségét Dr. Kozlovszky Miklósnak, akinek nagy szerepe volt a témaválasztásban, illetve a kutatás megkezdésében, és Dr. Molnár Bélának, aki a rendelkezésemre bocsátotta a 3DHistech Kft. által a témában már elért eredményeit.

Az Óbudai Egyetemen dolgozó közvetlen kollégáimnak - különösen Dr. Tick Józsefnek, Prof. Dr. Sima Dezsőnek, Dr. Sergyán Szabolcsnak - köszönöm a támogatást, a kitartó ösztönzést értekezésem elkészítése során.

Köszönöm családomnak a megértést, bíztatást és az áldozatvállalást, amivel az értekezésem elkészítését lehetővé tették.

# BEVEZETÉS

Az orvosi célú digitális képfeldolgozás használata napjainkban egyre elterjedtebb a patológusok körében. A legújabb rendszerekben elérhetővé vált, hogy a szövettani vizsgálatok által igényelt lépések nagy része automatizálható (festések, tárgylemezek továbbítása, digitális felvételek készítése stb.), és várhatóan ez a trend a jövőben is folytatódik [**1**]. Az újszerű eszközök a szövetminták feldolgozásának közvetlen előnyei mellett (nagyfelbontású, jó minőségű, jól fókuszált képek) számos új lehetőség előtt nyitották meg az utat, a telepatológiai rendszerekben ugyanis nincs szükség többé a tárgylemezek fizikai továbbítására, hanem hálózaton keresztül is van lehetőség a nagyfelbontású felvételek katalogizálására, megosztására, reprodukciójára, távoli elérésére. Az így kiépült rendszerek elterjedése, a hardvereszközök teljesítményének növekedése, és a képfeldolgozási eljárások fejlődése együttesen készítette elő a következő lépést, amely a képfeldolgozó eljárások megjelenését jelenti a napi rutin diagnosztikában.

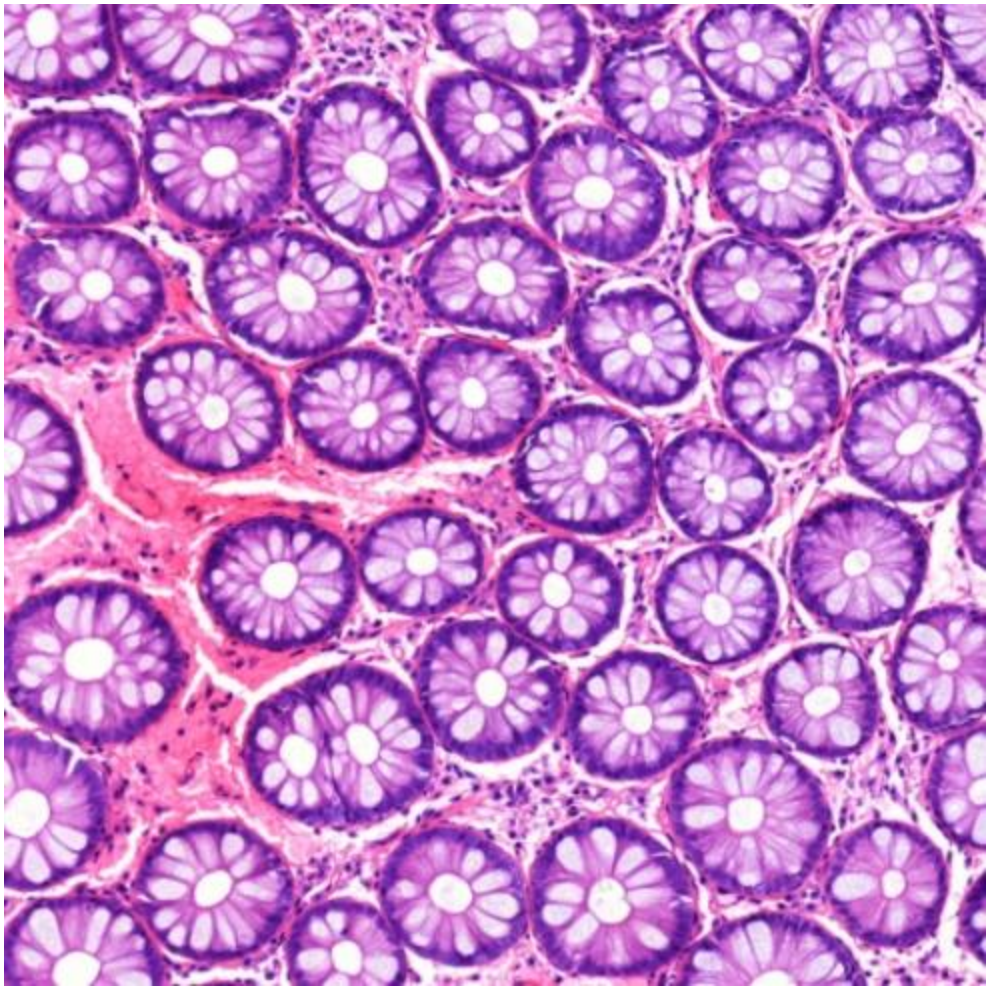
Ezek egy része pusztán a képalkotási eljárások továbbfejlesztésének tekinthető (különféle képjavítási algoritmusok [**2**], CT adatok értékelése [**3**], 3D rekonstrukciók), azonban megjelentek azok a speciális diagnosztikai szoftverek is, amelyek a szövetminták tartalmi analízisére is vállalkoznak. Ez utóbbi esetben az első lépés egy diagnózis felállítása felé általában a különféle szöveti komponensek detektálása. Erre a célra egyelőre nem találhatunk mindenre kiterjedő, általánosan használható módszereket, például az általam részletesebben vizsgált vastagbél szövetminták esetén is számos, egymástól jelentősen módszert találhatunk a sejtmagok keresésére [**4**][**5**][**6**][**7**][**8**][**9**][**10**], illetve a további szöveti komponensek elkülönítésére (mirigyek, felszíni hám). A különféle szegmentálási eredmények felhasználhatók akár közvetlenül, a képernyőn megjelenítve különféle morfológiai és morfometriai paramétereket, vagy akár a közeljövőben lehetőség nyílhat ezeken keresztül egy részben, vagy akár teljesen automatikus diagnosztizáló rendszerek fejlesztése felé [**11**][**12**].

A kidolgozott különféle eljárások egymástól jelentősen eltérő alapelvek alapján működnek. Csak a sejtmagok detektálását végző algoritmusok között is találhatunk olyan, a K-közép [**13**] eljáráson alapuló módszert, amely pusztán a képernyőn elhelyezkedő pixelek színeire, illetve azok eloszlására épít, így próbálva meg különféle osztályokat kialakítani, amelyekből az egyikbe remélhetőleg csak a sejtmagokat alkotó pixelek kerülnek. De emellett találhatunk jóval összetettebb, a pixelek színeit és elhelyezkedését is figyelembe vevő régiónövelési [**14**] módszereket, amelyek ugyan jóval nagyobb futásidő mellett, de valamivel pontosabban tudják meghatározni az egyes sejtmagok pontos elhelyezkedését.

A szoftveres környezet változásán túlmenően az utóbbi években jelentősen megváltozott a hardver lehetőségek tárháza is. Az informatikában egészen a 2000-es évekig megszokottá vált, hogy a különböző processzorok teljesítménye évről-évre folyamatosan növekszik, miként ezt a közkedvelt Moore-törvény előre meg is jósolta. Ez a nagyon egyszerű jóslat hosszú éveken keresztül időtállónak bizonyult, napjainkban azonban ez a dinamikus fejlődés megtorpant, vagy legalábbis jelentősen irányt változtatott [**15**]. A processzorgyártók kénytelenek voltak szembesülni azzal, hogy nem tudják az órajelfrekvenciákat az eredetileg elképzelt ütem szerint növelni tovább, és emiatt új megoldásokat kellett keresni a további fejlesztések számára. Ezek közül az utóbbi években a legszembetűnőbb a többmagos processzorok megjelenése [**16**], amelyek az évenkénti teljesítménynövekedést egy meglehetősen kézenfekvő ötlettel biztosítják: egy processzoron belül bizonyos egységeket egyszerűen megdupláztak (négyszereztek, nyolcszoroztak), amelynek segítségével az eszköz elméleti számítási kapacitása továbbra is növekszik, kielégítve a piac igényeit.

A gyakorlatban azonban ez a szoftverfejlesztés számára egy drasztikus szemléletmód váltást igényelt. A szoftverfejlesztők ugyanis megszokhatták, hogy a nagy teljesítményigényű rendszerek esetében az optimalizálás mellett az időre is bátran számíthattak, hiszen a későbbiekben megjelenő hardver eszközökön futtatva az alkalmazásokat, azok gyakorlatilag automatikusan gyorsabban működtek. Napjainkban ez azonban már nem mondható ki ennyire egyértelműen, a többprocesszoros rendszerek maximális teljesítménye ugyanis csak az összes processzor(mag) egyidejű terhelésével, tehát párhuzamos algoritmusokkal aknázható ki. Emiatt nagyon sok hagyományos módszert érdemes optimalizálni, vagy ha ez nem lehetséges, akkor teljesen egészében újragondolni, újratervezni a mai kor igényeinek megfelelően.

A többmagos processzorok megjelenésén túlmenően azonban időközben megjelentek egészen újszerű architektúrák is, mint például az általános célú számításokra használható grafikus kártyák (GPGPU-k). Az eredetileg a monitoron látható kép megjelenítésére kialakított hardvereszközök idővel különféle 3D számítások végrehajtására lettek alkalmasak, az ipari (és nem kevésbé az otthoni játékos) felhasználók igényeinek megfelelően egyre több egyszerű végrehajtóegységgel rendelkeztek; majd különféle *shader* generációváltások [**17**] után felmerült a lehetőség, hogy általános célú programok futtatására is alkalmasak legyenek. Ennek adott nagy lökést az Nvidia 2007-es lépése, amikor kiadott egy szoftverfejlesztési környezetet is a saját kártyáihoz (*CUDA 1.0*).



1**. ábra:** *Hematoxilin-eosin festésű vastagbél szövetminta.*

A szoftverfejlesztők számára egy teljesen új világ nyílt meg a több száz magot tartalmazó, ám meglehetősen egyszerű, és sok korlátot tartalmazó rendszerek programozásának lehetőségével. A nyers teljesítményt tekintve ezek az eszközök már évekkel ezelőtt is sokszorosan meghaladták az aktuális csúcs CPU-k feldolgozási kapacitását, így meglehetősen sok, nagy számításigénnyel rendelkező alkalmazás ügyében folynak kutatások azok GPGPU-n való alkalmazhatóságáról [**18**][**19**][**20**]. A fejlesztések azonban meglehetősen nehézkesen haladnak, ugyanis az újszerű eszköz számos korláttal és speciális jellemzővel bír a hagyományos CPU-khoz viszonyítva, továbbá hiányoznak a kiforrott eszközök, és a több éves tapasztalatok is, amelyek a hagyományos fejlesztéseknél már régóta rendelkezésre állnak.

Az általam elsődlegesen vizsgált hematoxilin-eozin festésű vastagbél szövetmintákban (**1.** ábra) a sejtmagok megbízható detektálása tűnik az egyik legkritikusabb pontnak a későbbi automatikus diagnosztikai rendszerek felé vezető úton. Egyrészt ezek a szöveti komponensek már önmagukban is számos információt tartalmazhatnak (elhelyezkedésük, méretük,

mennyiségük) a későbbi diagnózishoz, másrészt számos további komponenseket (mirigyek, hámsejtek) azonosító eljárás létezik, amelyek a már előzőleg azonosított sejtmagokra alapozzák a további feldolgozást.

Emiatt mindenképpen lényeges, hogy rendelkezésre álljanak olyan módszerek, amelyek mind a pontosság, mind pedig a sebesség tekintetében megfelelnek a gyakorlati felhasználás követelményeinek. A különböző eljárások értékelésekor a sebesség mérése nem jelent különösebb problémát, a pontosság tekintetében azonban már meglehetősen nehéz összehasonlítani két, egymástól teljesen különböző alapelvek szerint működő algoritmust.

Célom emiatt, hogy kidolgozzak egy mutatót, amely megmutatja, hogy egy adott sejtmagkeresési eljárás milyen pontos eredményt adott egy teszt képen (amennyiben ismert annak a referencia eredménye is). Mivel maga a mutató kiértékelése is meglehetősen számításigényes művelet lehet (a nagyfelbontású képek több ezer sejtmagot is tartalmazhatnak akár), emiatt megtervezek egy olyan módszert, ami a lehetőségekhez képest minél kevesebb lépéssel tudja visszaadni ezt az értéket nagyméretű minták esetében. Ez a mutató alkalmas kell, hogy legyen a már meglévő módszerek összehasonlítására, illetve a későbbiekben kifejlesztésre kerülő új módszerek előnyeinek objektív vizsgálatára is.

Az már az előzetes vizsgálatok alapján is nyilvánvaló, hogy nem lesz egy tökéletes módszer a sejtmagok keresésére, általában találkozhatunk kisebb pontosságot nyújtó gyors eljárásokkal, illetve nagyobb pontosságot nyújtó lassabb módszerekkel. Utóbbinak részben az az oka, hogy a különféle eljárások gyakran még a hagyományos képszegmentálási módszereken alapulnak, amelyek gyakran nem használják ki a napjainkban jellemző többprocesszoros rendszerek lehetőségeit, különösen nem az elmúlt években megjelent, legnagyobb teljesítményt rejtő GPGPU-két.

Célkitűzésem, hogy kidolgozok egy olyan adatpárhuzamos sejtmagkeresési eljárást, amelynek pontossága legalább azonos a napjainkban rendelkezésre álló egyik legjobbnak tekinthető régiónövelési módszerrel, mindezt az eredményt azonban rövidebb futásidő alatt adja vissza. A régiónövelési eljárás két fő részből áll, magából a régiónövelési folyamatból, illetve az ehhez szükséges kiinduló (*seed*) pontok kereséséből. Célom egy olyan, a GPGPU-k speciális követelményeihez is alkalmazkodó új, adatpárhuzamos módszer kidolgozása, amely mindkét területen minél jobban kihasználja az eszközök rendelkezésre álló számítási kapacitását.

A különféle képszegmentálási eljárások egészen másféle paramétereket igényelnek, de általában elmondható, hogy bármilyen jó is a kidolgozott módszer, csak a megfelelő paraméterkészlet segítségével lehet elérni annak maximális pontosságát. A lehetséges paraméterek száma azonban gyakran olyan nagy, hogy az ideális paraméterkészlet megtalálása manuálisan reménytelen feladat.

Célom, hogy kidolgozzak egy olyan evolúciós alapokon nyugvó módszert, amely alkalmas arra, hogy nagyméretű paramétertér esetén is képes ajánlani egy, a gyakorlatban jól használható paraméterkészletet a különféle sejtmagkeresési eljárások számára. A genetikus algoritmusok meglehetősen számításigényesek, így elkészítem egy elosztott rendszer terveit és implementációját is, hogy annak segítségével találjak egy ideális paraméterkészletet a régiónövelési algoritmus számára.

# 1. SZÖVETMINTA SZEGMENTÁLÓ ALGORITMUSOK ÖSSZEHASONLÍTÓ VIZSGÁLATA

## 1.1. Szegmentálási algoritmusok értékelési módszereinek áttekintése

A képszegmentálás az egyik legkritikusabb képfeldolgozási feladat. Célja elsődlegesen a bemeneti kép részekre bontása, majd pedig ezen részek azonosítása. Szöveti képek analízisére számos módszer létezik, ha csak egy speciális részterületet tekintünk, például a vastagbél szövetekben a sejtmagok detektálását, akkor is számos publikációt találhatunk, amelyek mind egy újfajta módszert írnak le [**4**][**21**][**5**][**6**][**22**][**7**][**8**][**9**][**10**]. Meglehetősen nehéz összehasonlítani ezeket a módszereket, hiszen egészen különböző utakon próbálják meg elérni ugyanazt az eredményt, azonban az objektív döntések érdekében szükség van valamilyen jól értelmezhető mérőszámokra, amik ezekben az esetekben biztosítanak egy konzekvens, könnyen kezelhető értéket.

Ez különösen fontos lehet abban az esetben, ha a kutatás célja egy már meglévő módszer továbbfejlesztése, vagy már meglévő módszereken alapuló újszerű eljárás kidolgozása, hiszen csak ilyen mérőszámok segítségével lehet ténylegesen ellenőrizni a kutatás eredményességét. Egy algoritmus jóságát számos szempontból lehetne értelmezni, feladatunk szempontjából ezek közül a pontossággal és a futásidővel érdemes részletesebben foglalkozni, illetve célszerű lenne olyan jósági függvényt találni, amely könnyen kezelhető és gyorsan kiértékelhető, hogy a későbbiekben végrehajtott automatizált paraméteroptimalizálásnál is jól használható legyen.

Természetesen már most is számos, a képfeldolgozó algoritmusok pontosságát vizsgáló módszer áll rendelkezésre, ezeket a vizsgálati szempontok szerint az alábbi csoportokba sorolhatjuk [**23**]:

- **Egyszerű analitikus (*analytical*):** Az analitikus módszerek közvetlenül magát a szegmentáló algoritmust vizsgálják (alapelvek, követelmények, komplexitás stb.). A gyakorlatban ez a módszer sajnos csak néhány speciális esetben használható eredményesen, ugyanis nincsenek mindenre kiterjedő, általánosan használható elméleti modellek a képfeldolgozás területén.
- **Empirikus jóság (*empirical goodness*):** Az empirikus módszerek mindig azon alapulnak, hogy a vizsgálandó algoritmust teszt képekre alkalmazzák, majd az így kapott eredményt értékelik. Az empirikus jóságot kereső módszerek magát az

eredményt közvetlenül, önmagában próbálják elemezni, általában bizonyos, emberi intuíción alapuló jósági szempontok szerint.

- **Empirikus eltérés (*empirical discrepancy*):** Ezek a módszerek szintén a szegmentálási algoritmus egy teszt képre való alkalmazásának eredményét vizsgálják, azonban lényeges különbség, hogy itt már rendelkezésre áll egy referencia eredmény a helyes megoldással, és a jósági vizsgálat alapja a két eredmény összehasonlítása. További előnye, hogy egyszerűbbnek tűnik egy mérőszámot rendelni az abszolút jóság helyett a jó eredményhez való hasonlóság fokához.

Mivel a vizsgálandó algoritmusok alapvetően szövetmintákon található objektumokat fognak azonosítani, így célszerű az orvosi vizsgálatok módszereit is figyelembe venni. A klinikai gyakorlatban meglehetősen gyakori a „gold standard" teszteken alapuló értékelés [**24**], amely a fenti csoportok közül az empirikus eltérésen alapuló módszerekkel egyeztethető össze. Ehhez a feldolgozandó képek mellett szükségünk van egy referencia eredményre („gold standard"), amely az adott feladat jelenleg rendelkezésre álló legjobb megoldását jelenti, ez ebben az esetben (szövetmintákban található sejtmagok azonosítása) képzett patológusok által annotált képeket jelent.

A legáltalánosabb megoldás a két eredmény összehasonlításával felállított igazságmátrixon (*confusion matrix*) [**25**] alapul, ami két lehetséges kimenetet tekintve az alábbi adatokat tartalmazza: *igaz-pozitív*, *igaz-negatív*, *hamis-pozitív*, *hamis-negatív* találatok száma. Ez az orvosi vizsgálatokban is gyakran használatos osztályozás nagyon egyszerűen és hatékonyan alkalmazható képfeldolgozási algoritmusok vizsgálatakor is, amennyiben ezeket az alábbiak szerint értelmezzük:

- *Igaz-pozitív (true-positive, későbbiekben TP):* Mind a referencia, mind pedig a teszt eredményben helyesen sejtmaghoz tartozónak jelölt pixel.
- *Igaz-negatív (true-negative, későbbiekben TN):* Mind a referencia, mind pedig a teszt eredményben helyesen sejtmaghoz nem tartozónak jelölt pixel.
- *Hamis-pozitív (false-positive, későbbiekben FP):* A referencia eredményben nem, a teszt eredményben azonban hibásan sejtmaghoz tartozónak jelölt pixel.
- *Hamis-negatív (false-negative, későbbiekben FN): A* referencia eredményben sejtmaghoz tartozónak, a teszt eredményben azonban hibásan nem annak jelölt pixel.

Itt referencia eredmény alatt az orvosok által annotált, teszt eredmény alatt pedig a vizsgált algoritmus által adott végeredményt értjük. A mérőszám értelmezhető a teljes vizsgált képre,

de akár páronként pixelcsoportok összehasonlítása esetében is. Mivel egyelőre csak a sejtmagok megkeresése a feladunk, így nincs szükség ennél több osztály felállítására.

A *pontosság* (*accuracy*) ezek alapján már egyszerűen származtatható mérőszám (a pozitív találatok és az összes találat aránya) [**25**]:

$$\text{Pontosság} = (TP + TN) / (TP + TN + FP + FN) \quad (1)$$

Illetve gyakran szükség lehet a *precizitás* (*precision*) és *felidézés* (*recall*) értékére [**25**]:

$$\text{Precizitás} = TP / (TP + FP) \quad (2)$$

$$\text{Felidézés} = TP / (TP + FN) \quad (3)$$

Az így kapott értékek szemléletesek (pl. a pontosság esetében a 100% azt jelenti, hogy az algoritmus pontosan ugyanazt az eredményt adta, mint a referencia eredmény, a 0% pedig azt, hogy egy pixelt sem azonosított helyesen), illetve nincs szükség az így kapott eredmények utólagos normalizálására sem, az értékek mindig egy jól behatárolható tartományon belül lesznek, így azok összehasonlítása is egyszerűsödik.

Számos további módszert találhatunk, amelyekkel megpróbálhatjuk tovább finomítani az eredményt, pl. a hibásan osztályozott pixelek esetén érdemes lehet figyelembe venni azt is, hogy az így tévesen azonosított pixel milyen messzire esik a legközelebbi, valóban ebbe az osztályba tartozó pixelhez [**23**]. Így ennek megfelelően két hibásan detektált pixelt nem feltétlenül kell azonos súlyú hibának tekinteni (ami további származtatott értékek esetén felveti a fuzzy rendszerek használatának igényét [**26**][**27**]).

Eddig azonban csak pusztán a teszt és referencia képek pixelenkénti összehasonlításából indultunk ki, és önmagában ez az egyszerűsített értékelés nem ad mindig kielégítő eredményt. A szegmentálás során ugyanis általában nem csak az a kérdés, hogy a képernyő egy pixele megadott típusú objektumhoz tartozik-e, hanem az egyes objektumokat magukat kell azonosítani.

Ugyanis gyakran célszerű megvizsgálni a detektált objektumok számát, hogy ez mennyiben különbözik a referencia és a saját algoritmusunk által adott eredmény között (mivel a diagnosztika szempontjából jelentőséggel bírhat, hogy megadott területen mekkora a sejtmagok száma, illetve az ebből származtatható sűrűség), illetve ezt célszerű lehet tovább finomítani, hogy az egyszerű megszámláláson túlmenően vegyük figyelembe az egyes

detektált objektumok különböző geometriai jellemzőit is (mivel a sejtmagok alakjából is lényeges következtetéseket vonhatunk le a későbbiekben, illetve ez segíthet a további szöveti komponensek azonosításában). Ezek a geometriai jellemzők az általunk vizsgált sejtmagok esetében célszerűen az alábbiak: középpont helyzete, sejtmag területe, sejtmag átmérője (minimális illetve maximális), sejtmag pixelenkénti pontos elhelyezkedése [**28**].

A végső értékelés szempontjából tehát mind a pixelszintű, mind pedig az objektumszintű összehasonlítás értékes eredményeket szolgáltathat. Természetesen a két módszer nem zárja ki egymást, sőt, célszerű lehet egy, a fentieket egyaránt figyelembevevő aggregált jósági függvény használata [**29**], ahol külön-külön megmérjük az egyes szempontok szerinti eredményt, majd ezeket megfelelő súlyozás és normalizálás mellett összesítjük. Ez a megoldás a későbbiekben amiatt is hatékony lehet, mivel így az eddigiektől jelentősen eltérő szempontokat is figyelembe vehetünk, például az algoritmus jóságát befolyásolhatja a feldolgozó program futásideje, ami bár a végeredményben nem látszódik, de mégis nagyban befolyásolja a módszer gyakorlati használhatóságát. A megfelelő súlyok és arányok megtalálása persze meglehetősen nehéz lehet, sőt, erre előre nem is tudunk egyértelmű értékeket meghatározni, hiszen mindig az adott feladat, illetve a rendelkezésre álló erőforrások (hardver háttér, emberi személyzet, rendelkezésre álló idő) döntik el, hogy az egyes részeredményeket (pontosság, sebesség) milyen súllyal kell figyelembe vennünk.

## 1.2. Megfelelő mérőszám kialakítása

### *1.2.1. Sejtmagkeresési eljárások értékelési módszerei*

A mikroszkopikus szöveti képek feldolgozásának területén kulcsfontosságú lépés a szöveti metszeteken a sejtmagok detektálása, azok tulajdonságainak meghatározása, illetve a további képfeldolgozási műveletek számára azok elhelyezkedésének analízise. Amennyiben képfeldolgozó algoritmusokat próbálunk értékelni, szükséges lehet annak vizsgálata, hogy az algoritmusok eredményét pontosan mire kell majd felhasználni, azok milyen további feldolgozásokon esnek majd át. Értelemszerűen az algoritmusok értékelésekor azokat kell előnyben részesíteni, amelyek a gyakorlat szempontjából is lényeges területeken nyújtanak jó eredményeket, ebben az esetben ehhez először meg kell vizsgálnunk azt, hogy a klinikai vizsgálatok során a képfeldolgozó algoritmustól pontosan milyen eredményeket várunk.

A genom sérülése esetén, amennyiben a kromoszómák helytelenül rendeződnek, létrejöhetnek sejtek extra kromoszómákkal. Ez bizonyos esetekben hatalmas méretű sejtmagokban jelentkezik, majd végül, mivel a sejtmagban nincs hely az extra kromoszómák számára, maga

a membrán is deformálódik. Általános szabály, hogy minél több a bizarr méretű sejtmag, annál agresszívabb a betegség [**30**]. Ennek megállapításához viszont szükség van az egyes sejtmagok méretéről pontos információkra, ennek megfelelően a képfeldolgozó algoritmusok értékelésekor ennek a képességnek nagy jelentőséget kell tulajdonítanunk.

A sejtmagok formája szintén fontos információkat rejthet. Bizonyos speciális elváltozásokat kivéve a sejtmagok általában körszerűek, betegségek esetén ez azonban megváltozhat, aminek a detektálása a diagnózis szempontjából kritikus lehet. Minél inkább változatosak a mintában megtalálható sejtmagok formailag, annál kedvezőtlenebb lehet a diagnózis [**30**]. Tehát lényeges, hogy az egyszerű méret és elhelyezkedés adatokon túlmenően előnyben részesítsük azokat a szegmentáló algoritmusokat, amelyek az egyes sejtmagok pontos alakjáról is nagy pontossággal adnak információkat.

Emellett lényeges a sejtmagok egymáshoz viszonyított elhelyezkedése is. Szövetmintától függően a sejtmagok általában egy előre várható sűrűséggel fognak megjelenni az egészséges szövetek esetében. Azonban a különféle elváltozások megjelenésének hatására ez megváltozhat, ami szintén fontos információ lehet a későbbi diagnózis számára. Például a prosztatarák esetében „*gleason grade*” korrelál többek között a sejtmagok sűrűségével egy megadott területen belül [**31**]. A szegmentáló algoritmusnak tehát az egyes sejtmagokról lehetőség szerint nem csak mint teljesen különálló egyedekről kell információkat nyújtania, hanem mindezt olyan formában kell megtennie, hogy abból a későbbi diagnosztikához szükséges elhelyezkedési adatokat is minél nagyobb pontossággal ki lehessen nyerni.

A fentiekből adódik, hogy a legegyszerűbb, pusztán a referencia és az eredményül kapott képek pixelenkénti összehasonlítása nem felel meg a szükséges ellenőrzési kritériumoknak, hiszen például sok kisméretű sejtmag *hamis-negatív* detektálása esetén ez a módszer csak kis hibát jelezne, míg a sűrűség változása jelentős. Tehát mindenképpen olyan értékelési módszert kell választani, amelyik objektum szinten hasonlítja össze az eredményeket.

A sejtmagok szegmentálása nem csak az azokból közvetlenül származtatható adatok miatt érdekes (darabszám, méret stb.). Számos szegmentálási algoritmusnak a sejtmagok meghatározása pusztán az első lépés, hogy a további lépésekben már erre az információra támaszkodva próbálják meg a további komponenseket elkülöníteni. A vastagbél szövetek vizsgálatakor fontos további komponensek, a mirigyek, kehelysejtek és a hámsejtek detektálása is jelentősen javítható abban az esetben, ha pontos információkkal bírunk a szövetben található sejtmagok helyzetéről és alakjáról. Például a mirigyek detektálásának

egyik módszere azon alapul, hogy azok körvonalán általában az egymáshoz közeli sejtmagok egy láncot alkotnak, amelyeket a megfelelő szűrők alkalmazásával [**32**] összekapcsolhatunk, ezzel kirajzolva magát a mirigyet. De ehhez hasonlóan a hámsejtek keresése során is nagy segítséget nyújt az azt alkotó sejtmagok pontos detektálása.

Az előzőekből jól látható, hogy az egyes sejtmagokra vonatkozó statisztikai (darabszám, sűrűség stb.) illetve különféle származtatott geometriai adatok (elhelyezkedés, méret stb.) értékelése önmagában szintén nem elengedő, hanem mindenképpen szükség van az egyes sejtmagok pontos alakjának a referencia eredményekkel való összehasonlítására. Tehát a választott értékelési módszernek figyelembe kell vennie az egyes sejtmagok elhelyezkedését és azok pontos alakját, ezt pedig a legpontosabban a teszt és referencia képeken egymásnak megfeleltetett sejtmagok egymás közti pixelenkénti összehasonlításával tudjuk megtenni.

### *1.2.2. Saját mérőszám kialakítása*

A fentiek alapján elmondható, hogy önmagában sem a pusztán statisztikai adatokon alapuló, sem pedig a pixelenkénti összehasonlítást végző kiértékelő algoritmusoktól nem várhatjuk el a számunkra megfelelő eredményt. Az általam kidolgozott mérőszám emiatt a két módszert ötvözve nem csak pixelszintű összehasonlítást végez, hanem első körben megpróbálja egymáshoz rendelni a referencia és a teszt eredmények sejtmagjait, aminek segítségével a kiértékelésben figyelembe tudja venni az egyes sejtmagok elhelyezkedését (és így közvetve azok egymáshoz viszonyított helyzetét, darabszámát stb.). Az egymáshoz rendelés során egy referencia sejtmaghoz csak egy teszt sejtmag tartozhat, és ugyanez fordítva is igaz, egy teszt sejtmag csak egy referencia sejtmaghoz rendelhető. Ez egy meglehetősen szigorú szabály (erősen bünteti azokat az eredményeket, ahol a valóban létező egy darab sejtmagot a szegmentáló algoritmus több kisebb sejtmaggal próbálja lefedni), de jelentősen egyszerűbbé teszi az eredmények értékelését, mivel így nincs szükség a többszörösen figyelembe vett területek miatti korrekciókra.

A sejtmagok egymáshoz rendelését követően következik azok egymáshoz való hasonlóságának vizsgálata. Mivel a további feldolgozáshoz szükség lehet a sejtmagok pontos elhelyezkedésére és alakjára, emiatt nem érdemes az egyes sejtmagok származtatott paramétereit összehasonlítani (tehát például mindkét sejtmag esetén kiszámolni a terület, kerület, átmérő stb. tulajdonságokat, majd ezek összehasonlítása alapján hozni valamilyen döntést), hanem célszerűbb az egyes sejtmagok pixelenkénti összehasonlítását választani, valószínűleg ezzel tudjuk a legpontosabb vizsgálatot elvégezni. Emiatt a teszt és referencia

sejtmagok egymáshoz rendelését követően egy ilyen összehasonlítással próbáljuk értékelni a találatot.

A pixelszintű összehasonlítás már régóta használt és jól bevált technika, közvetlen kimenete célszerűen az alábbi négyes szokott lenni: *igaz-pozitív, igaz-negatív, hamis-pozitív, hamis-negatív* találatok száma. A mérőszámok már ismerősek lehetnek az igazságmátrixnál ismertetettek alapján, azonban lényeges különbség, hogy az összehasonlítás alapja itt most nem két azonos méretű kép, hanem két darab, az előzőekben leírt párosítással egymáshoz rendelt sejtmag, amelyek közül az egyik a referencia eredményből, a másik pedig a teszt eredményből származik. Ennek megfelelően a fenti értékeket célszerű pontosítani, ami a *igaz-pozitív* találatok esetén meglehetősen egyszerű, minden egymást fedő pixel egységnyi súllyal számítható.

A *igaz-negatív* találatokat ebben az esetben nem értelmezhetjük, hiszen szigorúan csak a két sejtmagot alkotó pixelek halmazát hasonlítjuk össze, így értelemszerűen nem találkozhatunk olyan pontokkal, amelyek egyik sejtmagnak sem részei. A későbbiekben viszont újra megjelennek majd ezeket az értékek, a sejtmag párok összehasonlításán túlmenően ugyanis a teljes kép feldolgozására is próbálunk értékelést adni, és itt, az egész kép viszonylatában már értelmezhetjük az *igaz-negatív* találatokat is. Ez tulajdonképpen azon pixelek száma lesz, ahol sem a referencia, sem pedig a teszt eredmény nem talált sejtmagot (mivel a „gold standard” képek esetében csak a teljes kép egy részén lettek manuálisan megjelölve a sejtmagok, így ebben az esetben csak erre a manuálisan annotált területre célszerű kiszámítani).

A *hamis-pozitív* és *hamis-negatív* találatok esetében pedig a már megszokott fogalmakkal dolgozhatunk, azonban itt is célszerű pontosítani a kiszámítás módját. Nem érdemes minden eltérést a *igaz-pozitív* találatoknál használt egységnyi súly használatával értékelni, hanem érdemes figyelembe venni a hibásan detektált pixel távolságát a referencia sejtmagtól. Raszter képek esetén a sejtmagok kontúrja általában nagyon nehezen határozható meg egyértelműen (illetve további jelentős különbségeket jelenthet, hogy a teszt vagy a referencia képen a külső vagy a belső kontúr lett megjelölve), így az ennek a közelében található hibákat célszerű kisebb súllyal számolni.

Mivel a tolerálható távolság nagyban függ a kép felbontásától, illetve a nagyítás fokától, így ezt célszerűen a referencia sejtmag méretének (legnagyobb átmérő) arányában határoztuk meg:

$$\text{Súly}_i = \text{Min}(\text{Min}_j(\text{Táv}(T_i, R_j)) / \text{Átmérő}(R) * KT), 1) \quad (4)$$

Ahol:

- *Táv($T_i$, $R_j$):* A teszt sejtmag i. pixelének és a referencia sejtmag j. pixelének a távolsága.
- *Átmérő(R):* A referencia sejtmag (legnagyobb) átmérője.
- *KT:* Konstans paraméter

A kifejezésben *KT* egy konstans, aminek segítségével meghatározható, hogy a referencia sejtmag átmérőjéhez viszonyítva milyen távolságon túl található pixeleket tekintsünk már egységnyi súlyú hibának (pl. *KT* = 0,5 esetében az átmérő felénél nagyobb távolságnál már így számolunk). Ezen a távolságon belül pedig a távolsággal lineárisan arányosan számoljuk a hiba mértékét. A konstans beállításával lehet meghatározni, hogy mennyire szeretnénk az értékelésnél szigorúan figyelembe venni az eltéréseket, *KT* = 0 esetén például minden *hamis-pozitív* pixel 1 súllyal számít, ezzel könnyen visszatérhetünk a hagyományos számítási metódushoz. Ugyanez a módszer használható mind a *hamis-pozitív*, mind pedig a *hamis-negatív* találatok esetében, ahol ennek megfelelően a végeredmény nem csak a pixelek számát, hanem a fenti módon súlyozott értékek összegét mutatja majd.

Amennyiben a referencia képen található sejtmaghoz nem sikerült a teszt eredményből egyet sem rendelni, vagy éppen fordítva, akkor ezeket az eredményeket egységnyi súllyal tekinthetjük *hamis-pozitív*, illetve *hamis-negatív* pixeleknek.

Az egyes sejtmag párokra, illetve a különálló sejtmagokra a fentiek alapján kiszámolt értékeket összegezve már ki lehet számolni egy egyszerű pontossági értéket (feltételezve, hogy a referencia eredmény legalább egy sejtmagot tartalmazott): (*TP*+*TN)* / (*TP* + TN + *FP* + *FN*). Az így kapott érték 1 lesz abban az esetben, ha az algoritmus pontosan ugyanannyi sejtmagot talált, mint amennyit a referencia eredmény tartalmazott, továbbá az egyes sejtmagok pixelei páronként megegyeznek. Hiányzó, vagy éppen téves találatok esetén ez az érték csökken, illetve 0 lesz abban az esetben, ha egy helyes pixelt se talált.

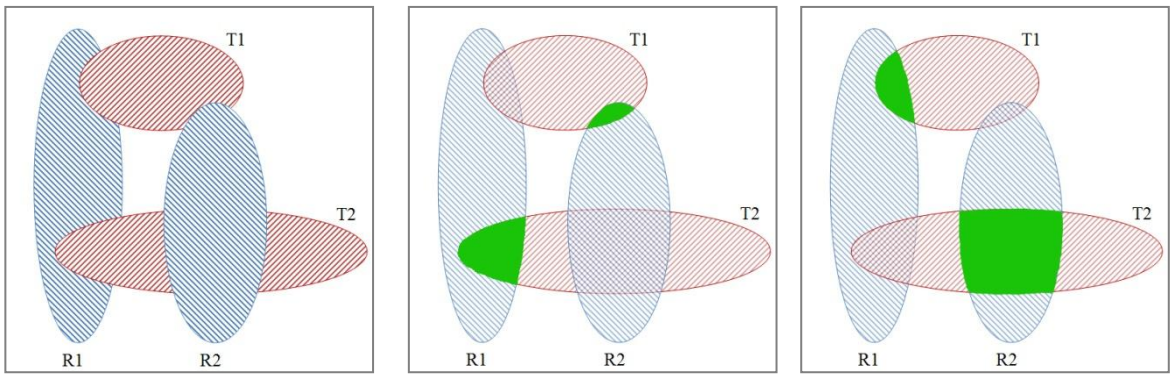
## 1.3. Kiértékelő algoritmus tervezése és implementálása

### *1.3.1. Egymást átfedő sejtmagok csoportokba rendelése*

A fenti értékelési mód egy meglehetősen precíz objektum és pixel alapú összehasonlítást végez, figyelembe véve a sejtmagkeresési feladatok sajátosságait. A kiértékelésnek egy meglehetősen kritikus pontja, hogy miként rendeljük egymáshoz a referencia és a teszt eredményekben található sejtmagokat, hiszen nyilván ez befolyásolja a végeredményt. Mivel az egymást átfedő sejtmagok miatt ez a párosítás gyakran sokféleképpen elvégezhető, emiatt lényeges, hogy a több lehetséges párosítás közül azt vizsgáljuk, amelyik a legmagasabb végső pontszámot határozza meg (globálisan, a teljes képfeldolgozás eredményét tekintve).

A fenti, legjobb eredményre törekvő párosítás implementálását azonban megnehezíti, hogy az meglehetősen számításigényes, egy nagyobb képen ugyanis több ezer sejtmag található, a köztük elképzelhető összes párosítás közül az optimális megállapítása egyszerű lineáris kereséssel [**33**] a gyakorlati felhasználásra alkalmatlan futásidőt eredményezne. Természetesen nincs is szükség az egymástól távol lévő sejtmagok egymáshoz rendelésére, így a ténylegesen vizsgálható elemek száma jóval kisebb lesz, az egymáshoz közeli sejtmagok közül azonban nem lehet mindig egy lépésben eldönteni, hogy melyiket melyikkel célszerű egymáshoz rendelni.

Ahogy az ábrán is látható (2. ábra), ha a sejtmagokat egy mohó algoritmussal próbáljuk egymáshoz rendelni, és az *R1* referencia sejtmag feldolgozása az első lépés, akkor ehhez célszerűen hozzárendeljük az alsó *T2* sejtmagot (mivel ebben az esetben így a legnagyobb az átfedő pixelek száma). Ennek következményeként azonban a következő, *R2* referencia sejtmaghoz már csak a felső *T1* teszt sejtmagot lehet hozzárendelni, és jól látható, hogy az így




a) b) c)

**2. ábra:** Sejtmagok lehetséges párosítása: *(a) kék referencia sejtmagok: R1, R2; piros teszt sejtmagok: T1, T2 (b) első lehetséges párosítás: R1-T2, R2-T1 (c) második párosítás: R1-T1, R2-T2.*

nyert *R1-T2*, *R2-T1* párosítás nem lesz optimális. A másik lehetséges, *R1-T1*, *R2-T2* párosítás esetében nagyobb az egymást fedő pixelek száma, így nagyobb a kiértékelés végeredményeképpen kapott pontossági érték is.

A példából is látható, hogy nem lehet egyszerűen a feldolgozás sorrendjében lokálisan legjobbnak tűnő megoldást választani, hanem az egymást átfedő elemek közötti többi lehetséges párosítást is figyelembe kell venni. A fenti példában ez mindössze 2 eset vizsgálatát jelenti (ami valójában 7, ha megengedjük a szabadon maradó sejtmagokat), de értelemszerűen a lehetséges állapotok száma még nagyobb lenne, ha egy ötödik sejtmag is fedésben van egy, vagy akár több már meglévővel (a gyakorlatban az egymást fedő sejtmagok hosszú láncai alakulnak ki, amelyek nem ritkán több mint 50 sejtmagból is állhatnak).

Ez persze továbbra sem jelenti, hogy szükség lenne a referencia és a teszt eredményekben található minden sejtmag minden sejtmaggal való összehasonlítására, elég csoportokat kialakítani azokból a sejtmagokból, amelyek egymás közti párosítása együttes vizsgálatot igényel, és csak ezekre kell elvégezni a fenti keresést. Ez megoldható a klaszterezési algoritmusoknál [**34**] megszokotthoz hasonló módon: veszünk egy tetszőleges referencia sejtmagot, ezt tekintjük a csoport első elemének. Ezt követően megvizsgáljuk a rendelkezésre álló teszt sejtmagok közül azokat, amelyek ezzel fedésben vannak, ezeket szintén hozzáadjuk a csoporthoz. Elég csak a teszt sejtmagokat vizsgálni, ugyanis sem a referencia, sem pedig a teszt eredményeken belül nem találkozhatunk egymást átfedő sejtmagokkal (ami már eleve a formátumból is adódik, egy raszter kép egyes pixelei jelzik, hogy ott melyik sejtmag található, ha egyáltalán van ilyen), és természetesen ugyanez igaz visszafelé is. Ezt követően átvizsgáljuk a referencia sejtmagokat, és azokat, amelyek fedésben vannak bármelyik, a csoportban lévő teszt sejtmaggal, szintén hozzáadjuk a csoporthoz. Majd ezt az iterációt felváltva folytatjuk a teszt és a referencia sejtmagok között egészen addig, amíg már nem tudjuk bővíteni tovább a csoportot (hasonló módszert alkalmaztam egy másik klaszterezési feladatnál [**35**], ahol síkbeli pontokból álló gócokat kellett megkeresni, és ott jól használhatónak bizonyult).

A végső állapotában a csoport azoknak az elemeknek a legbővebb halmazát fogja tartalmazni, amelyek közül bármelyik tetszőleges számú lépésben elérhető bármelyik másikból az átfedések láncolatán keresztül. Ezeken a csoportokon belül azonban már nem tudunk egyértelműen, visszalépések nélkül kiválasztani olyan párosításokat, amelyek garantáltan a legjobb elérhető összeredményt adják. Emiatt itt a későbbiekben leírt kereséssel próbáljuk

egymáshoz rendelni a párokat. Előfordulhatnak mind a referencia, mind pedig a teszt sejtmagok között olyanok is, amelyek nincsenek átfedésben más sejtmagokkal, ezek önmagukban alkothatnak egy-egy csoportot.

Amennyiben az összes sejtmagra elvégezzük ezt a csoportosítást, akkor jól kezelhető csoportokat fogunk kapni, ahol minden sejtmag pontosan egy csoportba tartozik, és ahol a különböző csoportokba tartozó sejtmagok között biztosan nincs átfedés. Ennek köszönhetően a sejtmagok párosítását csoportokon belül meg lehet oldani, ezzel már jelentősen le lehet szűkíteni a szükséges számítások darabszámát.

### *1.3.2. A csoportokon belüli legjobb párosítások megkeresése*

Az eredmények gyakorlati vizsgálata során kiderült, hogy a sejtmagokat sűrűn tartalmazó területeken az ilyen átfedéseken keresztül egészen hosszú láncot kell végigjárni, amelynek eredményeképpen meglehetősen nagy számú referencia és teszt sejtmagokat tartalmazó csoportok jönnek létre. Mivel az elemszám növelésével a csoportok feldolgozási ideje exponenciálisan növekszik, így célszerű valamilyen hatékony algoritmust találni az egymáshoz rendelésre, ehhez egy módosított visszalépéses keresést (*back tracking*) [**36**] alkalmaztam.

A visszalépéses keresés egy régi, de bevált és jól használható technikai olyan keresési feladatoknál, ahol nincs pontos információnk arról, hogy a keresést milyen irányban kell elkezdeni, tehát jobb megoldás nem lévén, a megoldástér szisztematikus bejárásával próbáljuk megtalálni az egyik (vagy bizonyos megvalósítások esetén az összes, vagy éppen az optimális) megoldást.

A megoldás alapelve, hogy egy gráf (fa) mélységi bejárásához hasonló módon elindul egy lehetséges úton, az egyes csomópontokban mindig az első lehetséges továbbvezető utat választva. Amennyiben nem jut el egy lehetséges végeredményhez, akkor az aktuális csomópontnál megpróbál másik továbbvezető utat keresni, amennyiben pedig ez nem lehetséges, akkor visszalép a megelőző csomópontra és ott próbál meg egy másik irányt. Az algoritmus akkor ér véget, ha megtaláltuk a szükséges megoldást, vagy a legelső szintre visszalépve már nem találunk újabb próbálkozási lehetőséget (nincs megoldás).

Az algoritmus klasszikusan akkor alkalmazható jól, ha több részfeladatra kell megoldást találnunk, ahol az egyes részfeladatok lehetséges részmegoldásai ismertek, és az alábbi feltételek teljesülnek:

- A részfeladatokra adható lehetséges részmegoldások összes lehetséges kombinációja meglehetősen nagy.
- Az egyes részfeladatokra adott egyes részmegoldások kizárják egymást, így a keresés során tudhatjuk, hogy ezeken az utakon nem érdemes továbbhaladni.

A klasszikus feladatok mellett (8 királynő elhelyezése a sakktáblán, labirintus kijáratának megkeresése, hátizsák pakolási probléma) számos gyakorlati alkalmazással is találkozhatunk [**37**][**38**][**39**]. A visszalépéses keresésre vonatkozó publikációk során ugyan többnyire annak kiváltásán fáradoznak (bármennyire is hatékonynak tűnik egy egyszerű lineáris kereséshez viszonyítva, egy jó heurisztikákkal támogatott mohó algoritmus nagyságrendekkel jobb eredményt érhet el), azonban a gyakorlatban sok esetben ez tűnik az egyetlen, jól használható megoldásnak.

Természetesen a visszalépéses keresés során is van lehetőség különféle gyorsítási lehetőségekre, például az egyes részfeladatok során különféle heurisztikák segítségével érdemes lehet optimalizálni, hogy melyik utat vizsgálja át a keresés előbb (ebben az esetben ennek kevésbé van jelentősége, mivel optimális megoldást keresünk, tehát minden lehetséges megoldást meg szeretnénk vizsgálni előbb-utóbb).

A konkrét feladatra alkalmazva, a visszalépéses keresés részfeladatai az egyes referencia sejtmagokhoz a teszt sejtmagok közül valamelyik, vele átfedésben lévő sejtmag hozzárendelései. A részfeladatok száma tehát megegyezik a referencia sejtmagok számával, az egyes lehetséges részmegoldások pedig a vele átfedésben lévő teszt sejtmagok számával. Ezt értelemszerűen elég a fent már említett csoportonként lefuttatni, hiszen az egymástól különböző csoportokban nem lehetnek egymást átfedő sejtmagok, így a keresést ilyen irányba kibővíteni felesleges.

Ehhez a csoportból ki kell gyűjteni a referencia sejtmagokat, majd mindegyikhez egy saját vektorba ki kell válogatni a vele fedésben lévő teszt sejtmagokat (*PTCL-Potential Test Cell in this Level*), hiszen a párosítási próbálkozásokat eleve csak ezekre érdemes elvégezni. A feladat speciális jellegéből adódóan most nem követeljük meg, hogy minden részfeladatot megoldjunk, mivel elképzelhető, hogy egy referencia sejtmaghoz nem rendelünk egy teszt sejtmagot sem, miként az is elképzelhető, hogy egy teszt sejtmag egyik részfeladathoz sem lett hozzárendelve. Mivel a teszt és a referencia sejtmagok darabszáma egymástól független, ezek az esetek meglehetősen gyakran előfordulnak.

A keresés végeredménye a lehetséges megoldások közül az optimális megkeresése (az a megoldás, ahol a fenti értékelés szerint a csoporton belüli legnagyobb pixelenkénti pontosság érhető el). A vizsgálandó párosítások száma még így is meglehetősen magas, ezt azonban egy kiegészítő visszalépési feltétel segítségével tudtam tovább csökkenteni: minden referencia sejtmaghoz ki kell számolni és eltárolni (*LO*), hogy a vele átfedésben lévő teszt sejtmagok közül lokálisan az optimálist választva, milyen eredményt kapnánk. Keresés közben pedig az algoritmus nem is lép tovább a következő szintekre, ha ezek az előre eltárolt értékek alapján látszik, hogy az aktuális úton továbbhaladva még akkor sem érhetne el az eddig talált legjobbnál is jobb eredményt, ha az összes következő szinten sikerülne a lokálisan optimális párosítást választani.

A fentieknek megfelelő párosítást hajtja végre az 1. algoritmus. Ennek bemenetei:

- *level*: A visszalépéses keresés által feldolgozott aktuális szint.
- *RES*: Az eredményt tartalmazó vektor.

Felhasznált függvények:

- *SCORE(X)*: Visszaadja a paraméterként átadott X eredményhez (teszt-referencia sejtmag párosítás) tartozó pontosság értékét a fent megismert pixelenkénti összehasonlítást alkalmazva.

```
Próbál(level,RES)
  Ciklus (∀TC∈PTCL[level]∪∅)
    Ha (TC∉RES[1..level-1]∨TC=∅)
      RES[level]←TC
      Ha (level=N)
        Ha (score(RES)>score(MAXRES))
          MAXRES←RES
      Különben
        Ha (score([RES[1..level] LO[level+1..N]])<score(MAXRES))
          Próbál (level+1,RES)
  Visszaad MAXRES
```

**1. algoritmus:** S*ejtmagok párosítását végző visszalépéses keresés algoritmusa*.

Az algoritmus tehát az egy csoportba tartozó teszt és referencia sejtmagok közül visszaadja az optimális párosítást, ahol a MAXRES vektor i. eleme mutatja, hogy a csoporton belüli i. referencia sejtmagot a MAXRES[i] teszt sejtmaggal célszerű párosítani (∅ esetében pedig ezt a referencia sejtmagot érdemes szabadon hagyni).

Minden csoportra külön-külön lefuttatható a fenti algoritmus, és ennek megfelelően összegyűjthetők az optimálisan egymáshoz rendelt elemek (beleértve persze az egy elemű csoportokban található, nem párosítható elemeket is). Mivel már a csoportok kialakítása során elértük, hogy az egyes csoportok egymástól teljesen független sejtmagokat tartalmazzanak, így ez a feldolgozás párhuzamosan is végrehajtható egyszerre több csoportra. A művelet tipikusan csak számítás intenzív, az egyes szálak által igényelt memória mennyisége nem jelentős, így ez nem hátráltatja a párhuzamos futtatást. Rekurzív algoritmus révén érdemes megvizsgálni a veremkezelést: a rekurzió maximális mélysége egyenlő a referencia sejtmagok számával, a verembe pedig csak néhány változó kerül, így túlcsordulástól nem kell tartanunk.

Miután az összes csoportban megvannak az optimális párosítások, ezt követően az egyes párokra (illetve egyedül maradt elemekre) elvégezhető az előzőleg ismertetett értékelés. Ennek segítségével megállapítható először a sejtmag párokra, majd ezeket összegezve a teljes megoldásra vonatkoztatható *TP*, *FP*, *FN* súlyozott pixelszám. Ezek értelmezhetők önmagukban (például a paraméterek automatikus beállítása ezt igényelheti), vagy egy egyszerűbb és látványosabb összehasonlítás kedvéért a már említett pontosság segítségével.

$$\text{Pontosság} = (TP + TN) / (TP + TN + FP + FN) \qquad (5)$$

A pontosság természetesen kiszámítható az egyes sejtmagpárokra is (a már részletezett okokból elhagyva a *TN* tagot), és az így kapott eredmények szintén sokoldalúan értelmezhetők, a gyakorlat számára érdekes lehet például annak az osztályozása, hogy hány sejtmagot sikerült megadott pontossági osztályokon belül detektálni. A fenti mérőszám azonban olyan szempontból áttekinthetőbb értéket ad, hogy így nincs szükség az egyes sejtmagok eredményeinek önkényes súlyozására, mivel a pixel szintű eredmények ezt már eleve tartalmazzák.

## 1.4. Futásidő figyelembevétele

A szegmentálás jóságát értékelő különféle módszerek általában nem veszik figyelembe a futásidőt, ami indokolt is lehet azokban az esetekben, ahol maga a feldolgozás nem időkritikus, vagy olyan rövid ideig tart, hogy a felhasználó észre sem veszi a nagyságrendi

különbségeket sem. Valósidejű alkalmazásoknál, illetve hosszú futásidejű algoritmusoknál azonban ez már lényeges tényező lehet, a szöveti képek feldolgozása pedig mindkettőt magába foglalja.

Mivel nagyméretű képeket kell feldolgozni valós időben, így hiába működnek bizonyos algoritmusok nagyon jó pontossággal, ha az átlagosan használt nagyítás mellett a futásidő megakadályozza a gyakorlati felhasználást. Például a későbbiekben részletesebben is megvizsgált régiónöveléses sejtmagkeresés implementációja egy nagyobb kép esetében (8192x8192 pixeles felbontás) akár 1 órás futásidőt is igényelhet, és ilyen nagyságrendek mellett akár egy néhány százalékos sebességnövekedés (azonos pontosság mellett) is jelentősen javíthatja a gyakorlati használhatóságot.

A futásidő mérésekor nem vesszük figyelembe a program indításához szükséges, operációs rendszer által végrehajtott műveleteket. Ugyanígy nem vesszük figyelembe a szükséges bemenet betöltésének idejét, illetve a végeredmények eltárolásának idejét. A kettő közötti lépéseket viszont már érdemes egy egységként kezelni, hiszen ezek már a keresés részét képezik, attól elválaszthatatlanok:

- **Előfeldolgozás**: Képről másolatok készítése, különféle szűrők alkalmazása stb.
- **Keresés**: Maga, a tényleges sejtmagkeresési eljárás.
- **Utófeldolgozás**: A megtalált sejtmagok osztályozása, utószűrése, esetleges további feldolgozása.

A fenti műveletek időigénye természetesen nagyban függ a méréskor használt hardver környezettől, de mivel a cél minden esetben az egyes meglévő algoritmusok összehasonlítása, így célszerűen az egyes teszteket ugyanazzal a hardverrel érdemes végrehajtani, így ez a (nagyon nehezen mérhető és összehasonlítható) paraméter figyelmen kívül hagyható.

A fentiek figyelembevételével az általam használt időmérési metódus:

1. Az alkalmazás elindítása.
2. Egy sejtmagkeresés lefuttatása időmérés nélkül (bemelegítés). Ezt a futtatást a későbbiekben nem vesszük figyelembe, hiszen itt még előfordulhatnak különféle fordítási műveletek, illetve a gyorsítótárak sincsenek olyan állapotban, mint a későbbi mérések során.
3. N darab mérés lefuttatása, ami az alábbi lépésekből áll:
    a. Időmérés elindítása.

b. Teljes sejtmagkeresési folyamat végrehajtása.
c. Időmérés leállítása.
d. Futásidők feljegyzése.

4. Az alkalmazás leállítása.
5. Az előző mérések alapján átlagos futásidő kiszámítása.

A futásidő nagyban függ a kép méretétől, emiatt célszerű az alapján normalizálni az eredményeket. Ez lehet egy egyszerű osztás a képen található pixelek számával, de ez a sejtmagoknál már látott problémák miatt nehezen összehasonlítható adatokat eredményezhet, hiszen a teljes képnek általában csak egy kis részletén található meg maga a minta, illetve szintén gyakori, hogy a mintának is csak bizonyos területein jelennek meg sejtmagok. Emiatt ez a mérőszám a gyakorlati vizsgálatok során nagyon nagy szórást mutatott, ugyanannál az algoritmusnál egy nagyméretű és csak néhány sejtmagot tartalmazó képen jóval nagyobb sebességet jelzett, mint egy kisméretű és azon belül sok sejtmagot tartalmazó esetében.

Emiatt célszerűbb a mintában található sejtmagok számát és méretét figyelembe venni a normalizáláskor. Mivel a sebességi adatok feldolgozásakor már ismerjük a pontossági vizsgálat eredményeit, célszerű lehet az alábbi mérőszám használata:

Feldolgozási sebesség = feldolgozási idő / TP pixelek száma (*6*)

A mérőszám tehát tulajdonképpen azt adja meg, hogy az adott implementációnak mennyi ideig tartott egy valóban sejtmaghoz tartozó pixel megtalálása. Ez egy meglehetősen szigorú értékelési módszer, hiszen a feldolgozási idő tartalmazza a téves találatok illetve a nem sejtmaghoz tartozó területek átvizsgálását is, de mivel ez előbbit nem tekinthetjük „értékes” munkának, az utóbbit pedig már a pontosság vizsgálatakor sem vettük figyelembe, így ez az érték tűnik a legcélravezetőbbnek az egyes algoritmusok összehasonlításakor.

Az így kapott mérőszám természetesen minden egyes kép feldolgozása esetén más lesz a kép jellegétől függően, így csak az azonos képet feldolgozó algoritmusok eredményét érdemes ennek segítségével összehasonlítani. Magát az összehasonlítást ezt követően pedig célszerű elvégezni minden tipikusan előforduló minta esetében (egészséges szövet, beteg szövet, hibás minta stb.), de ez a már említett „gold standard” tesztekhez igazodva egyszerűen megoldható, hiszen rendelkezésünkre áll egy 41 mintából álló gyűjtemény, amely különféle jellemzőkkel bíró szövetmintákat tartalmaz).

## 1.5.Sejtmagkereső algoritmusok összehasonlító vizsgálata

### *1.5.1. Régiónövelésen alapuló módszerek*

Az első megvizsgált módszer a hagyományos, szekvenciális régiónöveléses sejtmagkeresés volt. A módszer alapja (részletesebb leírás a következő fejezetben található), hogy a feldolgozandó képen keres egy *kiinduló pontot* (*seed*), amely nagy valószínűséggel egy sejtmag területén belül található. Ezt követően megvizsgálja ennek a pontnak az egységnyi környezetét, hogy az ott található 4 (vagy implementációtól függően esetleg 8) szomszédos pont közül melyiket választva lesz a legvalószínűbb, hogy egy sejtmaghoz hasonló formához kezd közelíteni az így létrejött, egy pixellel nagyobb méretű régió. Ezt egy jósági függvény segítségével határozza meg, amely különféle szín és intenzitás [**14**] értékek alapján próbál egy jósági tényezőt rendelni a vizsgált pont irányába történő növeléshez. Ezt követően az így létrejött két pont méretű régiót tekintjük egy sejtmag jelöltnek, majd ennek a környezetében lévő 6 darab pontot kezdjük el vizsgálni az előbb megismert feltételek szerint, ezek közül ismét a jelölthöz adva azt, amelyiknél a jósági függvény a legnagyobb értéket adta. Mindezt addig folytatjuk, amíg valamelyik megállási feltétel be nem következik, majd ezt követően hozhatunk döntést afelől, hogy az így kapott területet sejtmagként elfogadjuk-e vagy sem.

Két régiónövelésen alapuló módszert is megvizsgáltam, az egyik egy hagyományos, CPU-n futó implementáció [**9**] (amelynek bizonyos részei egy szálon futnak, bizonyos részek pedig kihasználják a többmagos architektúrák lehetőségeit), a másik pedig egy általam kifejlesztett adatpárhuzamos, GPGPU-n implementált változat [**40**] (ami a lehető legnagyobb mértékben kihasználja a többmagos rendszerek lehetőségeit).

### *1.5.2. K-közép eljáráson alapuló módszerek*

A harmadik vizsgált módszer a képfeldolgozásban szintén gyakran használt *K-közép* (*K-means*) [**13**] módszer. Ez egy klasszikus klaszterezési technika, amely meglehetősen jól használható szöveti minták szegmentálására is, mivel megfelelően paraméterezve, a színintenzitások alapján meg tudja határozni egy képen, hogy az egyes pixelek közül melyek tartoznak egy sejtmaghoz, és melyek nem. Számos implementációja létezik ennek a módszernek, a kutatás során az Óbudai Egyetem Biotech csoportja által fejlesztett változatot vizsgáltam meg.

Önmagában a K-közép eljárás még nem adott volna a pontossági vizsgálat számára értékelhető eredményt, mivel az pusztán csak annyit tudott eldönteni, hogy a kép egy pixele sejtmaghoz tartozik-e vagy sem. Azt pedig láthattuk, hogy a kiértékelés során nem elégszünk

meg egy pixelenkénti összehasonlítással, minden egyes teszt és referencia sejtmagot külön entitásként kell kezelnünk, ezeket egymáshoz kell rendelnünk, majd az összerendelés alapján kell a későbbi kiértékelést végrehajtani.

Emiatt módosítottam a már implementált algoritmust, illetve kiegészítettem egy utófeldolgozással, amelyik a sejtmagokhoz tartozó pixelek halmazát felbontotta különálló sejtmagokra. Az utófeldolgozás lépései:

1. Először egymást követő több eróziós művelettel meghatároz középpontokat (azokat a pontokat, amelyek az egymást követő eróziók után egy pixel méretűként megmaradnak), ezeket tekintjük majd a későbbi sejtmagok középpontjának.
2. A középpontokból kiindulva ezt követően egy egyszerűsített régiónövelést hajt végre, ami során csak olyan további pontokat ad a folyamatosan növesztett sejtmag jelöltekhez, amelyek megjelentek a K-közép eredeti futása során a sejtmagnak tekintett pixelek között. A növelés során a jósági függvény mindig olyan értékeket ad vissza, hogy a lehetséges új pontok közül azt válassza, amelynek hatására a sejtmag a leginkább körszerűbb lesz.
3. Megállási feltételt külön nem kell megszabni, hiszen a cél az, hogy az így kialakuló sejtmagok teljesen lefedjék a K-közép algoritmus által is jelzett területet, tehát akkor kell csak megállni, amikor ezek a pontok elfogytak.

Hogy egy kezdőpontból kiindulva az első sejtmag ne foglalja el az összes, onnan elérhető pontot, az egyes középpontokból indított régiónövelések párhuzamosan futnak: minden egyes lépésben, minden sejtmag jelölthöz egy pixelt adunk hozzá (amennyiben ez lehetséges). Így az egyes sejtmag jelöltek garantáltan azonos sebességgel fognak növekedni (a szekvenciális megvalósítás miatt külön szinkronizációra nincs szükség). A gyakorlati tapasztalatok alapján a módszer meglehetősen hatékonyan el tudta határolni az egymástól független sejtmagokat.

### *1.5.3. Pontossági vizsgálat*

Az ismertetett pontosságvizsgálati módszer a sejtmagok nagy száma miatt már nem kezelhető manuális módszerekkel, emiatt implementáltam azt C# programozási nyelv segítségével [**41**][**42**][**43**] (de a két nyelv hasonlósága miatt [**44**] az így készült megoldás gyorsan portolható Java nyelvre is, amennyiben a platformfüggetlenség lényeges lenne). Az így elkészült alkalmazás segítségével lehetőség nyílt a rendelkezésre álló három sejtmagkeresési eljárás részletes összehasonlító vizsgálatára. Ez a fejezet ennek eredményeit tartalmazza.

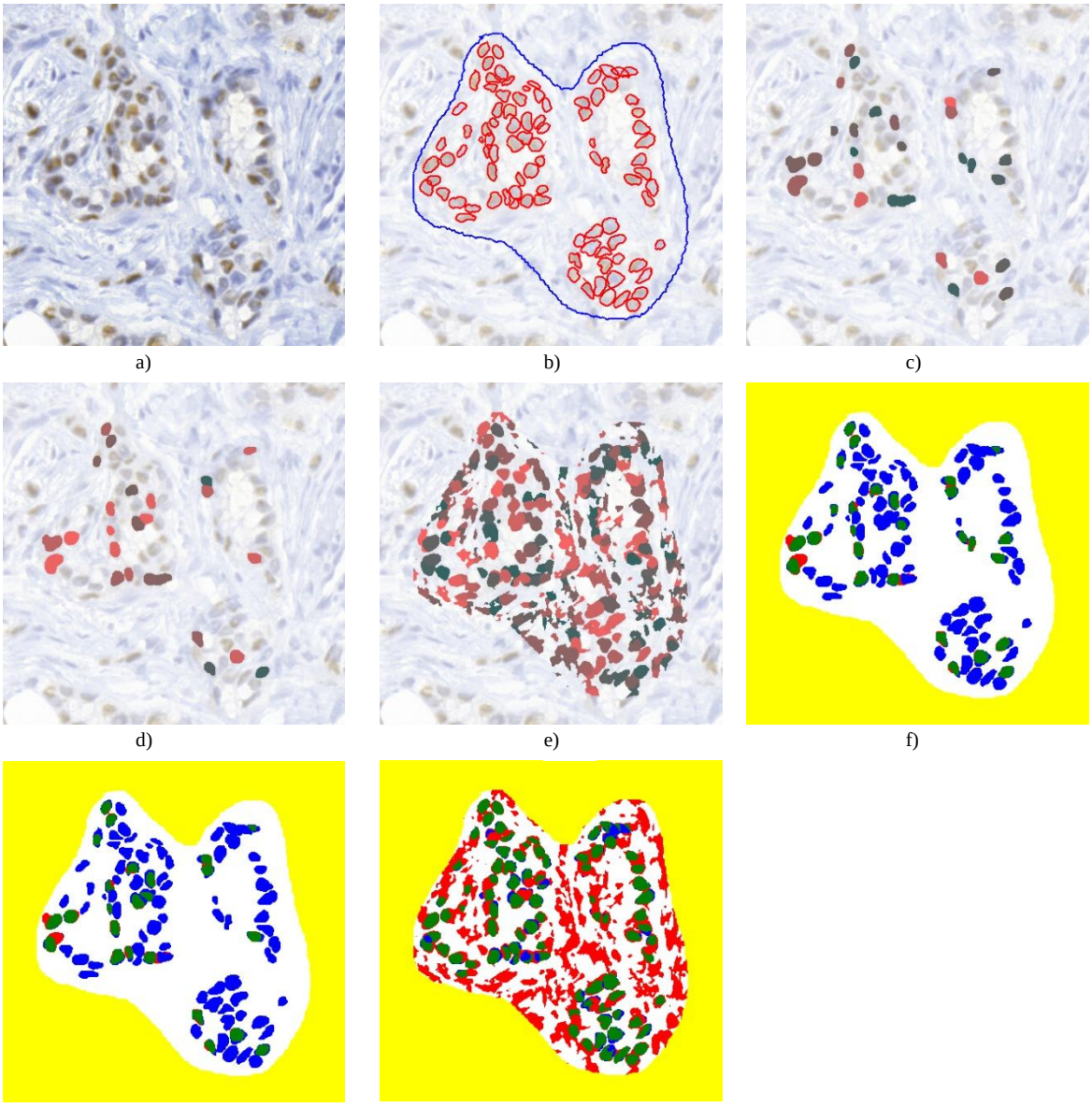
| Eredmény | CPU alapú régiónövelés | GPU alapú régiónövelés | K-közép |
|---|---|---|---|
| **Igaz-pozitív pixelek száma** | 4598 | 4193 | 13171 |
| **Igaz-negatív pixelek száma** | 64359 | 64378 | 45097 |
| **Súlyozott hamis-pozitív pixelszám** | 747,44 | 744,46 | 22573,2 |
| **Súlyozott hamis-negatív pixelszám** | 13285,35 | 13706,97 | 4272,0 |
| **Pontosság** | 83,09% | 82,59% | 68,46% |
| **Teljes futásidő** | 137505ms | 40785ms | 25999ms |
| **Pixelre vetített futásidő** | 29,9 ms/px | 9,72 ms/px | 1,97 ms/px |

**1. táblázat:** *Pontossági vizsgálat részletes eredményei a „B2007_00259_PR_02_validation” mintára, KT = 0,3 esetén. A nem annotált terület mérete minden esetben 1 227 946 pixel volt.*

A gyakorlatban a program kimenete néhány számadat: igaz-pozitív, igaz-negatív, hamis-pozitív, hamis-negatív pixelek száma, manuálisan annotált területen kívüli pixelek száma, számított pontosság, feldolgozási sebesség (1. táblázat); illetve képek, amelyek pixelenként színekkel jelzik, hogy az egyes algoritmusok a bemenet melyik részét értékelték jól vagy rosszul.

Az eredmények látványosan ábrázolhatók, ha a pixeleket annak megfelelően színezzük meg, hogy azok melyik kategóriába estek bele a pontosság vizsgálatakor (3. ábra).

a) **Eredeti szövetminta** (kivágva és átméretezve)
b) **Referencia eredmény**: A kék vonal mutatja a manuálisan annotált terület határait, azon belül pedig a piros körvonallal jelzett objektumok a patológusok által megjelölt sejtmagok.
c) **CPU alapú régiónövelési eljárás (RG-C) eredménye**: Színes területek a detektált sejtmagok pixelei, különböző színek különböző sejtmagokat képviselnek.
d) **GPU alapú régiónövelési eljárás (RG-G) eredménye**: Színes területek a detektált sejtmagok pixelei, különböző színek különböző sejtmagokat képviselnek.
e) **K-közép módszer (KM) eredménye**: Színes területek a detektált sejtmagok pixelei, különböző színek különböző sejtmagokat képviselnek.




**3. ábra** *Összehasonlítás eredménye: a) Eredeti minta (b) Referencia eredmény (c) RG-C eredmény (d) RG-G eredmény (e) KM eredmény (f) RG-C kiértékelési eredménye (g) RG-G kiértékelési eredménye (h) KM kiértékelési eredménye.*

f) **CPU alapú régiónövelési eljárás eredménye összehasonlítva a referencia képpel**: Zöld pixel: igaz-pozitív találat, fehér pixel: igaz-negatív találat, piros pixel: hamis-pozitív találat, kék pixel: hamis-negatív találat, sárga pixel: az annotált területen kívüli terület.

g) **GPU alapú régiónövelési eljárás eredménye összehasonlítva a referencia képpel**: Zöld pixel: igaz-pozitív találat, fehér pixel: igaz-negatív találat, piros pixel: hamis-pozitív találat, kék pixel: hamis-negatív találat, sárga pixel: az annotált területen kívüli terület.

h) **K-közép eljárás eredménye összehasonlítva a referencia képpel**: Zöld pixel: igaz-pozitív találat, fehér pixel: igaz-negatív találat, piros pixel: hamis-pozitív találat, kék pixel: hamis-negatív találat, sárga pixel: az annotált területen kívüli terület.

A futásidő mérése minden esetben az alábbi konfiguráción történt (ugyanez a konfiguráció vonatkozik az összes későbbi mérésre is):

- Processzor: Intel(R) Core(TM) Quad CPU Q9400
  - Magok száma: 4
  - Órajel: 2660 Mhz
  - Technológia: 45nm
  - Gyorsítótár: L2 cache 6MB
- Memória: 8GB DDR2
- Grafikus kártya: Nvidia 580GTX
  - CUDA magok száma: 512
  - Órajel (grafikus/processzor): 772Mhz/1544Mhz
  - Memória: 1536Mb GDDR5
- Operációs rendszer: Windows7 64 bit
- Fordító: Visual Studio 2008
- CUDA változat: 2.0

Az eredmények értékelését követően az alábbi következtetéseket vonhatjuk le:

- A CPU és a GPU alapú régiónövelési algoritmusok pontossága szinte azonosnak tekinthető. Ez nem meglepő, mivel a két algoritmus alapelve hasonló, mégha a tényleges működésük és megvalósításuk egymástól teljesen különbözik is. A kisebb különbségeket az okozza, hogy a kiinduló pontok kiválasztási sorrendje nem minden esetben egyértelmű, viszont a sorrend befolyásolja a végeredményt.
- A K-közép algoritmus látványosan gyorsabb, mint a régiónövelésen alapuló társai. Második a GPU alapú régiónövelés, és csak ezt követi a CPU alapú megoldás.
- A legtöbb esetben (4 eset kivételével) a K-közép algoritmus pontossága jelentősen elmarad a régiónövelést használóktól.

  Látható, hogy mindkét régiónövelésen alapuló eljárás „túl óvatos”, így a hibákat általában az okozza, hogy néhány sejtmagot nem találnak meg, vagy ha igen, akkor is az utólagos ellenőrzés során elvetik őket. A későbbiekben a paramétereket célszerű lehet ebbe az irányba finomhangolni.

| Eredmény | CPU alapú régiónövelés | GPU alapú régiónövelés | K-közép |
|---|---|---|---|
| **Igaz-pozitív pixelek száma** | 710161 | 709158 | 900078 |
| **Igaz-negatív pixelek száma** | 4691515 | 4687161 | 3492053 |
| **Súlyozott hamis-pozitív pixelszám** | 319851 | 333403 | 1877068 |
| **Súlyozott hamis-negatív pixelszám** | 976833 | 977352 | 718641 |
| **Nem annotált pixelek** | 68650072 | 68650072 | 68650072 |
| **Pontosság** | 80,64% | 80,46% | 62,86% |
| **Teljes futásidő** | 3951sec | 2526sec | 1425229ms |
| **Pixelre vetített futásidő** | 5,56 ms/px | 3,56 ms/px | 1,58 ms/px |

**2. táblázat:** *Pontossági vizsgálat részletes eredményei 36 mintára.*

A pontossági vizsgálatot végrehajtottuk 36 referencia képre[1], az összesített eredményeket tartalmazza a 2. táblázat (*KT* = 0,3 esetén). A teljes, részletes eredmények pedig az 1. számú függelékben találhatók (16. táblázat, 17. táblázat, 18. táblázat, 19. táblázat).

## 1.6.Eredmények értékelése

Pontosság szempontjából tehát a két régiónövelési algoritmus egyenértékűnek tekinthető, lényegi különbség tulajdonképpen csak a feldolgozási időben van, ahol egyértelműen a GPU megvalósítás nyert. Minden esetben gyorsabb volt, a sebességnövekedés átlagosan másfél-kétszeres, de érdemes megjegyezni, hogy ezek a minták nagyon kis méretűek, a GPU előnye sokkal inkább nagyobb képek esetén jelentkezik.

A K-közép módszer kevésbé törekszik precíz megvalósításra, és meglepő módon pont emiatt bizonyos esetekben, amikor a kép nehezen értelmezhető (rosszul fókuszált vagy kevésbé kontrasztos képek), amikor a régiónövelés nem ad vissza egy sejtmagot sem, akkor ezzel sikerül jobb eredményt elérnie. Nem érdemes azonban ennek túl nagy jelentőséget

[1] A 41 rendelkezésre álló referencia kép közül 1 esetében láthatóan hibás volt a manuális annotáció (egy sejtmag se volt megjelölve), 4 esetében pedig még a visszalépéses keresés használatával is túl hosszú ideig futott a kiértékelés. Ez utóbbi problémát egy későbbi továbbfejlesztés áthidalta (túl nagy átfedő sejtmagszám esetén a kiértékelés dinamikusan egyszerűsít a legjobbnak tűnő párok előre rögzítésével), ez azonban némi pontatlanságot eredményez, emiatt az így számított adatok itt nem jelennek meg.

tulajdonítani, a legtöbb képnél már nem eredményes ez a technika, sok hamis-pozitív találatot ad vissza.

Mindezek mellett azonban mindenképpen érdemes kiemelni a K-közép algoritmus nagyon jó sebességi adatait. Ezek fényében, ha teljes megoldásnak nem is tűnik megfelelőnek, de egy gyors előfeldolgozásra mindenképpen érdemes lehet számba venni.

A gyakorlatban a mérőszám tehát jól használható, így kimondható, hogy sikerült kidolgozni egy megfelelő eljárást, amely ötvözi a pixel- és objektumszintű értékelések előnyeit, így a meglévő módszereknél jobban alkalmazható szövetmintákban sejtmagokat kereső szegmentáló eljárások objektív összehasonlítására.

Sikeresnek tekinthető a kiértékelő algoritmus implementációja is, amely nagy mennyiségű, egymást átfedő referencia és teszt sejtmagokat tartalmazó szegmentálási eredmények esetén a hagyományos keresésekhez képest kevesebb lépéssel határozza meg a legnagyobb pontossági értéket nyújtó párosításokat.

Az itt vizsgáltak mellett még számos más algoritmust találhatunk a szakirodalmat átvizsgálva, az ezekkel való összehasonlítást azonban a pontossági eredmények közlésének hiánya miatt nem tudtam összehasonlítani [**5**] [**6**] [**7**] [**9**]. [**21**]. Néhány publikáció tartalmaz ugyan pontossági értékeket (pl. 80% [**4**], 95% [**10**]), de nem tartalmaz információkat a pontossági vizsgálat pontos módjáról, néhány esetben pedig van ugyan leírás a vizsgálati módszerről (pl. 97% [**22**]), de az olyan mértékben különbözik az általam használt metodikától, hogy nem tartom indokoltnak az ott olvasható értékeket összehasonlítani az általam mértekkel**.**

# 2. ADATPÁRHUZAMOS RÉGIÓNÖVELÉSI ALGORITMUS KIDOLGOZÁSA

## 2.1. Paraméterek, tárhelyek megválasztása

Az egyik legpontosabb sejtmag detektálási algoritmus alapja egy régiónövelő eljárás, amely egy megadott kiindulópontból (*seed pont*) elindulva megpróbálja meghatározni az ott található sejtmag pontos elhelyezkedését. Maga a növelés egy jól párhuzamosítható műveletnek tekinthető, így célszerű lehet azt adatpárhuzamos környezetben megvalósítani.

Mivel olyan algoritmus lenne az ideális, amelyik alkalmas a jelenleg legnagyobb számítási kapacitású GPGPU-kon való implementációra (azon belül is a CUDA környezettel megcélozható Nvidia eszközökre), ezért számos korlátot figyelembe kell venni a tervezése során. Az első lényeges lépés a végrehajtási környezet paramétereinek meghatározása, hogy az alkalmazkodjon a grafikus kártyák specialitásaihoz (blokkok, szálak száma).

Maga a régiónövelés négy lépésből áll, amelyek egymásra épülnek:

1. új kontúrbővítési pontok keresése,
2. az így elérhető pontok értékelése,
3. ezek közül a legnagyobb értékű pont kiválasztása,
4. majd ezzel a terület bővítése.

Ezek a műveletek önmagunkban jól párhuzamosíthatók, azonban minden műveletnek szüksége van az előző lépés eredményére, így szinkronizációs pontok beiktatására van szükség. Ez jelentősen leszűkíti a választható megoldások körét, ugyanis megfelelő teljesítménnyel rendelkező szinkronizációs műveleteket a jelenleg elérhető GPGPU-kon csak egy blokkon belül lehet elérni, így célszerűnek tűnik egy-egy sejtmagkeresést egy-egy blokkon belül indítani (ez pedig azonnal egy korlátot is jelent számunkra, mivel az egy blokkban lévő szálak száma korlátozott, a jelenleg kurrensnek tekinthető *Compute Capability 2.1* esetén ez az érték 1024). A sejtmagok növelése során az egyik legszámításigényesebb lépés a kontúrpontok értékelése, így célszerű a párhuzamosítást ehhez igazítani: egy szál egy kontúrpontot vizsgál majd. Az említett korlát miatt ez azt jelenti, hogy egy sejtmagkeresés során a *maximális kontúrhossz* nem haladhatja meg az 1024 képpontot, de az előzetes vizsgálatok alapján ez nem jelent problémát, a rendelkezésre álló képek adatai alapján a kontúrok átlagos hosszúsága 240 pixel, a leghosszabb kontúr pedig 412 pixel hosszúságú, úgyhogy bőven maradnak így is tartalékok. A későbbiekben pedig lehetőség nyílik az

algoritmus továbbfejlesztésére, hogy minden szál egy helyett kettő vagy akár több kontúrpontot is megvizsgálhasson. Az algoritmust tehát úgy tervezem, hogy az indított szálak száma egyenlő legyen a maximális kontúrhossz számával.

Maga a sejtmag kereső eljárás a konstans paraméterek mellett egy koordinátát igényel, a kiindulópont helyzetét. Mivel egy időben több blokk több sejtmagot is kereshet párhuzamosan, lehetőség van egyszerre több ilyen pont átadására is. A blokkok száma értelemszerűen egyenlő lesz a betöltött kiinduló pontok számával, maga a futtatáshoz szükséges *rács* (*grid*) pedig egydimenziós lesz.

A CPU és a GPGPU közötti kommunikációt az *eszköz memórián* (*device memory*) keresztül, illetve a kernel indítás paraméterein keresztül lehet megvalósítani. A nagy számú kiindulópont miatt ezek közül az eszköz memórián keresztül történő kommunikációt választottam. Ez lehetővé tette a továbbfejlesztések során azt, hogy a kiindulópontok betöltése független legyen a régiónövelések indításától, így azt a CPU és a GPGPU is megtehette.

Emellett természetesen sok egyéb paramétert figyelembe kell venni a futás során (eredeti és különféle előfeldolgozáson átesett képek, keresési paraméterek, megállási feltételek), ezek azonban az egyes kernel indítások között nem változnak, így azok folyamatosan a GPU eszköz memóriájában tárolhatók.

A paraméterek mellett a kernel működése során számos segédváltozót használ, ezekben tárolja a régió bővítésének történetét (új pontok helyzete, azok felvételének sorrendje) illetve számos segédadatot a kiértékeléshez (intenzitás különbségek) és a bővítéshez (az aktuális régió kontúrpontjainak koordinátái). Ezek jelentős részéből nincs szükség külön példányokra minden egyes szálnak, így ezek célszerűen a GPGPU *megosztott memóriájában* (*shared memory*) tárolhatók. Ennek a nagysebességű tárolónak a használata jelentősen csökkenti a kernel futásidejét, azonban egy újabb lényeges korlátot is jelent a felismerni kívánt sejtmagok méretére: Compute Capability 2.1 esetén a megosztott memória mérete 48KB multiprocesszoronként, és figyelembe kell vennünk azt is, hogy egy multiprocesszor egyszerre több blokkot is futtathat, így ebbe a memóriaterületbe el kell férnie az összes blokk minden kontúrpont adatának (kiegészítve néhány segédváltozóval, amelyeket szintén célszerű itt tárolni).

## 2.2. Régiónövelő iteráció

Maga a régiónövelés három egymást követő lépést iterál valamelyik megállási feltétel bekövetkeztéig:

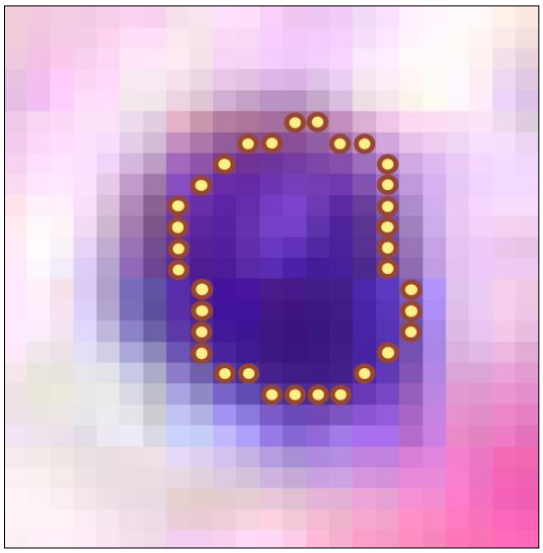
1. **Szomszédok vizsgálata**: Megvizsgálja, hogy milyen irányokba lehet bővíteni a már meglévő kontúrt. A négyes szomszédsági vizsgálatot értelemszerűen csak az utoljára elfogadott kontúrpont körül érdemes elvégezni (a kernel indításakor ez alatt magát a kiindulópontot értjük). Mivel a környező pontok vizsgálata egymástól független, ez is párhuzamosan történik, a blokk első 4 szála vizsgálja meg az egyes irányoknak megfelelő pontokat, hogy azok megfelelnek-e további növelés céljából:
   - nem részei az aktuális kontúrnak vagy régiónak,
   - illetve egyéb sejtmagokhoz tartozó régióknak.
2. **Kontúrpontok értékelése**: Ezt követően értékelni kell az egyes kontúrpontokat, hogy melyik irányban érdemes tovább növelni az eddig felismert régiót. Ehhez minden ponthoz ki kell számolni az alábbi költségfüggvényt [**9**]:

$$C(x,y) = \mathrm{abs}(I_{(x,y)} - I_{\text{régió}}) + \alpha \frac{d((x,y),(x,y)_{\text{régió}})}{\max_i (d((x,y)_i,(x,y)_{\text{régió}})} \qquad (7)$$

   Ahol:

   - $I_{(x,y)}$: Intenzitás a vizsgált (x,y) szomszédban.
   - $I_{régió}$: A régió átlagos intenzitása.
   - $d(.)$ : Euklideszi távolság.
   - $(x, y)_{régió}$: A régió súlypontja.
   - $i, (x, y)_i$: Valamely (a feltételeknek megfelelő) kontúrpont.
   - $\alpha$ : Súlytényező, konstans.

   A költségfüggvény két tag súlyozott összege. Az első tag az intenzitásbeli hasonlóságot, a második a súlyponttól mért képtérbeli távolságot minimalizálja. Fontos észrevennünk, hogy az $I_{régió}$ és az $(x, y)_{régió}$ minden új pont felvételekor megváltozik, így minden iterációban minden kontúrpontra újra kell számolni a függvényt. Ez azonban egy tipikus adatpárhuzamos számítás, így GPGPU-n jól párhuzamosítható, minden szál egy-egy kontúrpont költségét számítja (4. ábra).



**4. ábra:** *A régiónövelés egy köztes fázisa. A sárga körök mutatják az aktuális sejtmagjelölt kontúrját. Minden kontúrpontot egy külön szál dolgoz fel.*

3. Ki kell választani a legkisebb költséggel bíró kontúrpontot. A GPU *atomicMin* művelete jól használható arra, hogy a szálak megállapítsák, hogy mennyi a legkisebb költség értéke, majd pedig ezt a saját maguk által kiszámolt költséggel összehasonlítva megállapíthatják, hogy ez melyik kontúrponthoz tartozik. Mivel több ilyen is lehet, ezt egy versennyel döntik el, hogy melyikük értéke fog majd továbbjutni a kiértékelések során.

   Az algoritmusnak ez a lépése amiatt érdekes, mert ha esetleg több szál is a minimális költséget számolta a hozzá tartozó kontúrpont értékelésekor, akkor pusztán a véletlenen múlik, hogy ezek közül melyik fog győztesként kikerülni a kiválasztásból. A hagyományos CPU megvalósítással szemben (egyszerű minimum-kiválasztás) a GPGPU algoritmus innen kezdve már nem feltétlenül determinisztikus.

Minden iterációt követően kiértékelésre kerül egy jósági függvény, amely tartalmazza megadott pillanatban a régió külső-belső körvonala mentén mérhető intenzitás különbséget, illetve a régió körkörösségi tényezőjét. Az eljárás addig tart, amíg a régió eléri a maximális méretet (képpontban vagy sugár alapján), eredménye azonban nem feltétlenül a növelés legutolsó fázisában látható régió lesz, hanem a leállás pillanatáig a jósági függvény maximumához tartozó állapot.

Az egymással érintkező sejtmagok esetére egy másik megállási feltételt is tartalmaz az algoritmus. A tapasztalatok szerint az aktuális régió átterjedése egy másik sejtmag területére detektálható az intenzitás változásából [**9**]. A jellemzően csökkenő intenzitás ugyanis ilyenkor

hirtelen növekedni kezd. Emiatt az intenzitás-különbség időbeli függvényét időben differenciáljuk, és ha az így képzett függvény meghalad egy értéket, a régiónövelést leállítjuk.

```
RégióNövelés(kezdőpont)
  régió←kezdőpont; utolsó←kezdőpont; legjobb←Ø; kontúr←Ø
  Ciklus
    SegédVáltozókFrissítése(régió, I_régió, R_MAX, Center)
    Párhuzamosan 4 szálon {szálindex = 1,2,3,4}
      ⇊Ha Létezik(utolsó+D_szálindex) akkor
      ⇊  új←utolsó+D_szálindex
      ⇊  Ha új∉régió∪kontúr és Szabad(új) akkor kontúr←kontúr∪új
      ⇊Elágazás vége
    C_MAX←0
    Párhuzamosan Méret(kontúr) szálon (szálindex = 1,2..Méret(kontúr))
      ⇊KP←kontúr[szálindex]
      ⇊C←|I_KP-I_régió|+alfa*(d(KP,Center)/R_MAX)
      ⇊C_MAX←AtomiMaximum(C,C_MAX)
    Párhuzamosan Méret(kontúr) szálon (szálindex = 1,2..Méret(kontúr))
      ⇊Ha C=C_MAX akkor utolsó←kontúr[szálindex]
    régió←régió∪utolsó
    kontúr←kontúr\utolsó
    Ha legjobb=Ø ∨ Score(régió)>Score(legjobb) akkor legjobb←régió
  Ciklus amíg ¬MegállásiFeltétel(régió, kontúr)
  Visszaad legjobb
```

**2. algoritmus:** Adatpárhuzamos régiónövelés vázlata

Ezt mutatja a 2. algoritmus, ahol

- $I_{régió}$: Régió átlagos intenzitása.
- $R_{MAX}$: Régió legnagyobb sugara.
- *Center*: Régió tömegközéppontja.
- *D:*Tárolja a növekedési irányokat: $D_1$=[0,1]; $D_2$=[1,0]; $D_3$=[0,-1]; $D_4$=[-1,0].
- $I_{pont}$: Megadott pont intenzitása.
- *Létezik(pont):* Megadja, hogy egy pont még a képen belül van-e.
- *Szabad(pont):* Megadja, hogy egy pont nem része egy másik régiónak.
- *d(a,b):*Az a és b pont (euklideszi) távolsága.
- *Score(régió)*: Régió jósági tényezőjének értéke (körkörösség, intenzitás stb. alapján).
- *MegállásiFeltétel(régió,kontúr)*: Régiónövelés megállási feltételének vizsgálata.
- ⇊ jel pedig az adatpárhuzamosan futtatott részeket jelzi.

## 2.3. Utófeldolgozás

A régiónövelést követően további műveleteket célszerű végrehajtani a detektált régiókon. Érdemes a megtalált sejtmagot dilatálni, hogy a terület kontúrját is magába foglalja az eredményként visszaadott régió. Emellett lényeges egy kitöltés végrehajtása, ugyanis előfordulhat, hogy a megtalált régió lyukas [**9**].

Ezt az utófeldolgozást a klasszikus konvolúciós műveletek (erózió, dilatáció) segítségével egyszerűen meg lehet oldani. Ezek a módszerek hatékonyan implementálhatók adatpárhuzamos környezetekben [**45**], így ez a GPGPU esetében is megoldható. Ezeket a műveleteket nem célszerű külön kernelben implementálni, mivel sokszor kell ezeket az eljárásokat használni, és a meglehetősen magas kernel indítási költség jelentősen rontaná az elérhető teljesítményt. Az utófeldolgozás műveletei így a régiónövelés kernelben kerültek megvalósításra.

A GPGPU-k programozásának specialitása azonban, hogy a futtatandó blokkok és szálak számát a kernel indításakor kell meghatározni, futás közben ez az érték már nem módosítható. A szakirodalomban gyakran megjelennek a különféle konvolúciós műveletek grafikus kártyákon való megvalósításai, ezek azonban mindig feltételezik, hogy szabadon választhatnak blokkméretet. Ebben az esetben azonban a régiónövelési kernelt a kontúrpontok számához igazított szál mennyiséggel indítottuk el, és mivel ugyanezen a kernelen belül kell végrehajtani a különféle konvolúciós műveleteket is, ez együtt jár azzal a korláttal is, hogy ez utóbbi műveleteket is úgy kellett megvalósítani, hogy egy már előre meghatározott szál mennyiséggel is hatékonyan tudjanak működni. Ez a konfiguráció (egy blokk és azon belül minél több szál használata) nem feltétlenül előnyös ezeknél a feladatoknál, így az elméletileg elérhető legnagyobb teljesítményt valószínűleg nem közelítik meg, de sikerült kidolgozni olyan megvalósításokat, amelyek így is hatékonyan végzik el feladatukat.

Az utófeldolgozást követi egy végső értékelés a régió területe, formája (körkörössége), intenzitása, a kontúr külső-belső intenzitáskülönbsége alapján. Ez már egy egyszerű döntés, de a memória átvitel minimalizálása miatt ennek kiértékelése is a GPGPU-n történik meg. Minden blokk első szála elvégzi ezt a kiértékelést, majd a végeredményt (sikerült-e sejtmagot detektálni vagy pedig sem, illetve ha igen, akkor ennek a sejtmagnak a részletes adatai) pedig elhelyezi az eszköz memóriában találhat eredmény tömb blokk indexnek megfelelő helyére.

Miután minden blokk végzett, ezt a tömböt egy memória átvitellel a CPU vissza tudja olvasni, és az adatok további feldolgozás már ott történik meg.

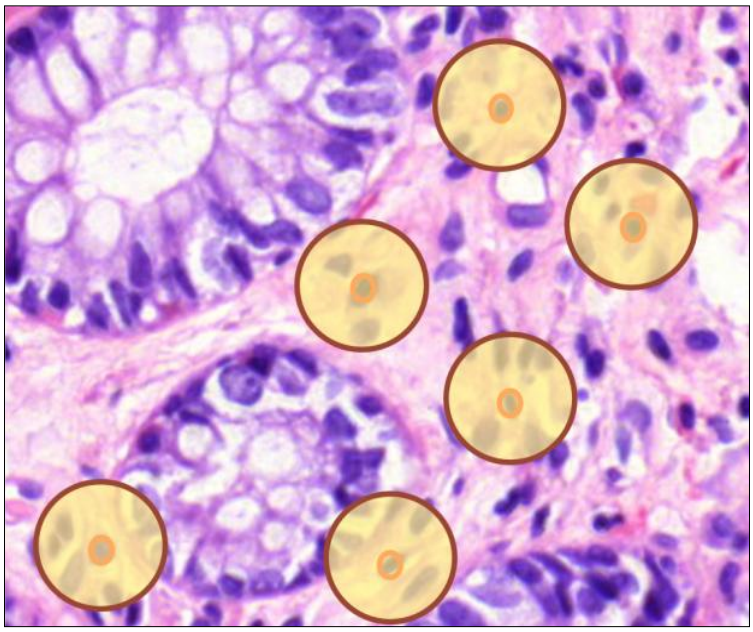
## 2.4.Kiindulópont keresés párhuzamosítása

Az egész algoritmus futásideje szempontjából az kezdőpontok keresése szinte elhanyagolható a régiónövekedés idejéhez képest. Mégis indokolt ennek megvalósítása a GPGPU-n, mivel az egyes régiónövelési eredmények (az előzőleg megtalált régiókat alkotó pontok koordinátái) szükségesek a következő kiindulópont kereséséhez, így minden kereséshez az előző régiónövelési eredményeket át kellene másolni a GPGPU memóriájából a CPU memóriájába, ami jelentősen rontaná a futásidőt.

A kiindulópont keresés is egy jól párhuzamosítható feladat, mivel a cél a megadott legnagyobb intenzitású, különféle feltételeknek megfelelő (nem lehet egy már feldolgozott régió területén belül stb.) pontok megtalálása. A CPU-n futó szekvenciális algoritmus esetén ez egy pontot jelent, a GPGPU esetében azonban többet is, mivel több blokkban érdemes lehet egyidőben több sejtmagkeresést is elindítani. Ez utóbbi esetben számos problémát vetnének fel az egymáshoz közeli kiindulópontok, hiszen azok párhuzamos keresése további megoldandó lehetőségeket vetne fel.

Az alapvető cél, hogy ne legyenek olyan pixelek a képernyőn, amelyek egyidőben több sejtmaghoz is tartoznak, többféle módon is elérhető, azonban mindegyik meglehetősen sok további erőforrást igényelne:

A legegyszerűbb megoldás az lehet, hogy a folyamatosan növekedő régiók minden egyes területi bővítés során figyelembe vennék a velük párhuzamosan futó más régiók aktuális állapotát is (esetleg már a kontúrpontok kiértékelése során), és csak olyan pontok irányába tudnának terjeszkedni, amit egy konkurens növelés még nem foglalt el. Ez azonban drasztikusan megnövelné a futásidőt, mivel az egyes blokkokban futó szálakat szinkronizálni kellene minden új pont felvételekor, ami minden esetben azt jelenté, hogy az összes blokknak meg kellene várnia a leglassabb konkurens blokkot (GPGPU környezetben a blokkok közötti kommunikáció egyébként sincs jól megoldva, úgyhogy ez külön problémákat vetne fel).



**5. ábra:** *Egymástól kellő távolságban lévő, így függetlennek tekinthető kiinduló pontok.*

A másik lehetőség, hogy minden régiónövelés lefutna a többitől függetlenül, majd miután azok végeztek, egy utófeldolgozás ellenőrizné, hogy a létrejött sejtmagok között vannak-e átfedések. Ha pedig igen, akkor ezeket szét kellene választani, majd újra eldönteni, hogy az így megmaradt csonka sejtmagok még mindig megfelelnek-e a követelményeinknek. Ez ismét jelentős időveszteséget jelentene, hiszen számos esetben el kellene dobni a már megtalált sejtmag jelölteket.

Szerencsére azonban tudjuk, hogy megadott nagyítású kép esetén egy sejtmag maximálisan mekkora lehet, így tudható, hogy annak pontjai maximálisan mekkora sugarú körben helyezkedhetnek el, így az egymástól legalább négyszer ekkora távolságból lévő kiindulópontokból indított keresések egymástól teljesen függetlennek tekinthetők. Ezek a keresések tehát nyugodtan párhuzamosíthatóak a második módszernél leírtaknak megfelelően, azonban a későbbi átfedések ellenőrzése szükségtelen, hiszen az így indított növelések által visszaadott régiók biztosan nem fognak összenőni (5. ábra).

Ehhez egy valamivel összetettebb keresési algoritmusra van szükségünk, amelyik megadott darab, egymástól megfelelő távolságban lévő induló pontot ad vissza (mindezt persze a GPGPU-n is hatékonyan implementálható adatpárhuzamos módon):

1. Az indítási feltételeknek megfelelő legnagyobb intenzitású képpontok kigyűjtése egy $S_{waiting}$ halmazba (mivel az intenzitást csak 8 biten tároljuk, valószínűleg több ilyen is lesz).

2. A $S_{waiting}$ halmazból kivesszünk egy elemet, és áthelyezzük egy $S_{confirmed}$ halmazba.
3. Az $S_{waiting}$ halmazban megvizsgáljuk a következő elemet, hogy a $S_{confirmed}$ halmazban található elemek közül kizárja-e valamelyik annak párhuzamos feldolgozását (tehát a két pont távolsága nincs-e a kritikus határérték alatt). Amennyiben nincs kizárás, ez az elem is átkerülhet a $S_{confirmed}$ halmazba, egyébként marad a helyén.

   A 3. lépést addig ismételjük, amíg elfogynak az $S_{waiting}$ halmaz elemei, vagy nem találunk már áthelyezhetőt, vagy pedig a $S_{confirmed}$ halmaz megtelik (ennek mérete az egyidőben maximálisan indítani kívánt régiónövelések száma).
4. Elindítjuk a régiónövelő kernelt a $S_{confirmed}$ halmazban található kiindulópontokból.
5. A kernel végrehajtását követően tároljuk az eredményeket, töröljük a $S_{confirmed}$ halmazt, illetve az $S_{waiting}$ halmaz azon elemeit, amelyek így már nem felelnek meg az induló feltételeknek.
6. Ha az $S_{waiting}$ halmazban még maradtak elemek, akkor folytatjuk a 2. lépéstől, ha pedig már üres, akkor az 1. lépéstől a feldolgozást (csökkentve a kiindulópontra vonatkozó minimális intenzitási korlátot).

Nagyméretű képek esetén a szóbajöhető kiindulópontok száma meglehetősen magas, így célszerű lehet itt is kihasználni a GPGPU lehetőségeit. Az algoritmus jól láthatóan egymásra épülő lépéseket tartalmaz, így itt is csak az egy blokkon belüli futtatás nyújt megfelelő szinkronizációs lehetőségeket. Ez korlátozza a szálak számát, így pl. az 1. lépésben egy iterációval minden szálnak több képpontot is meg kell vizsgálnia, hogy azok intenzitása megfelelő-e, és ha igen, akkor az atomi műveletek segítségével el kell helyezni azokat a $S_{waiting}$ halmazba.

Mivel a $S_{waiting}$ halmaz valószínűleg sok elemet tartalmaz, így a következő műveleteknél is célszerű kihasználni az adatpárhuzamos architektúrát, ami persze a hagyományos műveletek újragondolását igényli. Az egyes potenciális kiindulópontokat egy-egy szál kezeli (mivel a pontok száma nagyobb lehet, mint a szálak száma, így egy iteráció segítségével egy szálhoz több pont is tartozhat), és a koordináták kiválogatása több lépcsőben fut le. Minden kiindulópont rendelkezik egy állapottal, ez kezdetben *várakozó*.

1. Első lépésben minden szál megnézi, hogy a hozzá tartozó kiindulópont még használható-e (pl. lehet, hogy az előzőleg lefutott sejtmag növelés során már foglalt lett az általa vizsgált képpont). Amelyik nem, az egy *kizárt* státuszba kerül, a többiek pedig versenybe kerülnek egymással.

2. Ezt követően minden szál ellenőrzi, hogy a hozzá tartozó kiindulópont bekerülhet-e az $S_{confirmed}$ halmazba. Ha igen, akkor közülük az egyik (hogy ki, azt egy versennyel döntik el) elhelyezheti az általa vizsgált pontot az $S_{confirmed}$-be, és a szál állapotát *feldolgozandóra* váltja. Ez az iteráció addig fut le, amíg elfogynak a potenciális kiindulópontok, vagy összegyűlik elegendő elem a hatékony régiónövelés indításhoz.
3. Miután a fenti ciklus véget ért, az első szál a nyertes kiindulópontokat betölti egy $Q_{processing}$ sorba, amelyből majd a régiónövelési eljárás ki tudja venni a koordinátákat. Mivel a kiindulópont keresés és a régiónövelés kernelek meglehetősen különböző végrehajtási paramétereket (blokkok és szálak száma) igényelnek, ezek nem egy, hanem két, egymást felváltva követő kernelben lettek megvalósítva.

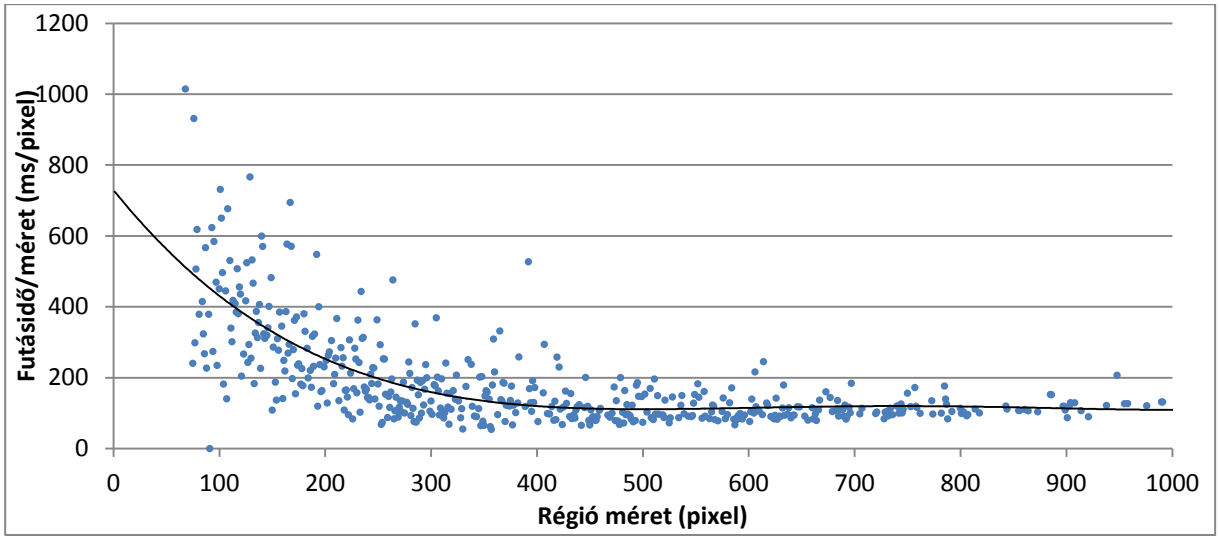
## 2.5. Az algoritmus jósági és hatékonysági vizsgálata

### *2.5.1. Az algoritmus jósági vizsgálata*

Az algoritmus vizsgálatát megkönnyítette, hogy rendelkezésre áll egy CPU-n már implementált, hasonló célokat szolgáló megoldás [**9**]. Az elkészült algoritmus bizonyos beállítások mellett képes lenne a CPU implementációval teljes egészében megegyező eredményt adni (amit a tesztelés során ki is használtunk), azonban néhány GPGPU jellegzetességből adódóan annál valamivel pontosabb eredményt várhatunk. A GPGPU tipikusan lebegőpontos számokkal tud gyorsan számolni, emiatt a CPU algoritmusban esetleg optimalizálási okokból csak egészként kezelt változókat is a nagyobb pontosságú levegőpontos számok váltották fel.

A CPU megvalósításhoz képest szintén jelentős előnyt jelent, hogy a GPGPU implementáció esetében (miként ez a legtöbb GPGPU algoritmusra jellemző) tipikusan nem a számítások mennyisége, sokkal inkább a memóriaműveletek korlátozzák az elérhető teljesítményt. Ezért míg a CPU algoritmusnál célszerű lehet különféle egyszerűsítésekkel élni (pl. ha nem változott jelentősen a régiónövelés során a terület súlypontja és intenzitása, akkor nem érdemes újra számolni a jósági függvényt az összes kontúrpontra), addig a GPU esetén ezekre nincs szükség, a rendelkezésre álló nagy számítási kapacitás mellett ezek a számítások minden iterációban egyszerűen elvégezhetők. Az egyszerűsítések nélkül a végeredmény valamivel pontosabbnak tekinthető.

A pontosság vizsgálata a már előzőleg kidolgozott kiértékelő algoritmus alapján történt. Ennek részletes adatait az előző fejezet tartalmazza, itt most főként a sebességkülönbséget szeretném bemutatni.




**1. diagram:** *Egy pixel feldolgozási sebessége a régió méret függvényében a GPGPU alapú megvalósítás esetén.*

A vizsgálat 46 rendelkezésre álló minta alapján történt[2] (ezek méret szerint 1024x1024, 2048x2048, 4096x4096 méretűek, tartalmukat tekintve pedig találhatóak benne ép szövetek, beteg szövetek illetve rosszul fókuszált hibás képek).

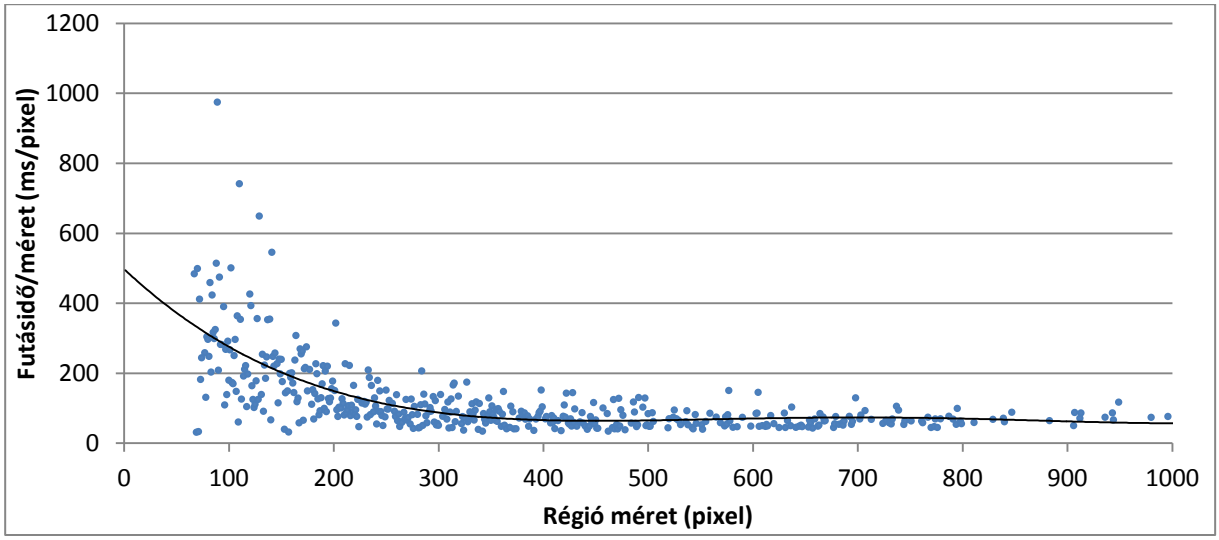
### *2.5.2. Futásidő vizsgálata blokkméret alapján*

A GPGPU-k működési elvéből adódóan egy darab végrehajtóegység teljesítménye messze elmarad egy CPU mag teljesítményétől, a teljesítmény kiaknázásának kulcsa a végrehajtóegységek nagy számának kihasználása. Az algoritmus erre természetesen lehetőséget ad, hiszen minden kontúrponthoz egy saját szálat rendelünk, viszont a szálak pontos számának meghatározása kritikus kérdésként merül fel: a párhuzamosság kihasználása érdekében minél több szálat kell indítanunk, azonban a GPGPU hardver sajátossága az is, hogy ha egy blokk túl sok szálat (és ezzel együtt túl sok egyéb erőforrást) igényel, akkor az csökkenti az egy multiprocesszor által futtatható blokkok számát.

Emiatt célszerű megvizsgálni, hogy a GPU teljesítményét milyen méretű régiók esetén lehet a legjobban kihasználni. Ez egyrészt lehetőséget ad az algoritmus finomhangolására (ennek megfelelő mennyiségű szálat rendelni egy sejtmag kereséséhez), illetve egy jó ajánlásnak tekinthető, hogy milyen méretűre célszerű skálázni a bemeneti képet.

Ehhez teszteket futtattam és több futás adatait összegezve megmértem (1. diagram), hogy adott méretű régiókat mennyi idő alatt tudott feldolgozni a GPU.

[2] Ezek nem azonosak az előző fejezetben már bemutatott „gold standard” mintákkal. Ott ugyanis az volt a fő szempont, hogy olyan mintákat kerestem, amelyeknél rendelkezésre állt egy referencia eredmény (manuális annotációk) a pontossági vizsgálathoz, itt pedig az volt a cél, hogy minél változatosabb tartalmú és méretű mintákon tudjam összehasonlítani a CPU és a GPGPU alapú régiónövelés teljesítményét.




**2. diagram:** *Egy pixel feldolgozási sebessége a régió méret függvényében a CPU alapú megvalósítás esetén.*

Az egyre nagyobb régiók értelemszerűen egyre több időt igényeltek, a mérés szempontjából emiatt szerencsésebb egy relatív mutatót vizsgálni, hogy az egyes régióméretek esetén pixelre vetítve mennyi ideig tartott a feldolgozás.

A diagramon jól látható, hogy kisméretű régiók esetén a pixelenkénti költség meglehetősen magas, a régiók növelésével ez azonban gyorsan csökken. A tesztelt beállítások mellett kb. 450 képpont méretű régiók mellett ez a költség elérte a minimumát, ezt követően már nem csökkent jelentősen. A legrosszabb (becsült ~800ms/pixel) időeredményekhez képest az itt mérhető ~100ms/pixel érték jelentős, 800%-os gyorsulást jelent. Tehát ahhoz, hogy hatékonyan kihasználhassuk a GPGPU teljesítményét, célszerű úgy skálázni a bemenő képet, hogy nagyságrendileg ekkora, vagy ennél nagyobb sejtmagokra számíthassunk.

A teljesítmény vizsgálatakor célszerű lehet a GPU implementáció teljesítményét összehasonlítani a CPU megvalósítás teljesítményével (2. diagram).

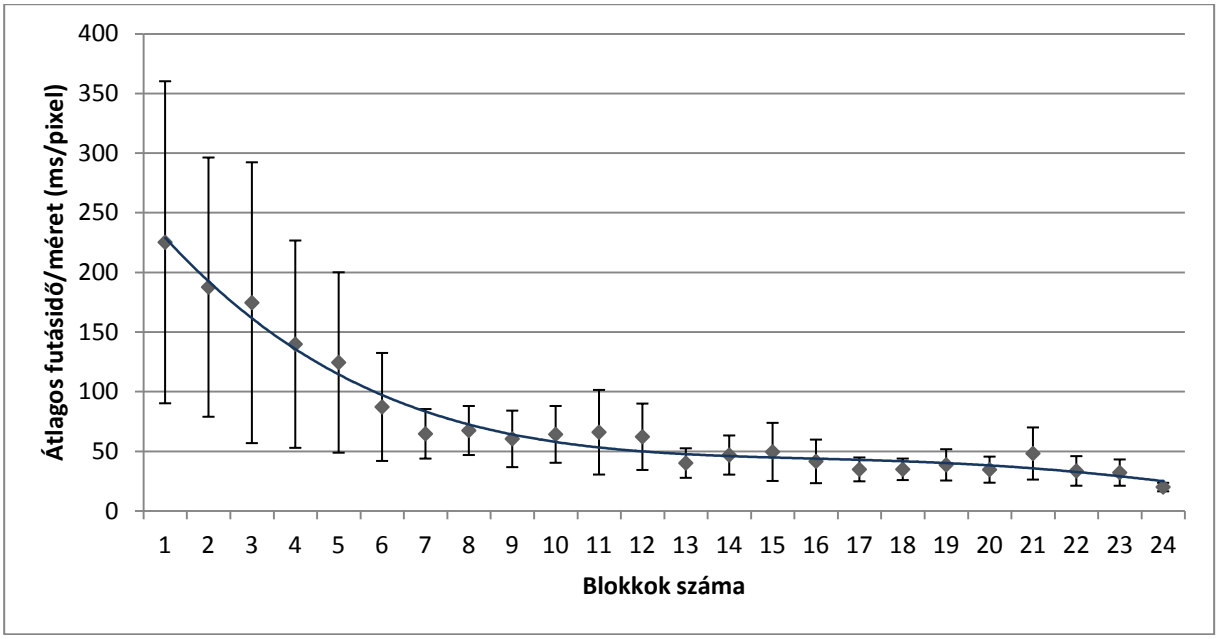
Ahogy az várható volt, a CPU esetében a kisebb régióméretek esetén a futásidő jóval kevesebb lett, és mivel a CPU-nak nincsenek jelentős tartalékai a nagyobb régióméretek során sem, a görbe gyorsan eléri az ideális állapotot, körülbelül 300 képpont méretű régiók esetén már optimális a kihasználtsága. A GPGPU számára ennél sokkal nagyobb régiókra (és ezzel együtt jóval nagyobb szál mennyiségre) van szükség ahhoz, hogy ezek együttes teljesítménye megközelítse a CPU-ét. Az mindenesetre jól látható az ábrán is, hogy a CPU viszonylagosan nagy előnyét (GPU ~800ms/pixel – CPU ~500ms/pixel) a GPGPU erősen megközelíti, de meghaladni azt már nem tudta (GPU ~100ms/pixel - CPU ~80ms/pixel).

| Blokkméret | Előfordulások száma | Futásidő (sec) | Futásidő/méret (ms/pixel) | Szórás (ms/pixel) |
|---|---|---|---|---|
| 1 | 27 | 743 | 237,86 | 134,99 |
| 2 | 29 | 1843 | 187,61 | 108,68 |
| 3 | 44 | 4202 | 174,56 | 117,72 |
| 4 | 43 | 3409 | 139,80 | 86,92 |
| 5 | 46 | 3841 | 124,45 | 75,63 |
| 6 | 40 | 3949 | 87,17 | 45,27 |
| 7 | 40 | 4522 | 64,64 | 20,77 |
| 8 | 38 | 4022 | 67,43 | 20,55 |
| 9 | 37 | 4286 | 60,40 | 23,68 |
| 10 | 43 | 4942 | 64,17 | 23,81 |
| 11 | 45 | 5650 | 65,97 | 35,48 |
| 12 | 35 | 4716 | 62,17 | 27,82 |
| 13 | 41 | 5083 | 40,13 | 12,37 |
| 14 | 34 | 4467 | 46,84 | 16,42 |
| 15 | 31 | 4449 | 49,49 | 24,36 |
| 16 | 35 | 4626 | 41,54 | 18,26 |
| 17 | 33 | 4621 | 34,81 | 9,99 |
| 18 | 36 | 5365 | 34,89 | 9,01 |
| 19 | 40 | 6167 | 38,60 | 13,11 |
| 20 | 40 | 6063 | 34,59 | 10,94 |
| 21 | 24 | 3861 | 48,13 | 21,87 |
| 22 | 31 | 4627 | 33,52 | 12,40 |
| 23 | 43 | 7416 | 32,14 | 11,02 |
| 24 | 4939 | 855441 | 19,99 | 3,57 |

**3. táblázat:** *GPGPU alapú régiónövelés eredményei a blokkméret függvényében.*

### *2.5.3. Futásidő vizsgálata blokkok száma alapján*

Elsőre csalódást okozhat az eredmény, hiszen a GPGPU hiába rendelkezik óriási csúcsteljesítménnyel, az algoritmus sajátosságai miatt (szinkronizációk és kiegészítő műveletek) nem tudta elérni a CPU teljesítményét. Azonban a fenti értékek csak egy-egy régió növekedésének vizsgálatát mutatják, ami a jelenlegi implementációban tulajdonképpen egy blokk működését jelentik. Amennyiben a bemenet megengedi (található megfelelő számú, egymástól távol eső, azonos intenzitású kiindulópont) akkor lehetőség nyílik egyidőben több blokk és ezzel több régiónövelés indítására is. Emiatt egy újabb mérést végeztem (3. táblázat).



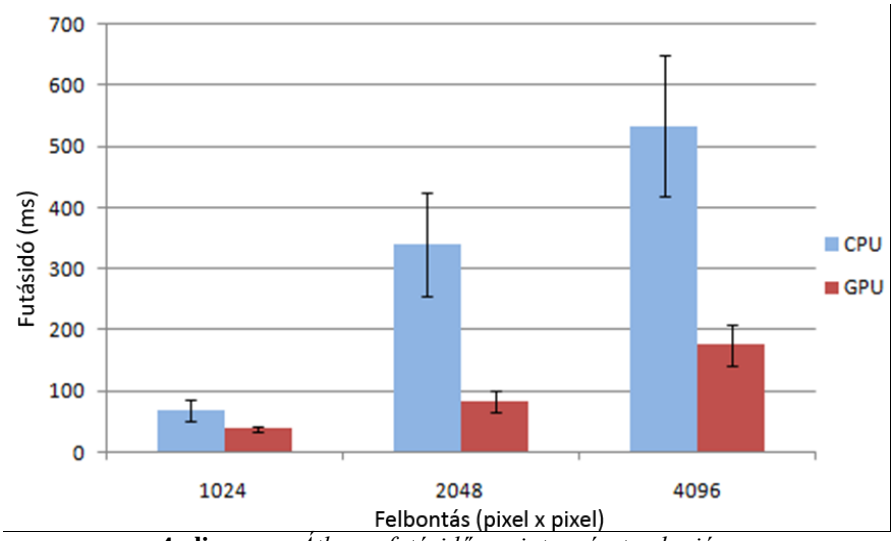

**3. diagram**: *Egy pixel átlagos feldolgozási ideje a párhuzamos blokkok számának függvényében.*

Azonban most nem a régiók mérete alapján mértem a teljesítményt, hanem az alapján, hogy a GPGPU egyidőben hány régión tudott dolgozni.

A teszt során 27 esetben csak 1 blokkot tudott indítani az ütemező (nem talált vele párhuzamosan indítható kiindulópontokat), az itt mért idő (237,86ms/pixel) megfelel az előzőekben megismert méréseknél is látható átlagnak, ez mint láttuk, valamivel elmarad a CPU idejétől.

Jól látható azonban (3. diagram), hogy amint sikerült a blokkok számát növelni, jelentősen csökkenni kezdett az egy pixelre eső végrehajtási idő. Ez a csökkenés a 14. párhuzamos blokk futtatásáig észlelhető, itt már elérte a GPU a teljesítőképessége határát, további blokkok indítása már nem befolyásolja jelentősen a mért időket. Az itt mérhető 33 ms/pixel eredmény pedig messze meghaladja a CPU legjobb eredményét is.

Az esetek nagy részében az egyidőben több blokk futtatása volt jellemző, emiatt a mért adatok szórása ezekben az esetekben már láthatóan jóval alacsonyabb volt, mint a néhány blokkot mutató esetekben. A gyakorlati tapasztalatok egyébként azt mutatják, hogy a közepes és nagyobb méretű képeknél nagyon ritkák azok az esetek, amikor nem lehet egyidőben több régiónövelést indítani, ez egyértelműen kedvez az adatpárhuzamos megoldásnak. Amikor ez mégis bekövetkezik, az is főként amiatt történik, hogy egyidőben csak azonos intenzitású kiindulópontokat válogatunk ki, és ezek egy idő után értelemszerűen elfogynak, így néha már csak emiatt is szükség van kevesebb blokk indítására.




**4. diagram:** *Átlagos futásidő a minta mérete alapján.*

### *2.5.4. Futásidő vizsgálata képméret alapján*

Bár a fenti értékek is lényegesek a későbbi optimalizálás számára, a gyakorlat szempontjából a leglényegesebb kérdés a teljes képfeldolgozási idejének analízise. Ehhez különféle méretű és tartalmú képekre futattam le a CPU és a GPGPU implementációt, majd kiszámoltam az azonos méretű minták esetében a futásidő átlagát (4. diagram).

A háromféle méretbe sorolható minták alapján (1024x1024, 2048x2048 és 4096x4096 pixel felbontásban) látható, hogy kisebb méret esetében még nem jelentős a GPGPU előnye, ez azonban már a 2048 pixeles méretnél egyértelműen megjelenik. A növekedés természetesen nem növekszik folyamatosan a minta méretével, mivel már egy közepes méretű kép esetében is mindig találunk elég kiindulópontot a szükséges párhuzamos régiónövelés indításához.

### *2.5.5. Mérési módszerek*

A futásidő mérése mindkét esetben a CPU segítségével történt, a *QueryPerformanceCounter* függvény segítségével. Ennek felbontása megközelítőleg 0,366 µs, az időzítő indítása illetve leállítása pedig megközelítőleg 3 µs hibát okoz a mérés során. A mért eredményeket tekintve nyilvánvaló, hogy ezek a pontatlanságok elhanyagolhatók. A mérések minden esetben három független mérés átlagát mutatják.

Továbbá a mérések esetén a legelső kernel indítás idejét (függetlenül attól, hogy az hány blokk indítását igényelte) nem vettük figyelembe, mivel elképzelhető, hogy a GPGPU illetve az azt kezelő driver számára az első induláskor további költségek merültek fel, amelyek eltorzítanák az eredményeket. Hasonló mérési metodikát követtünk a későbbi tesztek során is.

## 2.6.Eredmények értékelése

A fejlesztés fő célja egy új, adatpárhuzamos környezetben futtatható algoritmus kialakítása, amelynek kimenete megegyezik vagy esetleg valamivel pontosabb a már meglévő szekvenciális algoritmushoz. A kutatás jelenlegi fázisában az algoritmus és annak implementációja elkészült, így kimondható, hogy sikerült kidolgoznom mind az adatpárhuzamos régiónövelési eljárást, amely alkalmas szöveti mintákon párhuzamosan több kiindulópontból kiindulva sejtmagok detektálására, mind pedig az ehhez kapcsolódó adatpárhuzamos kiindulópont keresési eljárást, amely alkalmas a régiónövelési eljárás számára egyidőben több kezdőpont kigyűjtésére. Az előző fejezetben és az itt részletezett vizsgálatokkal pedig igazoltam, hogy az újonnan kifejlesztett adatpárhuzamos régiónövelési eljárás (kiindulópont keresés és régiónövelés) a jelenleg meglévő hagyományos régiónövelési eljárásokhoz képest azonos pontosságot ér el jelentősen kisebb futásidő mellett.

A meglévő algoritmus rendelkezik néhány korláttal, amiket érdemes lehet kiküszöbölni egy esetleges továbbfejlesztés esetén. Ilyen a már említett kontúrpontok számára, illetve a régió maximális méretére vonatkozó megszorítás. Mindkét esetben vannak különféle lehetőségek ezek kiküszöbölésére, azonban ezek minden esetben a teljesítmény csökkenésével járnak, emiatt a továbbfejlesztés előtt célszerű számba venni azok előnyeit, illetve hátrányait.

Amennyiben a továbbfejlesztés fő céljának a futásidő csökkentését tekintjük, mindenképpen érdemes kialakítani a fenti algoritmusnak egy több GPGPU-t is kezelni képes változatát (kiegészítve akár azzal a lehetőséggel, hogy velük egyidőben a CPU magjai is végezzenek szinkronizációs, vagy akár konkrét régiónövelési feladatokat). Technikailag ez egyszerűen megvalósítható, azonban felvet számos, megoldásra váró problémát. Az egyes GPGPU eszközök ugyanis nem látják egymás memóriáját, így a párhuzamosságot még egy szinttel magasabbra kell tolni, illetve ezzel együtt biztosítani kell a hatékony adatcserét és szinkronizációt az egyes eszközök között.

# 3. RÉGIÓNÖVELÉS PARAMÉTEREINEK OPTIMALIZÁLÁSA

## 3.1.Paraméter optimalizálás lehetőségei

Az előzőekben bemutatott régiónövelési algoritmus beváltotta a hozzá fűzött reményeket, finomhangolása során azonban újabb megoldandó feladatok merültek fel. A régiónövelési algoritmus lépései (előfeldolgozás, régiónövelés, összenőtt sejtek szétvágása) meglehetősen sok (az általunk használt implementációban 27 darab) paramétert igényelnek, amelyek pontos megválasztása esetén várhatjuk csak el, hogy a módszer a lehető legnagyobb hatékonysággal működjön.

Ezeknek a paramétereknek az egymásra gyakorolt hatása ismeretlen, így célszerűen függetlennek tekintjük őket. Minden paraméter jelentősen befolyásolja az algoritmus kimenetét, így egy optimális paraméterkészlet meghatározásához mindenképpen az összes paraméter együttes kezelésére van szükség. Azok nagy száma, illetve azok meglehetősen nagy értékkészlete miatt ez manuálisan véges időn belül reménytelennek tűnik, ezért ki kellett, hogy dolgozzak egy újszerű algoritmust egy, a gyakorlatban is jól használható paraméterkészlet megtalálására [**46**].

Maga a alapprobléma meglehetősen egyszerűen modellezhető, adott 27 darab, egymástól függetlennek tekinthető paraméterünk megadott értékkészlettel, adott egy algoritmus (és annak implementációja) ami szövetminták alapján a megadott paraméterekkel előállítja a mintában található sejtmagok listáját (méret, elhelyezkedés stb.), illetve adott egy kiértékelő függvény [**47**], amely a rendelkezésre álló, orvosok által manuálisan annotált minták (a már említett „gold standard” minták) és az előző régiónövelés által adott eredmény összehasonlítása alapján meghatároz egy pontossági értéket. A cél egy olyan paraméterkészlet megkeresése, amely az elérhető legnagyobb pontosságot eredményezi. Ez egy klasszikus optimalizációs feladat [**48**], ennek megfelelően számos megoldási lehetőséget találhatunk [**49**]. A különféle keresési módszerek alapvetően három kategóriába sorolhatóak [**50**]: *véletlen, leszámlálási* és *számítás alapú*.

### *3.1.1. Leszámlálási módszerek*

Ezek közül a *leszámlálási* (*enumerative*) módszerek alapja, hogy megvizsgálják az összes lehetséges megoldást, majd azok közül kiválasztják a legjobbat [**51**]. A módszert egyszerűen tudnánk adaptálni erre a feladatra, csak fel kell sorolnunk az összes lehetséges paraméter kombinációt (bár itt is felmerül a kérdés, hogy az egyes intervallumokat milyen

szemcsézettséggel bontsuk fel diszkrét értékekre), majd ezek között egyszerű lineáris kereséssel megtalálhatjuk az optimálist.

Ez ugyan garantáltan megtalálja a legjobb megoldást, azonban a lehetséges kombinációk nagy száma miatt a módszer számunkra a gyakorlatban használhatatlan. Bár a lehetőségek számát valamilyen szinten tudjuk befolyásolni (önkényesen az értékkészletek szűkítésével, illetve az intervallumok szemcsézettségének csökkentésével), egy, a gyakorlatban jól használható konfiguráció mellett valószínűleg évekig tartana a kiértékelés. És mivel nincsenek egzakt információink az egyes paramétereknek a végeredményre, illetve egymásra való hatásáról, így különféle heurisztikák segítségével sem tudjuk jelentősen leszűkíteni a rendelkezésre álló keresési teret.

### *3.1.2. Számítás alapú módszerek*

Másik fontos módszercsoport a *számítás alapú* (*calculus-based)* módszerek csoportja, ahol a meglehetősen nagy keresési tér még nem jelent feltétlenül nagy számításigényt, hiszen nincs szükség a teljes tér átvizsgálására. Ezekben az esetekben ugyanis elindulhatunk egy tetszőleges pontból, majd az algoritmus minden egyes kiértékelés után abba az irányba folytatja majd a keresést, amely az adott pillanatban a legnagyobb azonnali előnnyel jár (ennek klasszikus példái a különféle *mohó algoritmusok* (*greedy*) [**52**][**53**], illetve a *hegymászó* (*hill-climbing*) [**54**] módszerek).

Természetesen ezt a módszert is tudjuk implementálni erre az esetre, kiinduló pontként választhatjuk a paramétertér egy tetszőleges pontját, ezt követően megvizsgálhatjuk annak a környezetét, majd ezek közül a legnagyobb pontosságot ígérő pontra léphetünk, és ott folytathatjuk a vizsgálatot. Ezek a mohó algoritmusok meglehetősen gyorsan találnak eredményt, azonban alapesetben csak azt tudjuk garantálni, hogy a megtalált paraméterkészlet lokális optimum lesz, annak globális optimum mivoltáról nem lesz információnk. Mivel a paraméterek egymáshoz lazán kapcsolódnak, így nagyon valószínű, hogy sok lokális maximumot találhatunk (olyan pontokat, ahol a paraméterkészlet egy viszonylag jó eredményt ad, és bármelyik paraméter apró módosításával annál rosszabbat kapunk). A módszer hátránya, hogy nincs benne visszalépési lehetőség, tehát ha egyszer a keresés befutott egy ilyen lokális maximumba, onnan már csak egy másik pontból való újraindítással juthatunk másik eredményhez.

A keresés érzékeny arra, hogy milyen paraméterekkel indítottuk el, így természetesen lehet finomhangolni az eredményt, ha egymást követően több keresést indítunk el különféle

helyekről, majd ezek eredményét hasonlítjuk össze, azonban a paraméterek nagy számából adódóan sokféle kezdőpozíció létezik, és véges időn belül ezeknek is csak egy nagyon kis részét tudnánk átvizsgálni. További problémát jelent a paraméterek meglehetősen nagy száma, emiatt ugyanis még egy pont környezetének a vizsgálata is meglehetősen számításigényes lehet, ha az adott pontból minden lehetséges irányba meg szeretnénk vizsgálni az arra vezető utak eredményességét (sőt, a paraméterek nagy száma miatt valószínűleg egyszerűsítésekre lenne szükség, így még a hegymászó algoritmus előnye, az, hogy biztosan megtalálja a lokális maximumot, is elvész).

### *3.1.3. Irányított véletlen keresési módszerek*

A keresések harmadik csoportja a *véletlen keresési* (*random search)* módszerek csoportja, ami szigorúan nézve pusztán annyit jelent, hogy a problématérből véletlenszerűen választott pontokat vizsgálunk meg, és ezek közül mindig eltároljuk a legjobb eredményt adót. Önmagában ez persze kevésbé lenne hatékony számunkra, hiszen a futásideje meglehetősen nagy lehet, és nem ad semmiféle garanciát arra sem, hogy valamiféle lokális vagy globális optimumot eredményezne véges időn belül. A gyakorlatban azonban már jól használhatóak az *irányított véletlen keresési* (*guided random search)* technikák [**55**], amelyek bár alapvetően véletlenszerűen választott pontokat vizsgálnak, azonban a kiválasztás módját különféle heurisztikák segítségével próbálják finomhangolni.

A módszer klasszikus képviselői a *tabu keresés* (*tabu search*) [**56**], a *szimulált hűtés* (*simulated annealing*) [**57**] és az *evolúciós algoritmusok* (*evolutionary algorithms*) [**55**][**58**]. Ezek közül számunkra főleg a különféle evolúciós algoritmusok az érdekesek. Ezen módszerek alapja, hogy a biológiából kölcsönvett mechanizmusok segítségével próbálnak különféle kereséseket vagy optimalizációkat végrehajtani. Ezen algoritmusok alapelve mindig az, hogy egy kiinduló populációból (ahol az egyes egyedek alatt jelen esetben régiónövelési paraméterkészleteket értünk) mindig újabb és újabb generációkat állítunk elő, amelynek tagjai várhatóan egyre inkább életképesebbek lesznek (ami ebben az esetben annyit jelent, hogy egyre nagyobb pontosságot biztosító paraméterkészleteket fognak reprezentálni). Kapcsolhatunk hozzá különféle leállási feltételeket is, de a módszer enélkül is használható, hiszen az egyes generációk már futás közben is láthatók, így az aktuálisan legjobbnak tűnő paraméterkészlet mindig elérhető.

A mi esetünkben ez a módszer tűnik a legkézenfekvőbbnek, ugyanis a feladat teljesíti az alábbi feltételeket, amelyek előtérbe helyezik a genetikus algoritmus használatát:

- Sokdimenziós keresési tér, ahol a fontosabb változók közötti összefüggések ismeretlenek.
- Tradicionális megoldások elfogadhatatlan futási időt adnak.
- A keresési tér nehezen, vagy nem szűkíthető.
- Egy megoldás gyorsan ellenőrizhető, de egy tökéletes megoldás előállítása nehézkes.
- Nincs feltétlenül szükségünk a globális optimumra, csak egy minél jobb megoldást keresünk.

Emellett a genetikus algoritmusok számos további előnnyel kecsegtetnek:

- Jól párhuzamosíthatók, így többprocesszoros környezetben hatékonyan implementálhatók. Mivel bármelyik módszert is választjuk, az valószínűleg meglehetősen nagy számításigénnyel rendelkezik majd (ugyanis a keresés során minden paraméterkészletre végre kell hajtani a régiónövelést és a kiértékelést is), így lényeges, hogy ezt elosztott környezetben lehetőleg minél rövidebb idő alatt le tudjuk futtatni.
- A hegymászó módszerhez viszonyítva nagyobb eséllyel talál globális maximumot, mivel működéséből adódóan egy esetleges lokális maximum megtalálása esetén is folytatja a keresést, még egy stabil populáció esetén is a mutációk miatt folyamatos a változás. Emellett, bár nem tudja garantálni, hogy elfogadható időn belül meg fogja találni a globális maximumot, mégis számos tanulmány szól amellett, hogy maga a keresés általában egy jó eredmény felé konvergál [**59**][**60**][**61**].
- A témával foglalkozó irodalmat áttanulmányozva számos genetikus algoritmus alkalmazást találhatunk paraméter optimalizációkra, és ezek általában be is váltották a hozzájuk fűzött reményeket [**62**][**63**].

## 3.2.A genetikus algoritmus indításához szükséges paraméter intervallumok

A genetikus algoritmus számára azonban szükség van egy induló populációra, ezeket az egyedeket fel kell tölteni valamilyen kezdő génekkel (paraméterekkel), emiatt szükség van minden paraméter esetén egy közelítő becslésre, vagy legalább a ténylegesen vizsgálandó intervallum meghatározására. A paraméterek egy része kimondottan technikai, ezek értékét csak becsülni, illetve empirikus módszerekkel finomhangolni lehet, ilyenek például a különféle szűrők által használt ablakméretek stb. A paraméterek egy része azonban a gyakorlatban is jól mérhető, így azokra a „gold standard” minták alapján pontos határokat lehet meghatározni.

A vizsgálathoz készítettem egy alkalmazást, ami az alábbi algoritmus szerint működik:

1. Következő „gold standard” minta megnyitása.
2. A minta annotációnak betöltése, ezek közül az első terület az orvosok által annotált régiót azonosítja.
3. A maszk a használata a teljes képre, az ezen kívüli területek vizsgálata szükségtelen.
4. A többi annotáció az orvosok által megjelölt sejtmagokat jelöli, ezeket egyesével meg kell vizsgálni az összes vizsgálati szempont szerint.
5. Az így nyert statisztikai adatok elmentése egy adatbázisba.
6. Az adatbázis alapján a paraméter optimalizációhoz szükséges határértékek kinyerése.

Ennek megfelelően a fenti paraméterekre vonatkozóan meghatároztam az orvosok által megjelölt sejtmagok esetében megtalálható minimumot illetve maximumot, az így kapott intervallumon belül kell majd keresnünk az optimális paraméter értékeket. A gyakorlatban azonban nem ezeket a határokat használtam a későbbi genetikus algoritmus számára a kezdő generáció létrehozásánál, ugyanis sok esetben ez a határ indokolatlanul tág volt, egy-két szerencsétlenül elhelyezkedő sejtmag miatt jelentősen ki kellett volna bővíteni a határokat, ami a keresés idejének elhúzódását vonta volna maga után.

Emiatt a minimum és maximum értékek mellett megvizsgáltam a paraméterek eloszlását, és egy újabb segédprogrammal megkerestem azokat a legrövidebb intervallumokat, amelyekbe beleesik az összes elem 99,5%, 99%, 95%-a. Az optimalizáció során a genetikus algoritmus ezekkel a szűkebb intervallumokkal dolgozott, ami persze egyrészt már elvileg is megakadályozza, hogy tökéletes megoldást találjon, viszont cserébe jelentősen felgyorsult.

### *3.2.1. Sejtmagok méretének vizsgálata*

A régiónövelési algoritmus működése során az egyik legfontosabb információ a sejtmagok mérete. Ez az algoritmus működése során több helyen is szerepet játszik:

- A konkrét régiónövelés során minden egyes új iteráció során 1 pixellel növekszik a megtalált sejtmag jelölt mérete. A növelésnek számos korlátot szabhatunk, ezek közül az egyik, hogy maga a sejtmag nem lehet egy bizonyos méretnél nagyobb. Amennyiben a régiónövelés eléri ezt a maximális méretet, akkor a régiónövelési ciklus leállítható.
- A sejtmagok mérete szintén lényeges paraméter a régiónövelés leállítását követő utólagos ellenőrzés során. Miután a előállt a sejtmag jelölt, ezt különféle szempontok

alapján ellenőrizni kell, hogy valóban megfelel-e a szükséges feltételeknek, ezen feltételek között szerepel a sejtmag minimális mérete (azokat a sejtmagokat, amelyekben található pixelek száma nem éri el ezt a korlátot, elvetjük) és a sejtmagok maximális mérete (a túl nagyokat szintén elvetjük, bár ilyen a régiónövelésnél már leírt korlátozás miatt nem fordulhat elő).

A sejtmagok méretének vizsgálatához megvizsgáltam a rendelkezésre álló „gold standard" mintákat, azokból kiválogattam az összes annotált sejtmagot. Minden egyes sejtmagot maszkként használva, azokat felrajzoltam egy üres képre, majd megszámoltam az ezen a képen kirajzolt pixelek számát. Ezzel megkaptam az egyes sejtmagok méretét. Egy adatbázisban elmentettem ezt az értéket minden egyes sejtmagra, az így nyert 7889 sejtmag adatait pedig osztályoztam (4. táblázat).

| # | Intervallum kezdete | Intervallum vége | Sejtmagok száma | Arány |
|---|---|---|---|---|
| **1** | 22,00 | 124,80 | 499 | 6,33% |
| **2** | 124,80 | 227,60 | 2408 | 30,52% |
| **3** | 227,60 | 330,40 | 2599 | 32,94% |
| **4** | 330,40 | 433,20 | 1454 | 18,43% |
| **5** | 433,20 | 536,00 | 512 | 6,49% |
| **6** | 536,00 | 638,80 | 214 | 2,71% |
| **7** | 638,80 | 741,60 | 87 | 1,10% |
| **8** | 741,60 | 844,40 | 59 | 0,75% |
| **9** | 844,40 | 947,20 | 33 | 0,42% |
| **10** | 947,20 | 1050,00 | 14 | 0,18% |
| **11** | 1050,00 | 1152,80 | 5 | 0,06% |
| **12** | 1152,80 | 1255,60 | 2 | 0,03% |
| **13** | 1255,60 | 1358,40 | 1 | 0,01% |
| **14** | 1358,40 | 1461,20 | 0 | 0,00% |
| **15** | 1461,20 | 1564,00 | 1 | 0,01% |
| **16** | 1564,00 | 1666,80 | 0 | 0,00% |
| **17** | 1666,80 | 1769,60 | 0 | 0,00% |
| **18** | 1769,60 | 1872,40 | 0 | 0,00% |
| **19** | 1872,40 | 1975,20 | 0 | 0,00% |
| **20** | 1975,20 | 2079,00 | 1 | 0,01% |

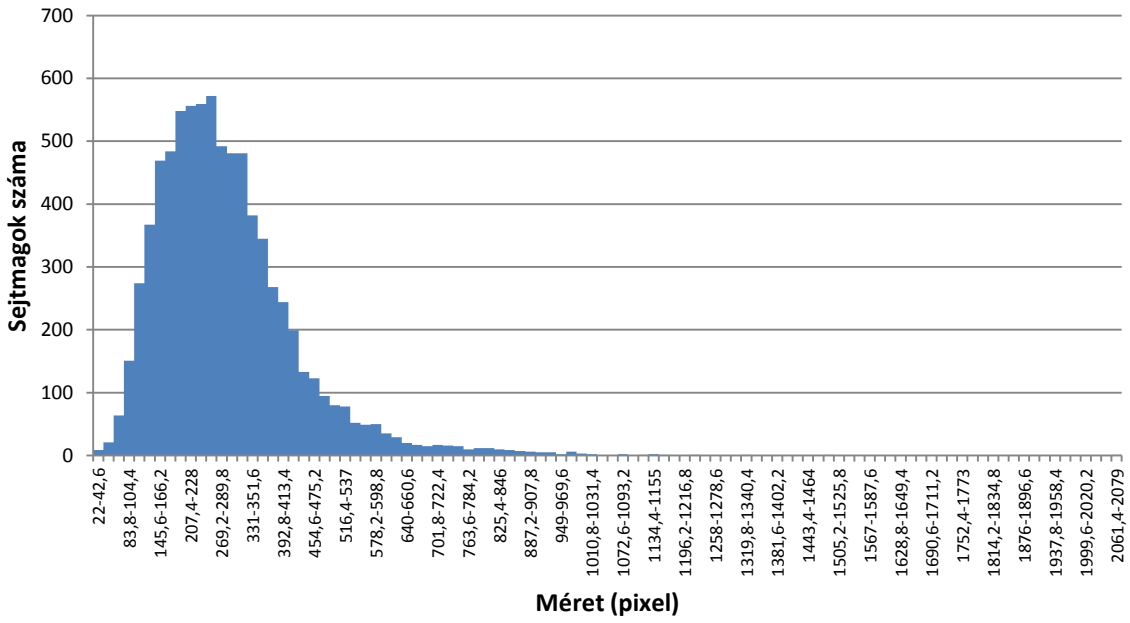
**4. táblázat:** *Sejtmagok száma méret alapján.*

A feldolgozott adatok alapján az alábbi határértékeket állapítottam meg::

- Legkisebb sejtmag mérete: 22 pixel
- Legnagyobb sejtmag mérete: 2079 pixel

A táblázatból is jól látható, hogy bár a legnagyobb sejtmag valóban 2079 pixel méretű, azonban ez jelentősen eltér az átlagos értéktől. Az adatokat diagramon ábrázolva látható (5. diagram), hogy az eloszlás messze nem egyenletes.

A méret esetében a sejtmagok 99,5%-a 34 pixel és 882 pixel méretűek között található (5. táblázat). A későbbi keresést tehát ebben az intervallumban érdemes folytatni.




**5. diagram:** *Sejtmagok száma méret alapján.*

| Lefedettség | Intervallum kezdete | Intervallum vége | Intervallum hossza | Int. hossz/egész |
|---|---|---|---|---|
| **100,0%** | 22,00 | 2079,00 | 2057,00 | 100,00% |
| **99,5%** | 34,00 | 882,00 | 848,00 | 41,23% |
| **99,0%** | 34,00 | 801,00 | 767,00 | 37,29% |
| **95,0%** | 65,00 | 556,00 | 491,00 | 23,87% |
| **90,0%** | 90,00 | 475,00 | 385,00 | 18,72% |
| **80,0%** | 108,00 | 397,00 | 289,00 | 14,05% |

**5. táblázat:** *Sejtmagok száma méret alapján.*

| # | Intervallum kezdete | Intervallum vége | Sejtmagok száma | Arány |
|---|---|---|---|---|
| **1** | 2,69 | 4,19 | 8 | 0,10% |
| **2** | 4,19 | 5,69 | 49 | 0,62% |
| **3** | 5,69 | 7,19 | 272 | 3,45% |
| **4** | 7,19 | 8,69 | 816 | 10,34% |
| **5** | 8,69 | 10,19 | 1543 | 19,56% |
| **6** | 10,19 | 11,69 | 1819 | 23,06% |
| **7** | 11,69 | 13,19 | 1475 | 18,70% |
| **8** | 13,19 | 14,69 | 858 | 10,88% |
| **9** | 14,69 | 16,19 | 506 | 6,41% |
| **10** | 16,19 | 17,69 | 251 | 3,18% |
| **11** | 17,69 | 19,19 | 130 | 1,65% |
| **12** | 19,19 | 20,69 | 79 | 1,00% |
| **13** | 20,69 | 22,19 | 44 | 0,56% |
| **14** | 22,19 | 23,69 | 25 | 0,32% |
| **15** | 23,69 | 25,19 | 6 | 0,08% |
| **16** | 25,19 | 26,69 | 2 | 0,03% |
| **17** | 26,69 | 28,19 | 3 | 0,04% |
| **18** | 28,19 | 29,69 | 2 | 0,03% |
| **19** | 29,69 | 31,19 | 0 | 0,00% |
| **20** | 31,19 | 31,93 | 1 | 0,01% |

**6. táblázat:** *Sejtmagok száma sugár alapján.*

### *3.2.2. Sejtmagok sugarának vizsgálata*

A pixelenkénti méret csak a felismert objektum méretéről ad információt, a sugár azonban valamilyen szinten már az alakjáról is. Emiatt érdemes vizsgálni az egyes sejtmagok sugarát is, ez a régiónövelés során az alábbi pontokban lehet lényeges adat:

- Ha a régiónövelés során elérünk egy maximális sugár értéket, akkor a régiónövelést azonnal leállíthatjuk. A sugár számítás módjából adódóan ugyan elképzelhető, hogy ez az érték a későbbiekben akár még csökkenhet is (ha eltolódik az aktuálisan vizsgált régió középpontja), a gyakorlatban azonban ez nem lényeges, sokkal fontosabb, hogy le tudjuk zárni a felesleges régiónövelési lépéseket.

A régiónövelés leállítása után szintén lényeges paraméter a régió aktuális sugara, ugyanis ez szintén egy szűrési feltételként szerepel majd a sejtmag végső elfogadása során. Bevezethető egy minimális sugár érték (ennél kisebb sugárral rendelkező sejtmag jelölteket el kell vetni),

illetve maximális sugár érték (bár a régiónövelésbe épített feltétel miatt ezt biztosan nem fogja meghaladni egy sejtmag sem).

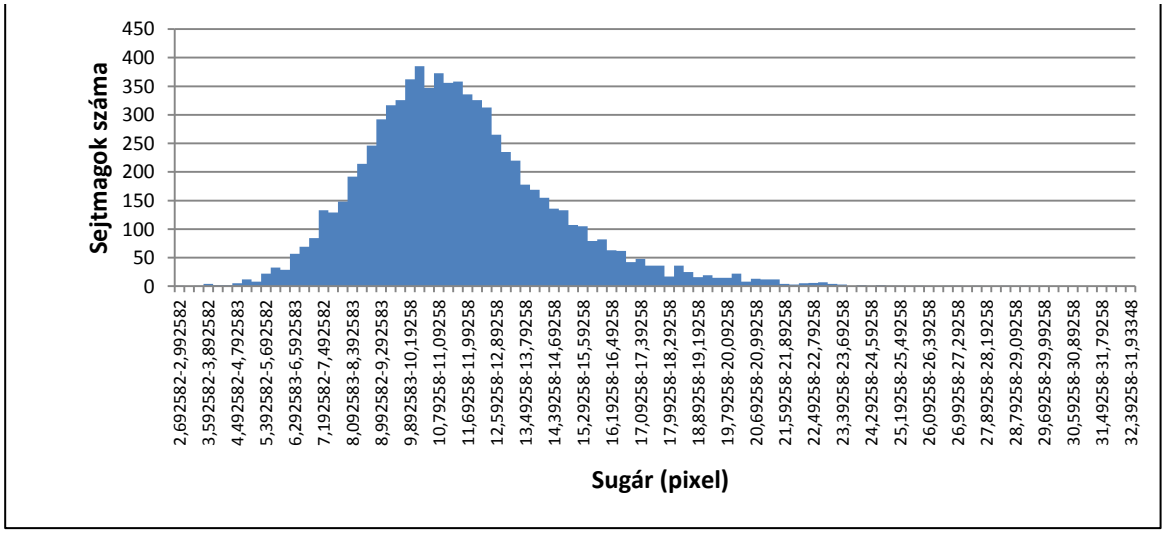
A sejtmagok sugarának vizsgálatához megvizsgáltam a rendelkezésre álló „gold standard" mintákat, azokból kiválogattam az összes annotált sejtmagot. Minden sejtmag esetén a sugarat az alábbi lépésekkel határoztam meg:

1. A sejt tömegközéppontjának meghatározása:
   $Center_X = \sum P[i]_X / N$ (*8*)
   $Center_Y = \sum P[i]_Y / N$ (*9*)
2. Minden pixelre kiszámítjuk a sejt középpontjától való távolságot:
   $Distance_i = Distance(Center, P[i])$ (*10*)
4. Ezen távolságok közül a legnagyobb tulajdonképpen az objektum sugara:
   $R = MAX(Distance_i)$ (*11*)

Ahol N a sejtmag mérete (területe pixelben megadva), $P[i]_x$ és $P[i]_y$ pedig az i. pont x és y koordinátái. Természetesen vannak ettől eltérő (sőt, valószínűleg ennél precízebb) sugár számítási módszerek is, itt azonban lényeges szempont volt, hogy magát a számítást gyorsan el lehessen végezni.

Kiszámoltam ezeket az értékeket minden sejtmagra (6. táblázat, 6. diagram), amely alapján az alábbi határértékeket állapítottam meg:

- Legkisebb sugár: 2,69 pixel
- Legnagyobb sugár: 31,93 pixel




**6. diagram:** *Sejtmagok száma sugár alapján.*

| Lefedettség | Intervallum kezdete | Intervallum vége | Intervallum hossza | Int. hossz/teljes |
|---|---|---|---|---|
| **100,0%** | 2,69 | 31,93 | 29,24 | 100,00% |
| **99,5%** | 4,80 | 22,83 | 18,03 | 61,67% |
| **99,0%** | 5,33 | 21,57 | 16,24 | 55,55% |
| **95,0%** | 6,29 | 17,78 | 11,49 | 39,30% |
| **90,0%** | 6,80 | 16,06 | 9,26 | 31,66% |
| **80,0%** | 7,49 | 14,49 | 7,00 | 23,94% |

**7. táblázat:** *Sejtmagok száma sugár alapján.*

A sugár esetében a sejtmagok 99,5%-a 4,8 pixel és 22,83 pixel méretűek között található (7. táblázat). A későbbi keresést tehát ebben az intervallumban érdemes folytatni.

### *3.2.3. Sejtmagok körszerűségének vizsgálata*

Az egyes sejtmagok körszerűsége szintén lényeges információ a sejtmagkeresési algoritmus számára. Természetükből adódóan a sejtmagok általában körszerű objektumok, ami a gyakorlatban fontos információ, ugyanis a különféle méret és szín feltételek mellett lényeges, hogy a régiónövelés során minél inkább körszerűbb régiókat építsünk fel. Ezt az alábbiak biztosítják:

- A régiónövelés során az új pont felvételekor az ott használt jósági függvény a különféle színintenzitás információk mellett figyelembe veszi az újonnan felvehető pixel helyzetét is. Ez a pixel minél közelebb van a potenciális sejtmag aktuális középpontjához, annál nagyobb jósági értéket tulajdonítunk neki. Így már a régiónövelés során is az algoritmus, lehetőségeihez mérten, automatikusan körszerű objektumokat fog felépíteni.
- A régiónövelés megállítása után is lényeges információ ez a körszerűség, ugyanis az utólagos ellenőrzés során ezt is vizsgáljuk. Paraméterként megadható egy alsó korlát, az annál kevésbé körszerű objektumokat elvetjük, az annál nagyobb körszerűségi tulajdonsággal rendelkezőket pedig elfogadjuk sejtmagként.

A sejtmagok körszerűségének vizsgálatához meg kell vizsgálnunk a rendelkezésre álló „gold standard” mintákat, azokból ki kell válogatni az összes annotált sejtmagot. A sejtmagok körszerűségét az alábbi lépésekkel határoztam meg:

1. A sejt tömegközéppontjának meghatározása:
   $Center_X = \sum P[i]_X / N$ (*12*)

$Center_Y = \sum P[i]_Y / N$ (*13*)

2. Minden pixelre kiszámítjuk a sejt középpontjától való távolságot:

   $Distance_i = Distance(Center, P[i])$ (*14*)

3. Ezen távolságok közül a legnagyobb tulajdonképpen az objektum sugara:

   $R = MAX(Distance_i)$ (*15*)

4. Ezek alapján számítható a körszerűség:

   $Körszerűség = N / (R^2 * \Pi) \sim 32 * N / R^2$ (*16*)

Ahol N a sejtmag mérete (területe pixelben megadva), $P[i]_x$ és $P[i]_y$ pedig az i. pont x és y koordinátái. Amennyiben kiszámoljuk a fenti értéket minden sejtmagra (8. táblázat), az alábbi határértékeket állapíthatjuk meg:

- Legkisebb körszerűségi érték: 17,32
- Legnagyobb körszerűségi érték: 103,2

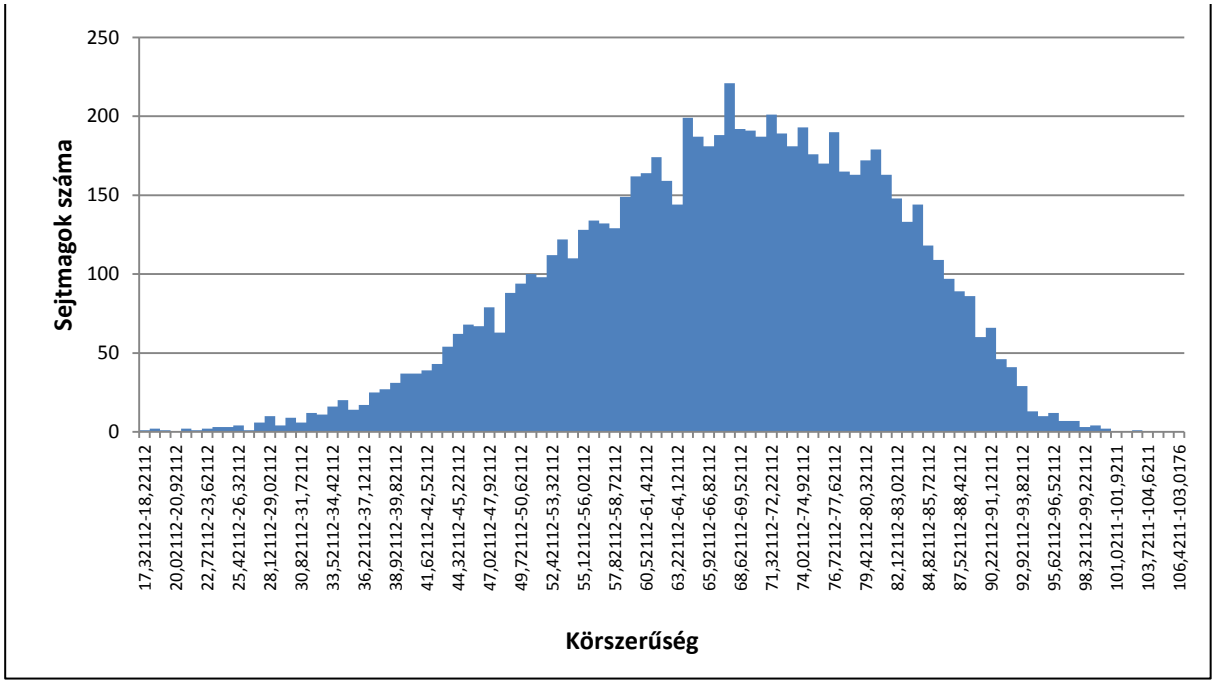
| # | Intervallum kezdete | Intervallum vége | Sejtmagok száma | Arány |
|---|---|---|---|---|
| **1** | 17,32 | 21,62 | 4 | 0,05% |
| **2** | 21,62 | 25,92 | 13 | 0,16% |
| **3** | 25,92 | 30,22 | 26 | 0,33% |
| **4** | 30,22 | 34,52 | 51 | 0,65% |
| **5** | 34,52 | 38,82 | 100 | 1,27% |
| **6** | 38,82 | 43,12 | 178 | 2,26% |
| **7** | 43,12 | 47,42 | 296 | 3,75% |
| **8** | 47,42 | 51,72 | 415 | 5,26% |
| **9** | 51,72 | 56,02 | 546 | 6,92% |
| **10** | 56,02 | 60,32 | 666 | 8,44% |
| **11** | 60,32 | 64,62 | 792 | 10,04% |
| **12** | 64,62 | 68,92 | 933 | 11,83% |
| **13** | 68,92 | 73,22 | 917 | 11,62% |
| **14** | 73,22 | 77,52 | 870 | 11,03% |
| **15** | 77,52 | 81,82 | 804 | 10,19% |
| **16** | 81,82 | 86,12 | 658 | 8,34% |
| **17** | 86,12 | 90,42 | 396 | 5,02% |
| **18** | 90,42 | 94,72 | 178 | 2,26% |
| **19** | 94,72 | 99,02 | 39 | 0,49% |
| **20** | 99,02 | 103,02 | 7 | 0,09% |

**8. táblázat:** *Sejtmagok száma körszerűség alapján.*

Az így megadott érték nem tekinthető tökéletesnek, de mivel a régiónövelés során minden egyes iterációban végre kell hajtani, így a paraméter kiszámítása során lényeges, hogy ezt minél gyorsabban el tudjuk végezni. A paraméter optimalizációnál a futásidő ugyan kevésbé kritikus szempont, de célszerű ugyanazt a metódust használni, mint amit a régiónövelési eljárás fog.

Az eloszlás természetesen itt sem egyenletes, a diagramon (7. diagram) is jól látható, hogy közelít a normál eloszláshoz. Mivel itt is van néhány, az átlagostól jelentősen eltérő sejtmag, így itt is célszerű az optimalizálás gyorsítása érdekében szűkíteni a vizsgált intervallumot.

A körkörösség esetében a sejtmagok 99,5%-a 27,66 és a 97,10 érték között található (9. táblázat). A későbbi keresést tehát ebben az intervallumban érdemes folytatni.




**7. diagram:** *Sejtmagok száma körszerűség alapján.*

| Lefedettség | Intervallum kezdete | Intervallum vége | Intervallum hossza | Int. hossz/teljes |
|---|---|---|---|---|
| **100,0%** | 17,32 | 103,02 | 85,70 | 100,00% |
| **99,5%** | 27,66 | 97,10 | 69,44 | 81,04% |
| **99,0%** | 31,72 | 96,26 | 64,54 | 75,31% |
| **95,0%** | 42,18 | 93,14 | 50,96 | 59,47% |
| **90,0%** | 46,74 | 90,99 | 44,25 | 51,64% |
| **80,0%** | 50,96 | 86,44 | 35,49 | 41,41% |

**9. táblázat:** *Sejtmagok száma körszerűség alapján.*

| # | Intervallum kezdete | Intervallum vége | Sejtmagok száma | Arány |
|---|---|---|---|---|
| **1** | 26,01 | 35,41 | 15 | 0,19% |
| **2** | 35,41 | 44,81 | 127 | 1,61% |
| **3** | 44,81 | 54,21 | 195 | 2,47% |
| **4** | 54,21 | 63,61 | 375 | 4,75% |
| **5** | 63,61 | 73,01 | 703 | 8,91% |
| **6** | 73,01 | 82,41 | 900 | 11,41% |
| **7** | 82,41 | 91,81 | 850 | 10,77% |
| **8** | 91,81 | 101,21 | 636 | 8,06% |
| **9** | 101,21 | 110,61 | 556 | 7,05% |
| **10** | 110,61 | 120,01 | 468 | 5,93% |
| **11** | 120,01 | 129,41 | 408 | 5,17% |
| **12** | 129,41 | 138,81 | 418 | 5,30% |
| **13** | 138,81 | 148,21 | 476 | 6,03% |
| **14** | 148,21 | 157,61 | 593 | 7,52% |
| **15** | 157,61 | 167,01 | 450 | 5,70% |
| **16** | 167,01 | 176,41 | 304 | 3,85% |
| **17** | 176,41 | 185,81 | 181 | 2,29% |
| **18** | 185,81 | 195,21 | 126 | 1,60% |
| **19** | 195,21 | 204,61 | 81 | 1,03% |
| **20** | 204,61 | 214,87 | 27 | 0,34% |

**10. táblázat:** *Sejtmagok száma átlagos intenzitás alapján.*

### *3.2.4. Sejtmagok átlagos intenzitásának a vizsgálata*

Általában feltételezzük, hogy a sejtmagok a képernyőn nagyobb intenzitású pixelek csoportjaként jelennek meg (a szövetmintákban ezek jobban festődnek meg, így sötétebb területek lesznek). Emiatt célszerű valamilyen korlátot bevezetni, amely segítségével el lehet dönteni, hogy egy pixelcsoport megfelel-e ennek a kritériumnak, vagy sem. A régiónövelést követő utófeldolgozás során van egy lépés, amelyik megvizsgálja a sejtmagjelölt pixeleinek átlagos intenzitását, majd ezt összehasonlítva egy minimum feltétellel, dönt, hogy a jelölt elfogadható-e (maximum feltétel megadására nincs szükség). Túl alacsony érték megnövelné a *hamis-pozitív* találatok arányát, túl magas érték esetében pedig elutasítanánk olyan jelölteket, amelyek valójában sejtmagok.

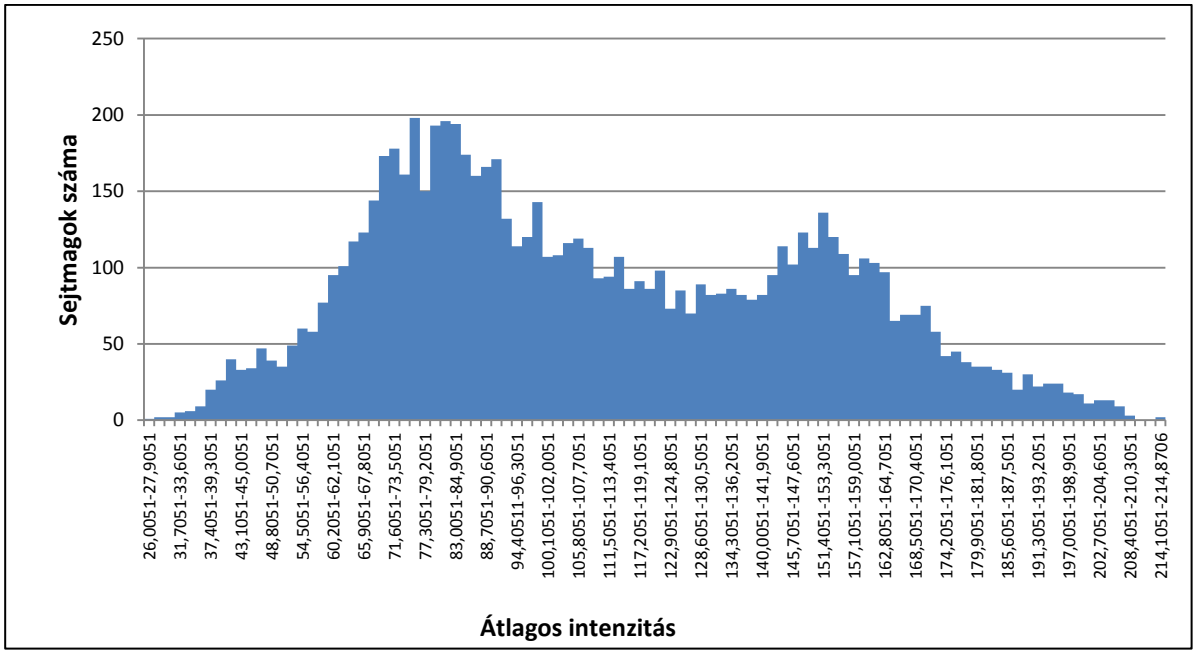
Ennek a paraméternek a meghatározásához is meg kell vizsgálni az összes, az orvosok által megjelölt sejtmagot (10. táblázat). A sejtmagok pixelenkénti vizsgálata során meg kell nézni,

hogy egy adott sejtmaghoz tartozó belső pont az eredeti képen milyen intenzitású pixelt jelöl, majd ezek átlaga adja meg a sejtmag átlagos intenzitását. Ez értelemszerűen függ magától a sejtmagtól, de jelentősen befolyásolja a minta megvilágítása, festettsége stb. Mivel azonban ezek ismeretlen értékek, így kénytelenek vagyunk egy általános mérőszámot keresni, amelyet minden minta esetében hatékonyan tudunk használni.

Amennyiben kiszámoljuk a fenti értéket minden sejtmagra (10. táblázat, 8. diagram), az alábbi határértékeket állapíthatjuk meg:

- Legkisebb átlagos intenzitás: 26,01
- Legnagyobb átlagos intenzitás: 214,87

Az átlagos intenzitás esetében a sejtmagok 99,5%-a 36,59 és a 205,01 érték között található (11. táblázat). A későbbi keresést tehát ebben az intervallumban érdemes folytatni.




**8. diagram:** *Sejtmagok száma átlagos intenzitás alapján.*

| Lefedettség | Intervallum kezdete | Intervallum vége | Intervallum hossza | Int. hossz/teljes |
|---|---|---|---|---|
| **100,0%** | 26,01 | 214,87 | 188,87 | 100,00% |
| **99,5%** | 36,59 | 205,01 | 168,42 | 89,17% |
| **99,0%** | 37,28 | 199,97 | 162,70 | 86,14% |
| **95,0%** | 41,71 | 180,78 | 139,08 | 73,64% |
| **90,0%** | 54,84 | 175,36 | 120,52 | 63,81% |
| **80,0%** | 61,44 | 161,11 | 99,67 | 52,77% |

**11. táblázat:** *Sejtmagok száma átlagos intenzitás alapján.*

| # | Intervallum kezdete | Intervallum vége | Sejtmagok száma | Arány |
|---|---|---|---|---|
| **1** | 0,00 | 12,80 | 1 | 0,01% |
| **2** | 12,80 | 25,60 | 0 | 0,00% |
| **3** | 25,60 | 38,40 | 0 | 0,00% |
| **4** | 38,40 | 51,20 | 0 | 0,00% |
| **5** | 51,20 | 64,00 | 1 | 0,01% |
| **6** | 64,00 | 76,80 | 0 | 0,00% |
| **7** | 76,80 | 89,60 | 7 | 0,09% |
| **8** | 89,60 | 102,40 | 11 | 0,15% |
| **9** | 102,40 | 115,20 | 31 | 0,41% |
| **10** | 115,20 | 128,00 | 59 | 0,78% |
| **11** | 128,00 | 140,80 | 165 | 2,18% |
| **12** | 140,80 | 153,60 | 485 | 6,42% |
| **13** | 153,60 | 166,40 | 1114 | 14,75% |
| **14** | 166,40 | 179,20 | 1356 | 17,95% |
| **15** | 179,20 | 192,00 | 1232 | 16,31% |
| **16** | 192,00 | 204,80 | 1171 | 15,50% |
| **17** | 204,80 | 217,60 | 1015 | 13,44% |
| **18** | 217,60 | 230,40 | 584 | 7,73% |
| **19** | 230,40 | 243,20 | 234 | 3,10% |
| **20** | 243,20 | 255,00 | 88 | 1,16% |

**12. táblázat:** *Sejtmagok száma kiindulópontok intenzitás alapján.*

### *3.2.5. Kiindulópontok intenzitásának vizsgálata*

A régiónövelés előtt meg kell találni azokat a pixeleket a feldolgozandó képen, amelyekből a következő növelési iteráció elindítható. Ehhez ki kell választani az aktuálisan legnagyobb intenzitású pixelt (mivel 8 bites képről beszélünk, ilyenből valószínűleg több is lesz), amely megfelel bizonyos feltételeknek (nem része egy már detektált sejtmagnak, lokális maximum). Mivel a képen található alacsonyabb intenzitású pixelekből indított növelések kis valószínűséggel eredményeznek valós sejtmagot, emiatt célszerű bevezetni egy minimum intenzitás feltételt, ezzel csökkenthető a *hamis-negatív* találatok aránya, illetve a futásidő is.

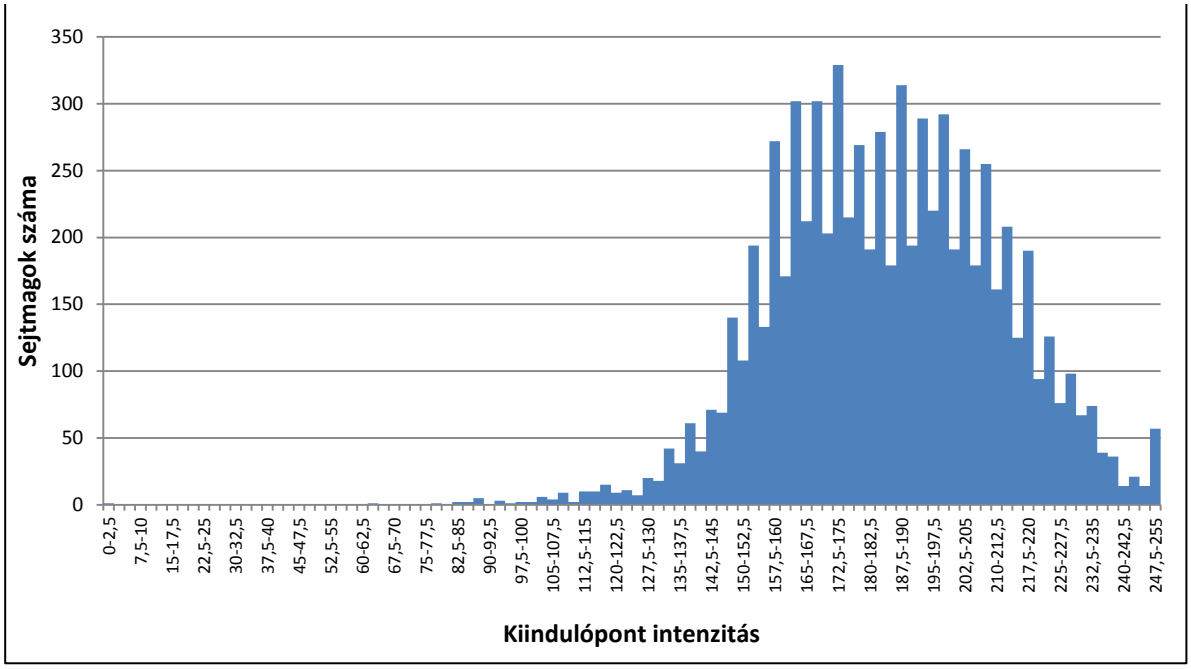
A megfelelő határértékek megállapítása érdekében megvizsgáltam (12. táblázat), hogy az orvosok által annotált sejtmagok detektálásához legalább milyen alacsony kiindulópont intenzitási korlátot kellett volna választani. Ehhez szükség van a sejtmag pontjaira, illetve magára a képre is, amelyiken a régiónövelő algoritmus magát a kiindulópont keresést

végrehajtja (ez az eredeti kép különféle szűrőkkel módosított változata [**9**]). Ezt követően a sejtmag minden egyes pixeléhez tartozó színt ellenőriztem az előzőleg említett képen, és ezek közül kiválasztottam a legnagyobb intenzitásút. Ez azt jelenti, hogy ha a kiindulási pont intenzitás minimumnak ezt a korlátot választjuk, akkor a régiónövelés a kiválasztott sejtmag területén legalább egy helyről biztosan el fog indulni.

Kiszámolva a fenti értéket minden sejtmagra (9. diagram), a határértékek:

- Legkisebb átlagos intenzitás: 26,01
- Legnagyobb átlagos intenzitás: 214,87

A kiindulópont intenzitás esetében a sejtmagok 99,5%-a 0 és 251 érték között található (13. táblázat). A későbbi keresést tehát ebben az intervallumban érdemes folytatni.




**9. diagram:** *Sejtmagok száma kiindulópontok intenzitása alapján.*

| Lefedettség | Intervallum kezdete | Intervallum vége | Intervallum hossza | Int. hossz/teljes |
|---|---|---|---|---|
| **100,0%** | 0,00 | 255,00 | 255,00 | 100,00% |
| **99,5%** | 0,00 | 251,00 | 251,00 | 98,43% |
| **99,0%** | 0,00 | 244,00 | 244,00 | 95,69% |
| **95,0%** | 117,00 | 255,00 | 138,00 | 54,12% |
| **90,0%** | 139,00 | 233,00 | 94,00 | 36,86% |
| **80,0%** | 148,00 | 219,00 | 71,00 | 27,84% |

**13. táblázat:** *Sejtmagok száma kiindulópontok intenzitás alapján.*

| # | Intervallum kezdete | Intervallum vége | Sejtmagok száma | Arány |
|---|---|---|---|---|
| **1** | 20,93 | 30,16 | 4 | 0,05% |
| **2** | 30,16 | 39,40 | 110 | 1,39% |
| **3** | 39,40 | 48,63 | 318 | 4,03% |
| **4** | 48,63 | 57,87 | 562 | 7,12% |
| **5** | 57,87 | 67,10 | 816 | 10,34% |
| **6** | 67,10 | 76,34 | 1018 | 12,90% |
| **7** | 76,34 | 85,57 | 876 | 11,10% |
| **8** | 85,57 | 94,81 | 704 | 8,92% |
| **9** | 94,81 | 104,04 | 609 | 7,72% |
| **10** | 104,04 | 113,28 | 614 | 7,78% |
| **11** | 113,28 | 122,51 | 557 | 7,06% |
| **12** | 122,51 | 131,75 | 503 | 6,38% |
| **13** | 131,75 | 140,98 | 377 | 4,78% |
| **14** | 140,98 | 150,22 | 266 | 3,37% |
| **15** | 150,22 | 159,45 | 186 | 2,36% |
| **16** | 159,45 | 168,69 | 142 | 1,80% |
| **17** | 168,69 | 177,92 | 111 | 1,41% |
| **18** | 177,92 | 187,16 | 64 | 0,81% |
| **19** | 187,16 | 196,39 | 40 | 0,51% |
| **20** | 196,39 | 205,63 | 12 | 0,15% |

**14. táblázat:** *Sejtmagok száma kontúr kontraszt alapján.*

A táblázatból látszik, hogy ha ragaszkodunk a 99,5%-os korláthoz, akkor nem tudjuk jelentősen szűkíteni a keresési intervallumot. Bár a pontok 95%-a 117-nél nagyobb intenzitású kiindulópontokból is megtalálható, de még így is számos értékes találat érhető el az alacsonyabb régiókban. A felső korlátra pedig jelen esetben nincs szükség, hiszen a vizsgálatot mindenképpen a lehető legnagyobb intenzitású (255) elemekkel kell kezdeni.

### *3.2.6. Sejtmagok kontúrjánál mérhető kontraszt vizsgálata*

Az egyes sejtmagokat nem csak az azokat alkotó pixelek intenzitása, illetve azok elhelyezkedése és mérete alapján értékeljük, hanem lényeges követelmény, hogy élesen elhatárolódjanak a környezetüktől. Ezt célszerűen egy kontraszt vizsgálattal végezzük el, amely során megvizsgáljuk, hogy a sejtmag belső kontúrja (amit még a sejtmag részének tekintünk) illetve külső kontúrja (amit már a környezet részének tekintünk) egymással milyen viszonyban áll. Erre az arányra szintén megadhatunk egy határértéket, és felállíthatunk egy

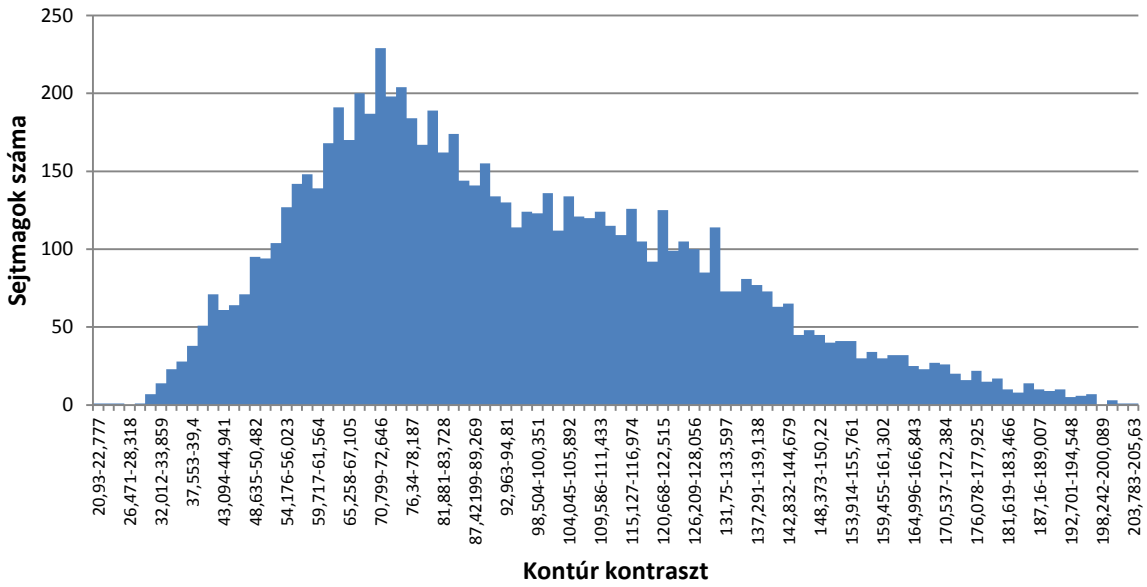
feltételt, amely szerint csak azokat a jelölteket fogadjuk el, amelyeknél ez a különbség elér egy bizonyos határértéket.

A külső és a belső kontúrt különféle konvolúciós műveletek sorozatával találhatjuk meg:

1. A sejtmagot alkotó pontokból egy maszkot készítünk. Ezt dilatáljuk, majd az így kapott képből kivonjuk az eredeti sejtmag pixeleit, így megkapjuk a *külső kontúrt*.
2. Ezt követően összegezzük az ezen a kontúron belül található pixelek intenzitás értékeit, majd elosztjuk a kontúrpontok számával. Ezzel meghatároztuk a *külső kontúr átlagos intenzitását*.
3. A *belső kontúr* hasonlóan előállítható egy erózió, dilatáció majd egy újbóli erózió után, ha az így kapott képből kivonjuk az eredeti sejtmag pixeleit.
4. Ezt követően összegezzük az ezen a kontúron belül található pixelek intenzitás értékeit, majd elosztjuk a kontúrpontok számával. Ezzel megkapjuk a *belső kontúr átlagos intenzitását*.
5. A két érték aránya megadja a nekünk szükséges *kontúr kontraszt* értéket.

Amennyiben kiszámoljuk a fenti értéket minden sejtmagra (14. táblázat, 10. diagram), az alábbi határértékeket állapíthatjuk meg:

- Legkisebb kontúr kontraszt: 20,93
- Legnagyobb kontúr kontraszt: 205,63




**10. diagram:** *Sejtmagok száma kontúr kontraszt alapján.*

| Lefedettség | Intervallum kezdete | Intervallum vége | Intervallum hossza | Int. hossz/teljes |
|---|---|---|---|---|
| **100,0%** | 20,93 | 20,563 | 184,70 | 100,00% |
| **99,5%** | 31,93 | 19,130 | 159,37 | 86,29% |
| **99,0%** | 31,93 | 18,456 | 152,63 | 82,64% |
| **95,0%** | 37,07 | 16,157 | 124,50 | 67,41% |
| **90,0%** | 41,33 | 14,730 | 105,97 | 57,37% |
| **80,0%** | 48,01 | 13,217 | 84,16 | 45,57% |

**15. táblázat:** *Sejtmagok száma kontúr kontraszt alapján.*

A kontúr kontraszt esetében a sejtmagok 99,5%-a 31,93 és 191,3 érték között található (15. táblázat). A későbbi keresést tehát ebben az intervallumban érdemes folytatni.

Természetesen itt sem beszélhetünk tökéletes pontosságról, de a mi esetünkben lényegesebb a minél nagyobb sebesség, akár kisebb pontatlanságok árán is, ez a funkció ugyanis szintén lefut minden régiónövelési iterációt követően,

### *3.2.7. Eredmények értékelése*

Az így nyert statisztikai adatok alapján tehát megkezdhető egy optimalizációs algoritmus fejlesztése, Amennyiben megelégszünk a teljes értékkészletek 99,5%-os lefedettségével, akkor a kezdő populációnak az alábbi intervallumokon belül célszerű kezdőadatokat átadni:

- Méret: 34 – 882 pixel
- Sugár hossza: 4,8 – 22,83 pixel
- Körszerűség értéke: 27,66 – 97,1
- Átlagos intenzitás értéke: 36,59 – 205,01
- Kiindulópont intenzitás értéke: 0 – 251
- Kontúr kontraszt értéke: 31,93 – 191,3

```
Kezdő populáció példányosítása
Kezdő populáció véletlenszerű inicializálása
Ciklus
     Előfeldolgozás
     Jósági függvény kiértékelése
     Genetikus operátorok alkalmazása
          Szaporítás
          Keresztezés
          Mutáció
Ciklus amíg ¬(leállási feltétel)
```

**3. algoritmus:** Genetikus algoritmusok általános felépítése.

## 3.3. Genetikus algoritmus kidolgozása

### *3.3.1. Genetikus algoritmusok általános felépítése*

Egy genetikus algoritmus általános felépítését mutatja a 3. algoritmus.

### *3.3.2. Kromoszóma reprezentáció*

Minden genetikus algoritmusnál döntést kell hozni arról, hogy milyen formában szeretnénk az egyes *egyedek* (ebben az esetben egy egyedet mindig egy *kromoszóma* határoz meg, emiatt a két fogalmat felváltva használom) adatait kódolni. Tehát a keresési teret le kell képeznünk valamilyen formában a kromoszómatérre. Erre számos lehetőség áll rendelkezésre, ebben az esetben az alábbi szempontokat kell figyelembe venni:

- Meglehetősen nagyszámú paramétert szeretnénk optimalizálni.
- Ezek a paraméterek egymástól függetlennek tekinthetők.
- A paraméterek mind számok, jelentős részük lebegőpontos szám.
- Az egyes paraméterek értékkészlete egymástól eltérő.
- A legtöbb paraméter egy alsó-felső határral körülírható, de több olyan paraméter is van, amelyeknek további szabályoknak is meg kell felelniük, (pl. csak páros szám lehet) és ezek a szabályok, nem biztos, hogy mind ismertek.

Meglehetősen gyakori módszer, hogy a paraméterek összességét egyszerűen egy bitvektorrá alakítjuk (azok kettes számrendszerbeli alakja alapján). Több kutató is azt javasolja azonban [**64**], hogy ne kódoljuk el a valós számokat bitvektorrá, vagyis a kromoszóma ténylegesen valós (lebegőpontos) számokat tartalmazzon. A módszer előnye, hogy könnyen alkalmazhatunk probléma specifikus keresztezést és mutációt.

Ebben az esetben többdimenziós keresési térről beszélhetünk, ennek megfelelően egy kromoszómán belül több gént (amelyek tulajdonképpen megfelelnek a régiónövelés paramétereinek) kell reprezentálnunk. Mivel az egyes gének közötti kapcsolatok nehezen írhatók le, így célszerűen olyan reprezentációt választottam, ahol minden egyes paramétert külön-külön kódolunk az értékkészletének megfelelően, illetve kimondhatjuk, hogy az egyes gének funkciója független azok lókuszától (kromoszómán belüli elhelyezkedésétől).

Természetesen előfordulhatnak életképtelen kromoszómák is (ha a keresztezés nem is, de a mutációk valószínűleg elő fognak ilyeneket idézni), ezeket azzal a módszerrel kezeltem, hogy a kiértékeléskor a jósági függvény visszatérési értéke 0 lesz. Így bár a kiértékelésük meg fog történni, de a következő generációkban már bizonyosan nem fognak részt venni.

### *3.3.3. Kezdő generáció felépítése*

A kezdő generáció felépítése általában véletlenszerűen generált egyedekkel történik meg. Amennyiben már előzőleg felderített paraméterek további finomhangolására van szükség, akkor természetesen van lehetőség azok elhelyezésére is a kezdő generációban, a mi esetünkben azonban egy teljesen új keresést szeretnénk indítani (habár már létezik a régiónöveléshez egy paraméterkészlet, amit a gyakorlatban is használnak, annak felvétele azonban nem járt volna sok előnnyel, hiszen a rendszer már néhány generációt követően is talált annál jobb megoldást).

A paraméterek egy részéről ismert azok legkisebb, illetve legnagyobb értéke [**65**]. Ezen paraméterek esetében az intervallumon belül egyenletes eloszlással lettek kiválasztva az egyes egyedekhez tartozó konkrét értékek.

Néhány technikai paraméternél ilyen előzetes vizsgálatra nem volt lehetőség, ezekben az esetekben a kezdő paraméterek értékei a jelenleg ismert legjobb paraméterkészlet értékei alapján lettek kiosztva. A intervallumok az említett érték ±10%-os környezetében, egyenletes eloszlás alapján lettek generálva. Ez természetesen nem garantálja azt, hogy az optimális eredmény is ebben az intervallumban lesz, emiatt mind az optimalizálás végeredményét, mind pedig a köztes generációk eredményeit célszerű megvizsgálni ebből a szempontból. Ha a jó eredményeket mutató egyedek valamelyik paramétere a fenti önkényesen választott határérték közelében mozog, akkor célszerű lehet ezt az intervallumot bővíteni. Bár épp a genetikus algoritmus működéséből adódóan ez nem feltétlenül szükséges, mivel a mutációk miatt a gének felvehetnek a fenti kezdeti intervallumokon kívüli értékeket is, tehát egy esetlegesen nem tökéletes kiinduló adatokkal létrehozott kezdő generáció is vezethet jó eredményhez.

Az első generáció minden egyede véletlen paraméterekkel lett létrehozva, így nagy valószínűséggel nagy számban jelennek majd meg köztük életképtelen egyedek is, ami a folytatás szempontjából nem szerencsés. Egyrészt szükséges, hogy minél nagyobb számú használható egyedünk legyen a következő generációkhoz, hogy minél több elem közül lehessen kiválasztani a legjobbakat, másrészt az első generáció fontossága egyébként is kitüntetett, hiszen szerencsés lenne, ha minden egyes paraméter minél több értékkel jusson tovább a következő generációkba, hiszen így nagyobb valószínűséggel marad köztük az adott paraméter optimális értéke is. Emiatt az első generáció jóval nagyobb elemszámmal indul, mint a következők, a mi esetünkben ez 3000 kromoszóma. A következő generációk már csak 300 egyedet fognak tartalmazni (mindkettő önkényesen, a kezdeti próbafuttatások tapasztalatai alapján kialakított érték).

### *3.3.4. Generációk kiértékelése*

Miután létrehoztuk a kezdő, illetve a későbbiekben az egyes újabb generációkat, ezeknek minden egyedét ki kell értékelni. Mivel itt a jóságot az határozza meg, hogy az egyes egyedek által képviselt paraméterkészletek segítségével mennyire jó eredménnyel fut le a régiónöveléses sejtmagkeresési algoritmus, így maga az egyedhez tartozó jósági tényező értékének meghatározása is két lépést igényel:

1. **Régiónövelés**: Először a megadott paraméterekkel le kell futtatni a régiónöveléses sejtmagkeresést [**9**][**40**]. Mivel a paraméterek másképp viselkedhetnek különféle típusú minták esetén (pl. kevés/sok sejtmag található a képen, kontrasztos/kevésbé elkülönülő sejtmagok), emiatt nem csak egy, hanem egymást követően több mintára is lefut a régiónövelés.
2. **Pontossági vizsgálat**: Ezt követi a régiónövelés eredményének kiértékelése. Ehhez össze kell hasonlítani egymással az algoritmus által adott eredményt és a referencia minták manuális annotációit [**47**]. Az így kapott eredmények átlagolásával kapjuk meg a paraméterkészlet jóságát. A genetikus algoritmus szempontjából ez nem más, mint a jósági tényező értéke.

### *3.3.5. Szülők és túlélők kiválasztása*

A szülőpárok kiválasztását tetszőleges módszerrel el lehet végezni, ami lényeges, hogy a nagyobb jóságértékkel rendelkezőket arányosan többször célszerű választani. Erre a legalapvetőbb megoldás a gyakran használt *rulettkerék módszer (roulette wheel)* [**66**]. Ebben az esetben egy képzeletbeli rulettkereket készítünk, amelyben minden egyedhez a jósági

tényező értékének megfelelő méretű rekesz tartozik. Az új szülők kiválasztása során pedig ezt a kereket „megpörgetve figyeljük meg, hogy a képzeletbeli golyó hová esik”.

Az aktuális generáció kiértékelése után már ismert az összes egyedhez tartozó jósági tényező. Ezek ismeretében a valószínűségeket az alábbi formában határozzuk meg:

$$P_i = (F_i - \mathrm{Min}(F)) / \sum(F_j - \mathrm{Min}(F)) \qquad (17)$$

Ahol

- $P_i$: Az i. egyed kiválasztásának valószínűsége.
- $F_i$: Az i. egyedhez tartozó jósági tényező.
- Min(F): Az aktuális generáció legkisebb jósági tényezője.

Jól látható, hogy ez a számítási mód azzal a mellékhatással is jár, hogy a legkisebb jósági tényezővel rendelkező egyedek teljesen kiszorulnak a következő generációból, a gyakorlatban azonban ez nem okoz problémát. Cserébe elkerülhetjük a rulettkerék módszer egyik nagy hátrányát, ami főleg akkor jelentkezik, ha az egyes kromoszómákhoz tartozó jósági tényezők egymáshoz túl közeliek.

Mivel elképzelhető, hogy életképtelen egyedek is bekerültek a generációba, ezért a kiértékelést követően 0 jósági értékű kromoszómákkal is találkozhatunk. Ezeket az elemeket még a fenti valószínűségek meghatározása előtt kizárjuk a vizsgálatból (a következő generációba egyébként se kerülhetnének át), így nem torzítják el a minimum értéket.

### *3.3.6. Elitizmus*

A paraméterek nagy száma miatt meglehetősen nagy a keresési tér, így az esetenként véletlenül előkerülő kiugróan jó eredményt adó egyedek könnyen el is tűnhetnek a következő generációkban a kötelező véletlen keresztezések miatt. Emiatt a klasszikus *elitizmushoz* (*elitism*) [**67**] hasonlóan az összes egyedből a legjobb jósági tényező értékkel rendelkezőket (nem csak a legjobbat, hanem a legjobb 30 darabot, tehát a teljes generáció 10%-át) keresztezés és mutáció nélkül továbbvisszük a következő generációba. Ez ugyan valamennyire csökkenti a generációnkénti próbálkozások számát, azonban garantálja, hogy a legjobb kromoszómák végig megmaradjanak, és génjeiket folyamatosan megtartsák. Ennek mellékhatása, hogy így generációnként a legjobb egyedhez tartozó jósági tényező értékek monoton növekvő sorozatát kapjuk.

Az elitizmus a feldolgozási teljesítményt nem befolyásolja, mivel az egymást követő generációkba is bekerülő azonos egyedekhez tartozó jósági tényezőt nem kell mindig újra kiszámolni, a keretrendszer ezeket az értékeket egy gyorsítótárban raktározza.

### *3.3.7. Keresztezés*

A genetikus algoritmusok legjellegzetesebb lépése a keresztezés. Ennek számos módszere ismert, kezdve a legegyszerűbb egypontos keresztezéstől, ahol egy véletlen helyen elvágjuk a kromoszómákat, és a keresztezési pont utáni kromoszómadarabokat kicseréljük [**59**]. Ez természetesen megoldható két- vagy többpontos formában is, a legáltalánosabb esetnek az *uniform keresztezést* [**68**] tekinthetjük, ahol egy véletlen keresztezési maszkot generálunk, ami gyakorlatilag bitenként eldönti, hogy a leszármazott melyik szülőtől kapja meg a gének egyes bitjeit.

Nehéz előre megállapítani, hogy melyik módszer a leghatékonyabb [**69**], az mindenesetre széles körben elfogadott, hogy a kétpontos keresztezés hatékonyabb, mint az egy pontos, a további kereszteződési pontok felvétele azonban nem jár feltétlenül előnnyel. Néhány további támpont:

- Viszonylag nagy populációnál kétpontos keresztezést érdemes használni, kisebbek esetén pedig uniform keresztezést.
- Rövid kromoszómáknál kevés, hosszabb kromoszómáknál pedig több keresztezési pontot érdemes választani.

Ebben az esetben a populáció mérete meglehetősen kicsinek tekinthető (mivel nagyon időigényes az egyes egyedek kiértékelése, így nem engedhetjük meg magunknak a nagy elemszámot), míg a kromoszómákat meglehetősen nagynak tekinthetjük (27 darab paraméter, összességében több száz bit), emiatt mindenképpen az uniform keresztezést célszerű választani.

Szigorúan véve az uniform keresztezést akár bitszinten is végre lehetne hajtani, ebben az esetben azonban ez nem lenne célszerű. A paraméterek között szerepelnek ugyanis olyanok, amelyeknek különféle szabályoknak kell megfelelniük (pl. oszthatóság), ezeket bitenként összekeverve könnyen juthatunk nem az értékkészlethez tartozó számokhoz.

Emiatt a keresztezés során csak teljes géneket keresztezünk. Egy új egyed minden egyes génje esetén egy véletlen szám alapján dől el, hogy azt melyik szülőjétől örökölje. A gyorsabb konvergencia érdekében a nagyobb jósági tényező értékkel rendelkező szülő génjeinek

nagyobb esélye van az öröklésnél. Minden egyes gén esetén az alábbi valószínűségek szerint dől el, hogy a leszármazott azt melyik szülőtől kapja:

$$P_a = (F_a - \text{Min}(F)) / (F_a + F_b - 2*\text{Min}(F)) \quad (18)$$

$$P_b = (F_b - \text{Min}(F)) / (F_a + F_b - 2*\text{Min}(F)) \quad (19)$$

Ahol

- $P_a$: Annak valószínűsége, hogy az A szülő génje öröklődjön.
- $P_b$: Annak valószínűsége, hogy a B szülő génje öröklődjön (nyilván $P_b = 1 - P_a$).
- $F_a$: Az A szülőhöz tartozó jósági tényező értéke.
- $F_b$: A B szülőhöz tartozó jósági tényező értéke.
- Min(F): Az aktuális generáció legkisebb jósági tényező értéke.

### *3.3.8. Mutációk*

A természetben is előforduló mutáció a genetikus algoritmusok működésében is fontos szerepet játszik. Ennek megléte főleg akkor lényeges, ha kellően sok generációt követően a változások már meglehetősen kicsik, és az egyes paraméterek már beálltak valamilyen értékre (ez nem jelenti azt, hogy egyáltalán nincs változás az egyes generációk között, azonban az előfordulhat, hogy a sok közül egy paraméter értéke stabilizálódik és így minden egymást követő generációban ugyanazt az értéket kapja minden kromoszóma esetén).

A mutációt önkényesen (a kezdeti próba futtatások alapján) az alábbi szabályok szerint választottam meg:

- A mutáció valószínűsége minden új generációban, minden paraméterre: 10%
- Mutáció mértéke: 60% eséllyel kicsi, 30% eséllyel közepes, 10% eséllyel nagy.

A mutáció mértéke nem határozható meg általánosan, minden paraméterre kiterjedően, hiszen a paraméterek értékei széles tartományokban mozognak. Bizonyos paramétereknél már számszerűen egészen kis változtatások is nagy hatásokkal bírnak, míg más paraméterek erre sokkal érzéketlenebbek. Emiatt a bitenként elvégzett mutációt elvetettem, hiszen ez bizonyos esetekben túl drasztikus változásokkal járna.

Sok paraméter meglehetősen kisméretű egész számként jelenik meg, ezeknél nem a százalékos eltérés használata se vezetne eredményre, hiszen ebben az esetben a kis és közepes méretű mutációk nem billentenék át az értéket a szomszédos egész számba. Emiatt ezek a $\pm\varepsilon$

*módszernek* [**70**] megfelelően, diszkrét értékekkel változnak (konkrétan kis változás 1, közepes változás 2, nagy változás értéke pedig 3; az előjel pedig az esetek felében pozitív illetve ugyanennyi esetben negatív).

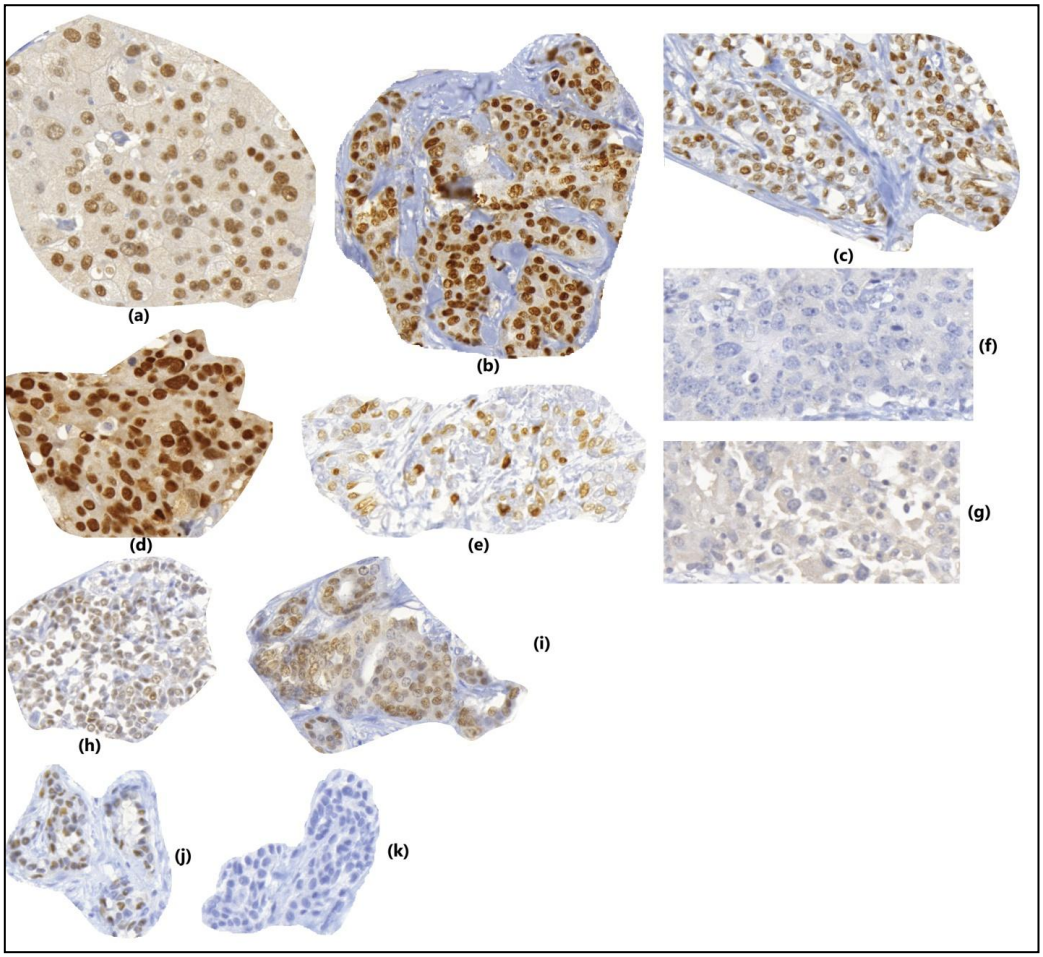
A mutációk esetében nincs beépítve külön ellenőrzés azügyben, hogy az így létrejött új érték megfelel-e az adott paraméter értékkészletének. Amennyiben egy mutáció miatt a paraméter ebből kicsúszna, akkor a kiértékelés során úgyis egy meglehetősen rossz (ha nem kiértékelhető, akkor 0) jósági értéket fog kapni, így a természethez hasonlóan ezek az életképtelen egyedek kiesnek a következő generációkból. Ezen egyedek száma pedig meglehetősen alacsony ahhoz, hogy jelentősen lerontaná a keresés menetét.

Egy kiegészítő ellenőrzéssel persze ez megelőzhető lenne, de egyrészt ennek a szabályrendszernek a kidolgozása nehézkes (mivel nem ismerjük pontosan a paraméterek egymásra való hatását), másrészt nem is lenne célszerű rögzíteni ezeket a szabályokat. A mutációknak ugyanis egy további lényeges szerepük is van a rendszerben: mivel a kezdő generációnál részben önkényesen megválasztott intervallumok között generáltunk paraméter értékeket, így elképzelhető, hogy egy tévedés okán az ideális gén érték (*allél*) be sem került a kezdő generációba. A mutáció ezekben az estekben biztosítani tudja, hogy egy esetlegesen szerencsétlenül választott intervallum (vagy kedvezőtlenül lefutott véletlenszám generálás) miatt egyébként feltáratlan területre is eljusson a keresés, ha az jó eredményekkel kecsegtet.

## 3.4.Elosztott genetikus algoritmusok áttekintése

Az előzőekben ismertetett genetikus algoritmus implementációját az alábbiak paraméterek megválasztásával végeztem:

- 27 darab paraméter értékét keressük.
- 3000 elemű kezdő generáció.
- 300 elemű minden ezt követő generáció.
- 11 reprezentatív szövetmintára (6. ábra) kell letesztelni minden egyes paraméterkészletet.




**6. ábra:** *A vizsgálat során feldolgozott szövetminták (a) B2007_02857_ES_01 b) B2007_02508_PR_01 c) 2007_02224_PR_01 d) B2007_02857_ES_02 e) B2007_03019_PR_01 f) B2007_03381_PR_01 g) B2007_03381_ES_01 h) B2007_02819_ES_01 i) B2007_02167_ES_01 j) B2007_00259_PR_02 k) B2007_00259_ES_02)*

A genetikus algoritmusok esetén gyakran a futásidő szab határt az optimális eredmények elérhetőségének. Ez ebben az esetben is fennáll, hiszen egyetlen egyed kiértékelése meglehetősen sok ideig tart: az első 1550318 kiértékelés alapján átlagosan 1498ms ideig tartott egy régiónöveléses feldolgozás, illetve további 8249ms ideig tartott a végeredmény kiértékelése egy-egy kép esetén (a régiónövelés eredménye és a referencia eredmény összehasonlítása). Mivel minden egyed esetében 11 szövetmintát vizsgálunk meg, így körülbelül 101 másodpercig tart egy kromoszóma vizsgálata, tehát egy-egy generáció feldolgozása több mint 8 órát igényel.

Mivel több száz generáció feldolgozásával kell számolnunk, ezért nyilvánvaló, hogy a feldolgozást valamilyen formában fel kell gyorsítani, ennek a legkézenfekvőbb módszere egy magasabb szintű párhuzamosítás.

### *3.4.1. Elosztott genetikus algoritmusok megvalósításának lehetőségei*

A szakirodalomban számos módszert találhatunk a genetikus algoritmusok párhuzamosítására. Ezek közül a legalapvetőbbek [**71**][**72**]:

A. **Egy populáción alapuló rendszerek:** Ezekben az esetekben maga a genetikus algoritmus tulajdonképpen nem különbözik a klasszikus szekvenciális megoldásoktól. Pusztán az implementációt próbáljuk meg úgy módosítani, hogy azonos eredmény mellett rövidebb futásidő alatt tudjuk elvégezni az egyes genetikus műveleteket.
   1. **Fordítón alapuló automatikus párhuzamosság**: Ebben az esetben a párhuzamosítást meglehetősen alacsony szinten valósítjuk meg, általában itt a hardver lehetőségeit próbáljuk meg kihasználni [**73**][**74**]. Mivel ez egy erősen technikai lehetőség, itt az esetek jelentős részében támaszkodhatunk a különféle fordítóprogramok segítségére, mivel ezek automatikusan végrehajtanak különféle optimalizálásokat a fordítás során [**75**].
   2. **Egy populáció – párhuzamos kiértékelés/keresztezés/mutáció**: Ebben az esetben a párhuzamosságot már egy magasabb szinten valósítjuk meg. Nem csak magukat az egyes elemi műveleteket próbáljuk meg párhuzamosítani, hanem nagyobb feladatokat tekintünk atominak, és ezeket oldjuk meg egymástól függetlenül. Ez a durvább szemcsézettség lehetőséget nyújt arra, hogy a párhuzamos feldolgozást kiterjesszük többprocesszoros vagy akár több munkaállomásból álló rendszerekre is [**76**].

B. **Több populáción alapuló rendszerek:** Ezekben az esetekben már nem feltétlenül egy globális populációval dolgozunk, hanem lehetőségünk van arra, hogy esetleg egyidőben több, egymástól független populáció fejlődését kezeljük. Természetesen kiépíthető ezek között is többféle kapcsolat, egymással kommunikálhatnak, majd ezek eredményeit összesíthetjük a feldolgozás végén. A két módszer nevében szereplő *szemcsézettség* tulajdonképpen nem más, mint a számítások és a kommunikáció aránya. Amennyiben ez az érték magas, akkor durva szemcsézettségű algoritmusról beszélhetünk, amennyiben pedig alacsony, akkor finom szemcsézettségű algoritmusról.
   1. **Durva szemcsézettségű megoldások:** A durva szemcsézettség (*coarse grain*) klasszikus esetének tekinthető, amikor egy elosztott rendszerben minden végrehajtó egység tulajdonképpen egy saját genetikus algoritmust futtat egy független populáción, annak minden műveletével (szelekció, keresztezés,

mutáció). Ez lehet egyrészt egymást *kölcsönösen kizáró* (*mutually exclusive)*, amennyiben ezek a független egységek egymástól teljesen elszigeteltek. Illetve lehet *nem kölcsönösen kizáró* (*non mutually exclusive)*, amely esetben valamilyen szintű kommunikáció történik az egyes populációk között (pl. megosztják egymással a legjobb eredményeket elért kromoszómákat). Számos egyéb megvalósítás is elképzelhető, a lényeg, hogy ezekben az esetekben valójában a genetikus algoritmus alapelve nem változik, ugyanúgy szekvenciális algoritmusok futnak, csak egyszerre több populációban [**77**].

2. **Finom szemcsézettségű megoldások**: Míg a durva szemcsézettség során gyakran egymástól teljesen elkülönült populációkat (szigeteket) képzelünk el, addig a finom szemcsézettségre (*fine grain*) tipikus példa lehet egy nagyméretű globális populáció, amelynek elemei egy rácsba vannak szervezve, és minden elem csak a szomszédjaival kommunikálhat (mindez természetesen párhuzamosítva az egyes egyedek szintjén). A végrehajtó egységek és a populáció méretének függvényében ez technikailag sokféleképpen valósítható meg, maga az elrendezés és a szomszédokkal való kommunikáció minden esetben egyedi tervezést igényel [**78**][**79**].

### *3.4.2. Egy globális populáción dolgozó módszerek értékelése*

Az A.1. és az A.2. módszer alapvetően ugyanazokon az alapelveken nyugszik, mint a hagyományos szekvenciális genetikus algoritmusok, azonban architektúrától függően annál gyorsabb végrehajtást ígérnek. Mivel az összes végrehajtó egység ugyanazon a populáción dolgozik, ez természetesen számos problémát felvet (zárolások, kommunikáció, egymás bevárásából adódó időveszteség), tehát mindenképpen figyelembe kell venni ezeket a veszteségeket a döntés meghozatala előtt. A szakirodalom ajánlásai alapján [**75**] egyértelmű, hogy ezeknek csak akkor van létjogosultságuk, ha az egyes egyedekkel kapcsolatos műveletek nagyon számítás- és időigényesek.

Ebben az esetben ez tipikusan igaz, hiszen minden egyes paraméterkészletre le kell futtatni egy teljes régiónövelést, majd pedig az így kapott eredmény pontossági értékelését (mivel itt meglehetősen precíz kiértékelésre van szükségünk, ez tovább tart, mint maga a régiónövelés), mindezt nem is csak egy, hanem egymás után több szövetmintára. Ezen műveletek költségéhez képest szinte elhanyagolható maguknak a különféle genetikus operátoroknak (szülőválasztás, keresztezés, mutáció) az erőforrásigénye, így ez alapján érdemes az A változatok közül választani.

Ezek közül az A.2. nyújt több lehetőséget, mivel a fordítóprogramok által nyújtott lokális optimalizálás meglehetősen korlátos lehetőségeket nyújt, illetve elsősorban nem is a genetikus operátorokat szeretnénk gyorsítani, hanem magát az egyes elemek jósági függvényének kiértékelését. Erre pedig a *master-slave* módszer nyújt egy nagyon jól implementálható megoldást, ami kellőképpen hatékonyan tudja kihasználni a rendelkezésre álló erőforrásainkat. A gyakorlatban is számos alkalmazást találhatunk, ahol már beváltotta a hozzá fűzött reményeket [**80**][**81**][**82**][**83**].

### *3.4.3. Több populáción dolgozó módszerek értékelése*

A B.1. módszer is meglehetősen nagy népszerűségnek örvend [**82**], azonban számos korláttal rendelkezik:

- Több figyelmet igényel a populációk méretezésénél, hogy a feldolgozóegységek mind teljesen kihasználtak legyenek.
- Bizonyos evolúciós lépések nehezen reprodukálhatóak a populációk feldolgozásának és a migrációk aszinkron volta miatt.
- Az egymástól független szigetek közötti kommunikáció jelentősen bonyolítja a modellt.

És bár az előnyei kétségtelenek bizonyos esetekben, jelen esetben ezek kevésbé dominálnak, ugyanis épp a fent említett nehézkes jósági függvény kiértékelés miatt nyilvánvaló, hogy az erőforrások szűkössége miatt nem tudunk kellőképpen nagy mennyiségű egyedet feldolgozni. És mivel a genetikus algoritmus számára így a populáció mérete lesz a szűk keresztmetszet, nem jelent előnyt, hogy mellette még további populációkat is fenntarthatunk.

Ugyanígy a B.2. módszer előnyeit sem lehet itt kihasználni. Ebben a feladatban ugyanis egyértelműen a jósági függvény kiértékelése igényli a legtöbb erőforrást, a többi művelet ehhez viszonyítva elhanyagolható. Ebben az esetben azonban a finom szemcsézettségű módszer különösebb előnyt nem jelent a master-slave megvalósításhoz képest, ellenben meglehetősen sok kötöttséggel jár. Ezek közül a legnagyobb problémát a szülőválasztás szabadságának elvesztése jelenti. Mivel a hagyományos szekvenciális módszerek használatával szemben, ahol a teljes populáció bármelyik két elemét választhatjuk szülő gyanánt, addig a durva szemcsézettségű módszer esetében már csak az adott alpopuláció egyedei közül válogathatunk, a finom szemcsézettségű esetben pedig még ennél is szigorúbb kötöttséghez kell alkalmazkodnunk, hiszen itt már csak a közvetlen szomszédok közül választhatunk [**84**].

A fentiek alapján tehát, habár a több populációval működő algoritmusok nagyobb népszerűségnek örvendenek, a mi esetünkben ezek használata nem tűnik célszerűnek. Egyrészt ezek egyik nagy előnye, a kommunikáció csökkentése nem jelent lényegi előrelépést, mivel ez a jósági függvény kiértékelés mellett elhanyagolható erőforrásigénnyel bír; másrészt épp ugyanezen nehézkes kiértékelés miatt a célunk egy minél gyorsabb eredmény megtalálása, ezt pedig várhatóan akkor érjük el, ha a teljes globális populáció minden tagjából szabadon válogathatunk a keresztezések során.

### *3.4.4. Hibrid megoldások*

Az első gyakorlati tapasztalatok alapján a későbbiekben célszerűnek tűnik egy hibrid rendszer [**85**] kialakítása. Az implementáció és az első futtatások során ugyanis kiderült, hogy az egyes egyedek feldolgozási ideje egymástól jelentősen különbözhet, a paraméterek bizonyos kombinációi során a régiónövelés futásideje akár az átlagos érték többszöröse is lehet. Ennek következményeképpen néha előáll egy olyan kedvezőtlen állapot, hogy bár egy kivételével a generáció minden egyede fel lett dolgozva, a párhuzamos kliensek várni kénytelenek az utolsó feldolgozás eredményére. Ezeket a holtidőket lehetne áthidalni azzal, hogy ilyenkor a várakozó klienseket egy másik populáció egyedeinek a feldolgozására ütemezzük.

### *3.4.5. Master-slave implementáció*

A master-slave számos előnye mellett (egyszerű implementáció; alapelve gyakorlatilag megegyezik a szekvenciális genetikus algoritmusokkal, így egyszerűen adaptálható; sok esetben nagyon jó teljesítményt nyújt) az egyik legnagyobb felmerülő probléma a meglehetősen nagy kommunikációs igény szokott lenni. Emiatt célszerű előzetes vizsgálatokat végezni azügyben, hogy még így is reális alternatíva-e.

Egy master-slave algoritmus végrehajtási ideje két fő komponensből áll:

- **A számításokra fordított idő:** Ez ebben az esetben főként a jósági függvény kiértékelésekből áll. A rendelkezésre álló 1550318 egyed feldolgozása alapján az alábbi átlagos értékeket mértük: 1498ms a régiónövelés lefutási ideje egy képen, 8249ms az ezt követő feldolgozás átlagos ideje. A különféle genetikus operátorok ideje gyakorlatilag elhanyagolható (0,16ms egy egyed esetén). Az általam választott populáció esetén ez összesen 50ms generációnként.
- **Kommunikációra fordított idő:** A master-slave rendszerben kommunikációról akkor beszélhetünk, amikor a master szétosztja a feladatokat a kliensek között, illetve amikor azok visszaküldik az általuk kiszámolt jósági tényező értékeket. Ez az érték egyrészt a

különféle felhasznált hardver (hálózat, csatlakozók) eszközöktől függ, másrészt a kommunikáció során felhasznált protokolltól is. Ebben az esetben ez utóbbival meglehetősen jól le tudjuk rövidíteni a kommunikáció idejét, hiszen a szerver és a kliens között csak minimális adatmennyiséget kell átvinni: a feladatok szétosztásakor a master átküldi a régiónövelés szükséges paramétereit, ez még a nem túl gazdaságos szöveges formában is csak átlagosan 70,67 byte. Az eredmények visszaküldése szintén csak néhány számadat küldését jelenti, 11 kép esetén ez átlagosan 539,32 byte.

A fentiekből jól látható, hogy a kommunikáció/számításidő aránya kimondottan kedvez ennek a megvalósításnak, még ha kisebb járulékos költségek ezen túlmenően is terhelik a kommunikációt (hálózat nem egyenletes terhelése, zárolások stb.), akkor sem fogja ez jelentősen visszafogni a feldolgozást.

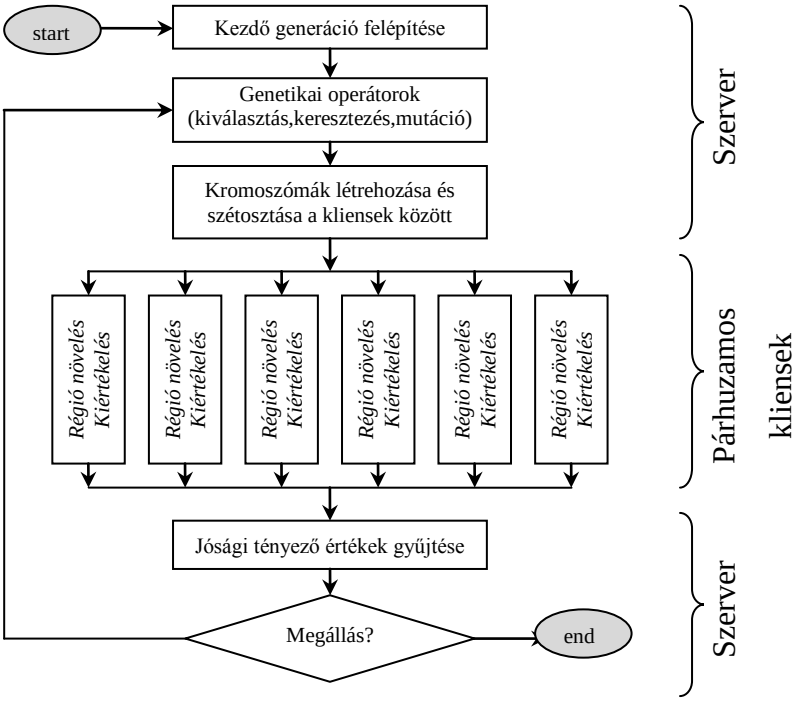
## 3.5.Elosztott rendszer kiépítése

### *3.5.1. Követelmények feltérképezése*

A gyakorlatban az elvi megvalósítás mellett számos implementációs lehetőséget is találhatunk a master-slave modell tekintetében, ezeket átvizsgálva alakítottam ki a saját modellemet [**86**].

Napjainkban már számos szabványosított technika adott az elosztott rendszerek kiépítésére, ezek közül vannak egészen komplex, ipari felhasználásra is alkalmas változatok [**87**][**88**][**89**] [**90**] (cloud, grid stb.) is. Bár tény, hogy ezek rendkívül divatos és elegáns megoldást nyújthattak volna a genetikus algoritmus futtatására, kezelésük azonban aránytalanul nagy erőforrástöbbletet jelentene, ami a jelenlegi kísérleti fázisban indokolatlan. Az újonnan készülő genetikus algoritmus megvalósítását persze még el lehetne végezni a fenti szempontok szerint, viszont alkalmazkodni kell a már meglévő külső modulokhoz is. Ezek pedig a fenti szabványos rendszerekkel való együttműködéshez jelentős módosításokat igényelnének (régiónövelési algoritmus, kiértékelő algoritmus), illetve bizonyos esetekben ez technikailag is megvalósíthatatlannak tűnik (GPGPU alapú régiónövelési algoritmus).

A felmerülő speciális problémák specializált megoldást igényelnek, emiatt érdemes ismét visszatérni az elosztott rendszerek régebben alkalmazott megoldásaihoz, amelyek ugyan valamivel több munkát igényelnek (nincs kész keretrendszer, hanem azt is létre kell hozni), viszont ennek köszönhetően a végeredmény minden tekintetben megfelel az igényeinknek.




**4. algoritmus:** Elosztott genetikus algoritmus felépítése.

A keresés során az Óbudai Egyetem infrastruktúráját, illetve esetleg néhány távoli gép erőforrásait tudtuk felhasználni, ezek viszont felvetnek néhány speciális igényt, amelyeket az újonnan kifejlesztett rendszernek (4. algoritmus) mind támogatnia kell:

- A legfontosabb szempont, hogy időben dinamikusan változó, hogy mikor hány darab klienst tudunk indítani (mindig csak az éppen szabad erőforrásokat tudjuk a keresésre használni, azokat viszont lehetőség szerint maradéktalanul). A rendszernek támogatnia kell tetszőleges időben új kliensek belépését, illetve meglévő kliensek kilépését [**91**].
- Lényeges, hogy a rendszer csak a lehető legegyszerűbb, szokványos kommunikációs módokat használja (protokollok, portok) hiszen elképzelhető, hogy a kliensek egy része tűzfal mögül próbál majd csatlakozni.
- Az egyes kliens munkaállomásokra lehetőség szerint minél egyszerűbben lehessen telepíteni a szükséges programokat, ideális esetben telepítésre ne is legyen szükség. Emellett legyen lehetőség automatikus frissítésre az egyes munkaállomások egyenkénti manuális elérése nélkül.

### *3.5.2. Genetikus bázis – kommunikációs réteg közötti kommunikáció*

A rendszer kialakításnál lényeges szempont volt, hogy rugalmas legyen, így a későbbiekben egyszerűen lehet majd fejleszteni hozzá további genetikus komponenseket, operátorokat (amelyek magát a genetikus algoritmust vezérlik), illetve protokollokat (amelyek a master-slave kommunikációt biztosítják).

A genetikus műveletek célszerűen egy komponensen belül helyezkedhetnek el tetszőleges megvalósításban, a rendszer pusztán azt követeli meg, hogy az valósítsa meg az alábbi interfészt (ahol a *Genom* típus egy kromoszóma adatait tartalmazza, alapvetően egy egész és lebegőpontos számokat tartalmazó vektor formájában):

- **Genom CreateRootItem()**: Létrehoz egy új kromoszómát tetszőleges paraméterekkel. A kiinduló, véletlenszerűen generált generáció egyedeinek létrehozásakor hívódik csak meg.
- **SelectParents(címszerint g1 : Genom, címszerint g2 : Genom):** Az eltárolt egyedek listájából tetszőleges módszerrel kiválaszt két szülőt (g1 és g2). Ennek a megvalósításával lehet a kiválasztás operátort megvalósítani annak megfelelően, hogy miként szeretnénk előnyben részesíteni a magasabb jósági tényező értékkel rendelkező egyedeket.
- **CreateChild(p1 : Genom p1, p2 : Genom) : Genom:** A paraméterként átadott két szülő alapján létrehoz egy harmadik kromoszómát, ami a függvény visszatérési értéke is egyben. Ez tulajdonképpen megfelel a keresztezés operátornak, itt lehet meghatározni, hogy a létrejövő új egyed pontosan milyen géneket kapjon szüleitől.
- **Mutation(címszerint g : Genom)**: A paraméterként átadott elem tulajdonságait tetszőleges módon megváltoztathatja. Megfelel a genetikus algoritmusokban szokásos mutáció operátornak, a megvalósítás teljesen tetszőleges.
- **PrepareParentSelector()**: Technikai segédmetódus, a fenti genetikus operátorokat képviselő metódusok előtt pontosan egyszer fut le. Itt célszerű a különféle inicializálásokat végrehajtani (például a rulettkerék módszer esetén a kiválasztás előtt létre kell hoznunk magát a kereket).
- **CONST_GENOM_ROOT_CNT:** A kezdő generáció (véletlenszerűen generált) mérete (kromoszómák száma).
- **CONST_GENOM_CNT:** A kezdő generált kivételével a többi generáció mérete (kromoszómák száma).

- **CONST_ELITISM_PRCNT:** Elitizmus százalékos értéke a teljes generáció arányában. Minden generációból a legjobb ennek megfelelő számú kromoszóma automatikusan átkerül a következő generációba.

A rendszer első változatában a fenti műveletek az alábbiak szerint implementáltam:

- **CreateRootItem:** Létrehoz egy új egyedet, inicializálja annak tulajdonágait, majd létrehoz számára egy új kromoszómát. Ehhez egy ciklus segítségével minden egyes génhez rendel egy véletlen számot a már említett intervallumok között. Mivel nem ismerjük az egyes génekhez tartozó értékek várható legjobb értékét, így az intervallumon belül minden érték ugyanolyan valószínűséggel kerül kiosztásra. Minden egyes új egyed létrehozása független az előzőektől, így elvileg előfordulhatnak akár teljesen ugyanolyan kromoszómák is (bár ennek esélye minimális).
- **PrepareParentSelector, SelectParents:** A már ismertetett rulettkerék módszert alkalmazza. A *PrepareParentSelector* metódus létrehozza a rulettkereket az előző (tehát az aktuálisan kiértékelt) generáció egyedeinek a jósági értékei alapján. A *SelectParents* pedig minden egyes meghívásakor visszaad két egyedet, amelyeket a rulettkerék módszer szerint választ, azok jósági értékei alapján.
- **CreateChild:** Létrehoz egy új egyedet. A keresztezés során csak teljes géneket keresztezünk, így az új egyed minden egyes génje esetén egy véletlenszám alapján dől el, hogy azt melyik szülőjétől örökölje. A gyorsabb konvergencia érdekében a magasabb jósági tényező értékkel rendelkező szülő génjeinek nagyobb esélye van az öröklésnél.
- **Mutation:** A nagy erőforrásigény miatt minél gyorsabban minél nagyobb területet szeretnénk lefedni a keresési térből, emiatt a megszokottnál valamivel nagyobb mutációs rátát választottam. Minden gén egymástól függetlenül 10%-os eséllyel mutálódik, azon belül 60% eséllyel a mutáció mértéke alacsony, 30% eséllyel közepes, 10%-os eséllyel nagy. Ezen túlmenően a mutáció 50%-50% eséllyel növeli, illetve csökkenti a paraméter értékét. Ezek pontos mértéke paraméterenként eltérő lehet.
- **CONST_GENOM_ROOT_CNT:** 3000 darab
- **CONST_GENOM_CNT:** 300 darab
- **CONST_ELITISM_PRCNT:** 10%

A fenti absztrakt metódusok megvalósítását követően a keretrendszer alkalmas a kezdő generáció létrehozására, illetve az azt követően szükséges genetikus operátorok alkalmazására. A fenti absztrakcióknak köszönhetően a technikai részletek jól elváltak a genetikus műveletektől, így lehetőség nyílik azok komplex kidolgozására, például a jósági függvény kiértékelésnél a már fent említett elosztott működés megvalósítására.

### *3.5.3. Master-Slave protokoll megvalósítása*

A futtató keretrendszer a fentiek szerint végrehajtja a szükséges genetikus operátorokat, illetve kezeli a jósági függvény kiértékeléseket is. Ez utóbbiakat elosztott módon, és lehetőséget ad arra, hogy ehhez meghatározhassuk a használandó protokollt. Ez amiatt lényeges, mivel így a működés könnyebben testreszabható attól függően, hogy a gyakorlatban hogyan lett megvalósítva a master-slave felállás: egy gépen belül *szálak* illetve *processzek* között [**92**] (itt a kommunikáció célszerűen nem igényli a hálózatot, célszerűbb az operációs rendszer által támogatott processz szintű kommunikációt használni), helyi hálózaton több gép között, vagy akár egymástól távol lévő gépek között (ami jelentős korlátokat jelent, ha ezek a gépek különféle tűzfalak mögött helyezkednek el).

A két oldal közötti kommunikációt egy köztes komponens végzi, amely (többek között) az alábbi interfész megvalósításával nyújt szolgáltatásokat mind a master, mind pedig a slave oldal irányába:

Master oldalról hívható metódusok:

- **ProcessNextGeneration(generationID : szám, genoms : Genom lista):** A master ezzel a hívással adja át a következő generáció azonosítóját, illetve az abba tartozó egyedek listáját. A kommunikációs komponens feladata, hogy a paraméterként kapott kromoszómák mindegyikére lefusson a jósági függvény kiértékelése.
- **ContinueGeneration(generationID : szám):** Technikai okokból szükségessé válhat egy már befejezett vagy éppen feldolgozás alatt álló generáció folytatása, ezt ezen a metóduson keresztül tudja kezdeményezni a master.

Slave oldalról hívható metódusok:

- **LoadWaitingPackets(generationID : szám):** A slave ezen a metóduson keresztül tudja jelezni, hogy be szeretne kapcsolódni a megadott generáció feldolgozásába.
- **LockNextProcessable() : Genom:** Lefoglalja és letölti a következő kiértékelendő egyed adatait. Visszatérési értéke egy *Genom* struktúra, ami tartalmazza az összes gént (jelen esetben a régiónövelés paramétereit).
- **FinishAndSaveScore(g : Genom, score : szám lista):** A jósági függvény kiértékelését követően a slave ezen a metóduson keresztül tudja visszatölteni az eredményeket. Mivel minden paraméterkészletet több szövetmintára is le kell futtatni, ennek megfelelően az eredmény is egy lebegőpontos számokból álló vektor.

Az objektumorientált szemléletnek köszönhetően a fenti interfészt tetszőleges formában megvalósíthatjuk, attól függően, hogy magát a tényleges kommunikációt hogyan szeretnénk megoldani [**72**]. Elsőként egy FTP-n [**93**] alapuló változatot implementáltunk. Ez valójában egy harmadik szintet is igénybe vesz a kommunikációhoz, ugyanis a master egy FTP szerverre tölti fel a feldolgozandó kromoszómák adatait, a kliensek innen foglalják le és töltik le maguknak a feldolgozandó elemeket, majd ugyanide töltik fel az eredményeket is.

Ennek előnye, hogy így nincs szükség közvetlen kapcsolatra a master és a slave gépek között, csupán annyi a lényeg, hogy egy olyan köztes FTP szervert kell használni, ami mindenhonnan elérhető. Maga az FTP pedig általában engedélyezett a tűzfalakon, így szükség esetén meglehetősen egyszerűen lehet egy-egy új klienst indítani. Bár maga a protokoll meglehetősen lassú kommunikációt jelent a többi szóbajöhető alternatívához képest, de összességében ez nem jelent jelentős hátrányt, mivel a feldolgozás során a jósági függvény kiértékelése jelenti a szűk keresztmetszetet. A rendelkezésre álló technikai környezet (gépek és hálózati beállítások) miatt pedig kénytelenek voltunk egy ilyen köztes réteget beiktatni.

### *3.5.4. Robusztusság biztosítása*

A rendszer működése során kiemelt fontosságú kritérium, hogy robusztus legyen. Elosztott rendszer lévén egyidőben több száz kliens is működhet, így a különféle esetlegesen felmerülő hibákat nem lehet futásidőben emberi személyzettel figyelni, illetve azokra reagálni. A javításokkal telt percek pedig a nagy számú kliens miatt meglehetősen nagy erőforrás kieséseket jelenthetnek. Emiatt lehetőség szerint a rendszernek minden bekövetkező problémát automatikusan tudnia kell kezelni [**94**]. Mindezt elsősorban úgy, hogy magát az alapműködést ez ne befolyásolja.

A legalapvetőbb problémák kiküszöbölése pusztán programozástechnikai feladat. Ilyenek a működés során szinte bármikor előfordulhatnak, például hálózattal kapcsolatos hibák (hálózat nem elérhető, szerver nem elérhető stb.), hardver eszközök ideiglenes vagy hosszabb távú kiesése (szerver leáll, egyes kliensek leállnak, áramszünet stb.). Ezeket főleg a jól felépített alkalmazás architektúrával lehet kivédeni, a rendszer fel van arra készülve, hogy minden külső hálózati (adatok küldése hálózatról, adatok fogadása hálózatról), vagy további processzek indítása (régiónövelő alkalmazás indítása) sikertelen lehet, így a hiba jellegétől függően a műveletet újra megkísérli, esetleg megpróbálja kijavítani.

Szintén gondot jelenthetnek a nem megfelelően megválasztott paraméterek is, amelyek vagy teljesen ellehetetlenítik, vagy pedig jelentősen megnehezítik (lelassítják) a régiónövelés, illetve a kiértékelő algoritmus működését. Mivel nem ismertek a paraméterek közötti összefüggések, ezek a problémás paraméterkészletek bármikor előkerülhetnek, általuk okozott hibákra emiatt fel kell készülni:

- Üres eredmények által okozott problémák.
- Hibás programfutás által okozott leállások.
- Megnövekedett futásidő által okozott problémák.

#### 3.5.4.1. Üres eredmények kezelése

Mivel meglehetősen nagyszámú, egymással laza kapcsolatban álló paramétert kell folyamatosan módosítani, így gyakran állnak elő olyan paraméterkészletek, amelyek a régiónövelő algoritmus számára használhatatlanok (pl. ha a paraméterként megadott minimális régióméret nagyobb, mint a paraméterként megadott maximális régióméret). A paraméterek hierarchiája meglehetősen összetett, és ismeretlen azok egymásra való hatása, így meglehetősen sok időt igényelne egy olyan összetett előzetes ellenőrzési rendszer beépítése, amely az összes, a gyakorlatban értelmetlen paramétert kiszűrne (már ha ez egyáltalán megvalósítható).

A rendszer emiatt azt a technikát alkalmazza, hogy minden generált paraméterkészletre megpróbálja lefuttatni a régiónövelést, amennyiben a paraméterek egymásnak ellentmondanak, akkor végeredményként valamilyen hibás választ várunk (tipikusan például egy olyan választ, hogy a program egyetlen sejtmagot sem talált). Bár ezek a kiértékelések is igényelnek erőforrásokat, az eredményhez vezető utat nem befolyásolják, hiszen a hibás paraméterekből adódó fals eredmények meglehetősen rossz pontszámokat kapnak a

kiértékeléskor, így ezek a kromoszómák tulajdonképpen maguktól kirostálódnak a következő generációkban.

#### 3.5.4.2. Hibás programfutás kezelése

Bizonyos paraméterek nem megfelelő megválasztása még a fentinél is kritikusabb eredményekkel jár, a program leállását, vagy akár a teljes lefagyását is okozhatják. Tipikusan ilyenek például a különféle szűrők paraméterei, amelyek egészen változatosak lehetnek, például az ablakméreteknél megszabhatnak valamilyen arányt, vagy az ablak szélessége csak páros szám lehet stb. Ezek ideális esetben még az algoritmus futtatása előtt kiderülnek az előzetes ellenőrzéskor, néha azonban csak a futás során, esetenként még a program lefagyását is okozva (mivel a régiónövelés használ külső komponenseket, például különféle szűrőket, amelyek hibás adatok esetén teljesen kiszámíthatatlanul viselkedhetnek, így még a lefagyást sem tudjuk minden esetben kiküszöbölni).

Emiatt egy külön ellenőrzést építettem be, hogy ha a program bármelyik minta kiértékelésekor egy hibakóddal áll le, vagy esetleg a régiónövelő alkalmazás lefagyna működése során, akkor a kliens ezt automatikusan 0%-os eredménynek tekinti, illetve be is fejezi a feldolgozást, nem lát neki az ehhez a paraméterkészlethez tartozó következő képnek, továbbá érvényteleníti az előző képeknél elért eredményeket is. Ez érthető, hiszen olyan paraméterhalmazzal, amivel nem tudunk teljes körűen minden képet feldolgozni, nem szeretnénk a későbbiekben foglalkozni, függetlenül attól, hogy milyen eredményt ért el azokon a képeken, amiknél véletlenül nem jelentkeztek hibák.

Bár itt is lehetne készíteni egy komplex előszűrő modult, ami megvizsgálja a bemenő paraméterek helyességét, de itt is azt a módszert követtük, hogy engedtük, hogy a genetikus algoritmus önmaga kiszűrje a hibás paramétereket. A módszer bevált, és ennek eredményei jól láthatóak az utólagos statisztikákban is, ugyanis amíg az első néhány generációban az elemek meglehetősen nagy része használhatatlan paraméterkészletet tárolt (ne felejtsük el, hogy az induló generációt megadott intervallumok közötti véletlen számokkal állítottuk elő, így nyilván sok, egymásnak ellentmondó paraméter jött létre), azonban ezek a későbbi generációkban meglehetősen gyorsan kitisztultak, az 5.generáció után már csak elvétve jöttek elő ilyen paraméterek (a mutációk és az ellenőrzés nélküli kereszteződések miatt persze ezek soha nem fognak eltűnni, de a fenti kezelési módnak köszönhetően soha nem jutnak tovább a következő generációba).

#### 3.5.4.3. Megnövekedett futásidő kezelése

A fentieknél jóval nagyobb elvi problémát okoznak azok a kromoszómák, amelyek ugyan nem okoznak semmilyen hibát, azonban jelentősen megnövelik a feldolgozás erőforrásigényét. A gyakorlatban egy kép feldolgozása kb. 10 másodpercet vett igénybe, azonban bizonyos paraméterek mellett ez az érték ennek a sokszorosára is növekedhet. Önmagában ez nem jelentene problémát, hiszen a kliensek egymástól függetlenül dolgoznak, a gond ott jelentkezik, hogy egy új generáció indításához összegezni kell az előző generáció minden kromoszómájának eredményét. Tehát ha egy kliens kap egy meglehetősen hosszú ideig tartó feldolgozást, akkor a többi kliens, miután végeztek minden, az adott generációhoz tartozó elem feldolgozásával, várakozni kényszerül. Ezalatt csak a töredékét használjuk fel a rendelkezésünkre álló erőforrásoknak, így ezeket a várakozásokat mindenképpen minimalizálni kell, akár még némi pontatlanság árán is: érdemesebb leállítani az ilyen túlzottan elnyúlt feldolgozásokat. Az átlagos idő kiszámítása mellett (10 másodperc) részletesebb statisztikai adatokat gyűjtöttük az egyes régiónövelések és kiértékelések futási idejéről, ez alapján 1 percnél húztunk meg ezt a határt.

Minden egyes időkorlát miatti leállítás természetesen a genetikus algoritmusok szabályainak figyelmen kívül hagyását jelenti. Érdemes azonban figyelembe venni, hogy a különféle evolúciós megvalósítások erőssége egyébként sem a pontosság, és a minél részletesebb elemzés, ennél lényegesebb, hogy minél több generációt tudjunk létrehozni, minél többféleképpen tudjuk kombinálni az életképesnek tűnő paramétereket. Mindez meglehetősen erőforrás-igényes, ezért célszerűnek tűnik az összes erőforrásunk felhasználásával több száz új kombinációt végigpróbálni, ahelyett, hogy egyetlen kromoszóma precíz kiértékelése miatt perceket álljon a teljes rendszer.

Bár jelen esetben a keresendő ideális paraméterkészlet alatt általában a lehető legpontosabb eredményt nyújtó paraméterhalmazt értjük, érdemes figyelembe venni azt a tényt is, hogy a régiónövelés egy meglehetősen számításigényes algoritmus, így a paraméterek helytelen megválasztása akár a gyakorlatban használhatatlan algoritmust is eredményezhet. Ezt a szempontot figyelembe véve még inkább indokolt a többszörös feldolgozási időt okozó kiértékelések leállítása, hiszen ezek azt ezt okozó kromoszómák a pontosságtól függetlenül sem lennének használhatóak a gyakorlatban. És bár nem ez a genetikus algoritmus elsődleges célja, de mintegy kedvező mellékhatásként ezzel a technikával a rossz pontosságot nyújtó elemekhez hasonlóan a rossz futásidőt eredményező paramétereket is ki tudjuk rostálni.

### *3.5.5. Vágás a kiértékelési algoritmusban*

A régiónövelés mellett maga az eredményeket kiértékelő algoritmus is meglehetősen nagy futásidővel rendelkezik bizonyos esetekben. Amennyiben nagyon sok elemet kell feldolgozni (tehát mind az eredeti „gold standard” minta, mind pedig a régiónövelés által talált sejtmagok száma nagy) és ezek egymással minél kevésbé vannak egy-az-egyhez átfedésben, annál több próbálkozást igényel a kiértékelés.

Az első fejezetben bemutatott visszalépéses keresésen alapuló algoritmust kiegészítettem egy plusz vágási művelettel: mielőtt a keresés elkezdi megkeresni az ideális párosítást, kiszámolja, hogy összesen hány lehetséges kombináció létezik (bár azt előre persze nem tudja, hogy a visszalépéses keresés ebből ténylegesen hányat fog majd megvizsgálni) és ha ez nagyobb mint egy előre megadott határérték, akkor megkeresi azokat a pontokat, amelyeknél tudja egyszerűsíteni a kiértékelést anélkül, hogy ezzel jelentősen befolyásolná a végeredményt.

Mindez az alábbi lépéseket jelenti:

1. Először egyesével végignézi az egyes részfeladatokat, és megkeresi, hogy az egyes részfeladatoknál melyik részmegoldás választása járna a legnagyobb jósági tényező értékkel.
2. Ezen lokális maximumok alapján megkeresi a globális maximumot, tehát azt a referencia sejtmag – teszt sejtmag párosítást, ami a többi párosítási lehetőségtől függetlenül a legnagyobb átfedést mutatja.
3. Miután ezt megtalálta, ezt a párosítást rögzíti (éppúgy, ahogy azt maga a visszalépéses keresés is tette volna a keresés végeztével).
4. Mivel a visszalépéses keresés algoritmusnak ebben a részfeladatban már nincs tennivalója, így törli a teljes részfeladatot. Amennyiben a részfeladat K darab párosítási lehetőség megvizsgálását igényelte volna, akkor a teljes visszalépéses keresés által lefedni kívánt problémaтér mérete K-adrészére csökken.
5. Amennyiben a lehetséges kombinációk száma még mindig túl magas, újrakezdi ezt a műveletet az 1-es ponttól kezdve.

Ez a módszer jelentősen lecsökkenti a futásidőt (ugyanis a lehetséges párosítások számával exponenciálisan növekszik a keresési idő, így néhány szerencsétlenül megválasztott párosítás drasztikus várakozási időt jelent a teljes rendszer számára). Hátránya természetesen az, hogy romlik a kiértékelés pontossága, mivel a fenti módszerrel már nem garantálható, hogy minden egyes sejtmag párosításnál a globálisan legjobb lehetőséget választjuk majd. Kisebb előnye a

fenti módszernek, hogy az egyszerűsítésben nincs véletlenszerű elem, így legalább a kiértékelő művelet bármikor megismételhető, mindig ugyanazt az eredményt fogja adni.

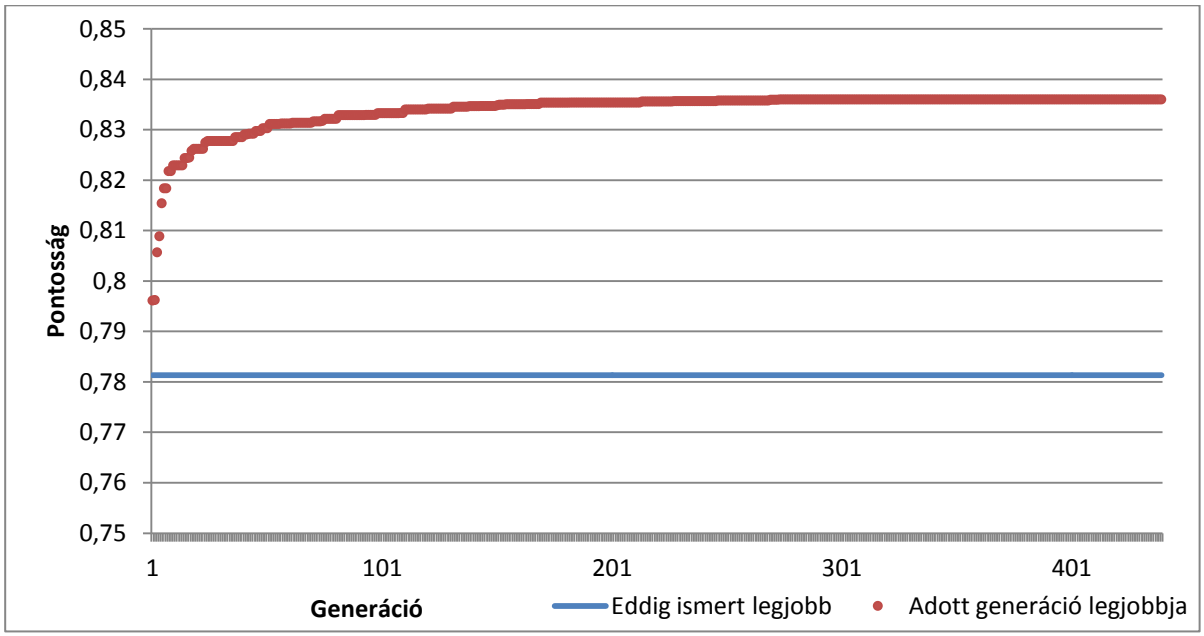
## 3.6. Genetikus algoritmus kimenetének analízise

Az előbbiekben vázolt rendszert implementáltam C# programozási nyelven Windows környezetben. A paraméterek finomhangolása ügyében több részleges futtatás történt, majd ezt követően volt lehetőségem egy hosszabb távú vizsgálatra. A következőkben ismertetem az eredményeket 440 generáció feldolgozása után (ez kb. 3948 munkaórát igényelt, a párhuzamos futtatásnak köszönhetően mindez a gyakorlatban egy hosszú hétvégét jelentett).

### *3.6.1. Generációnként a legjobb eredmény vizsgálata*

Kitűzött végcélról jelen esetében nem beszélhetünk, mivel nincs egy konkrét cél, amit el szeretnénk érni. Leszámítva persze a 100%-os pontosságot, de annak elérése reménytelennek tűnik, figyelembe kell ugyanis vennünk azt is, hogy maga a régiónövelés eleve nem garantálja azt, hogy hibátlan eredményt fog visszaadni, még ha meg is találjuk az optimális paraméterkészletet. Célunk tehát csak a lehetőségekhez mérten legjobb eredmény elérése.

Összehasonlítási alapként jól használható a Pannon Egyetem által kifejlesztett régiónöveléshez eredetileg kidolgozott paraméterkészlet [**9**]. Ezzel lefuttatva a 11 szövetminta vizsgálatát, majd a szokásos ellenőrzést, átlagosan 78,1%-os pontossághoz jutunk, jelenleg ezt tekinthetjük a legjobb ismert paraméterkészletnek. Ennek az értéknek bármilyen mértékű meghaladása mindenképpen hasznos eredménynek tekinthető.

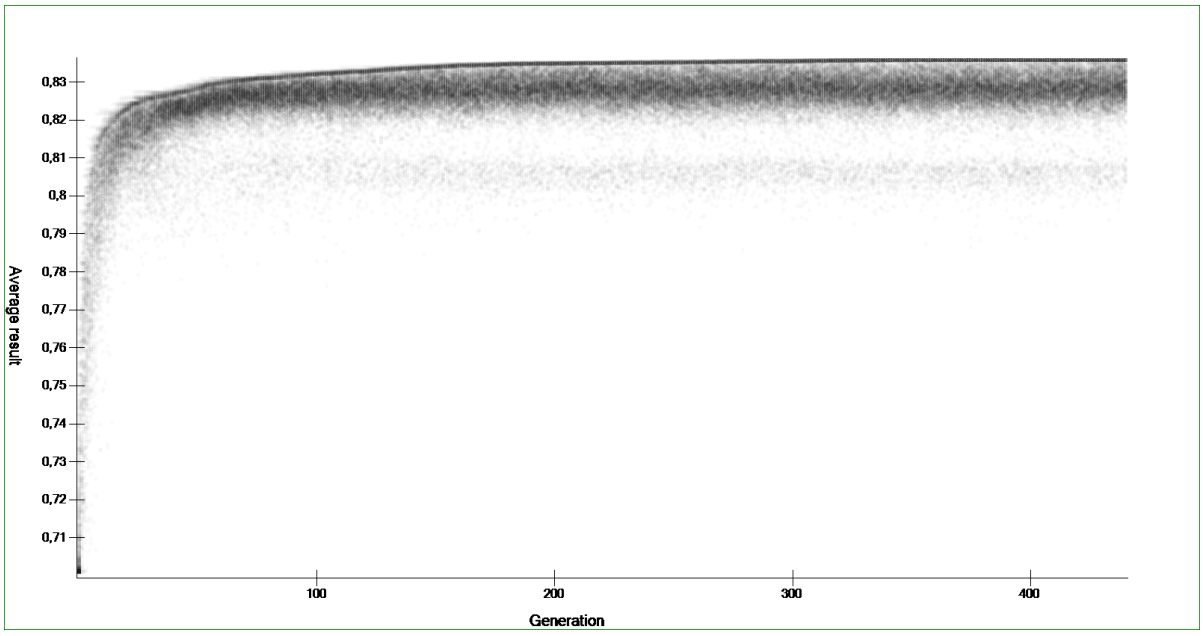



**11. diagram:** *Legjobb pontossági értékek generációnként.*

Az elitizmusnak köszönhetően minden generációból automatikusan továbbjut a legjobb eredményt elérő egyed a következő generációba is. Ezért generációnként a legjobb eredmények monoton javuló sorozatot adnak. A sorozat vizsgálatával arra lehet következtetni, hogy milyen ütemben halad az optimumkeresés (11. diagram).

Az ábrán látható az egyes generációk során elért legjobb eredmény, ahol a vízszintes vonal a 78,1%-os szintet mutatja. Jól látható, hogy némi szerencsének köszönhetően már az első generációban is volt olyan elem, amelyik elérte ezt a szintet, a későbbiekben pedig tovább sikerült javítani a találati pontosságon. A 273. generációra a pontosság elérte a 83,6%-ot, ez 5,5%-os növekedést jelent az eddig ismert legjobb eredményhez képest. Az algoritmus ezt követően még futott tovább, de leállításáig (440. generációig) már nem sikerült ennél jobb eredményt elérni.

Bizonyos esetekben célszerű lenne az egyes generációk minden egyedét ábrázolni egy közös diagramon. Mivel az elemek darabszáma meglehetősen nagy (440 generáció esetén már jóval 100000 feletti szám), így a hagyományos diagram típusokon ez nehezen lenne áttekinthető. Emiatt készítettem egy segédprogramot, amely ezekből az adatokból tud jól áttekinthető ábrákat készíteni, a későbbiekben a diagramok egy része innen származik. Az egyes egyedeket mindig szürke, áttetsző pontokkal jelöltem. Így ha egy helyen egymáshoz közel több egyed is ábrázolásra került, akkor ezt egyre sötétebb pontfelhők jelölik.




**12. diagram:** *Generációnként az összes egyed pontosságának értéke.*

A következő ábrán (12. diagram) látható az egyes generációkon belül az egyedek eloszlása is (az életképtelen egyedek kivételével). Itt fel lett rajzolva minden generáció esetén (vízszintes tengely) az összes életképes egyedhez tartozó pontosság (függőleges tengely) szürke pontokkal, értelemszerűen a sötét területek sok, hasonló eredményt elérő kromoszómát jeleznek. Látható, hogy az első néhány generációkban ezek többnyire még elmaradtak a fent említett 78,1%-os értéktől, azonban néhány generációval később már a többség ennél jobb eredményeket nyújtó kromoszómákat tartalmazott.

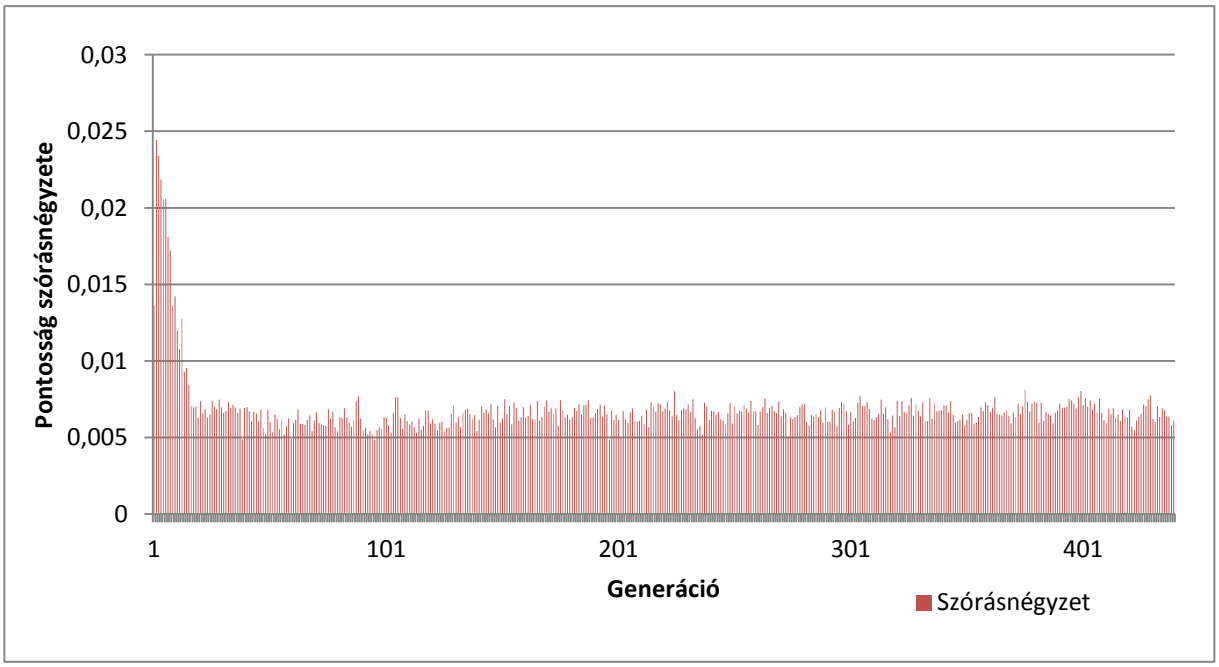
Az algoritmus lefutását részletesen vizsgálva az alábbi szakaszokat különböztethetjük meg:

1. Az első néhány populáció (kb. 4-5. generációig) még többnyire gyenge eredménnyel bíró egyedeket tartalmazott, ezek általában nem érték el az eddig ismert legjobb pontosságot sem, bár mindig volt közöttük néhány, amelyik már azt meghaladta. Ez a szám az egymást követő generációkban rohamosan növekedett. Látható egy sötét „folt" az első generációknál 70% környékén, ez annak köszönhető, hogy az üres eredményt eredményező (tehát technikailag életképes, csak a régiónövelés eredményeként egy sejtmagot se találó) kromoszómák esetében az átlagosan pontosság 70% lett[3], és az első generációk sok ilyen kromoszómát tartalmaztak.
2. Ezt követte egy dinamikus felfutási szakasz (kb. a 100. generációig), ahol a jó eredményeket elért egyedek génjei elterjedtek, így az eredmények is jelentősen javultak. Jól látható, hogy meglehetősen gyorsan szinte már minden egyed túllépte a 78,1%-os pontosságot. Ez a rész a genetikus algoritmusoktól elvárható folyamatot mutat, meglehetősen tág határok között induló populáció elemei egymáshoz egyre inkább közelednek, és mivel főleg a jobb eredménnyel bíró kromoszómák felé haladnak, így az átlagos eredmények is egyre magasabbak.
3. Ezt pedig követte az utolsó szakasz, ahol ez a dinamikus növekedés jelentősen lelassult. Egy ideig még talált a program jobb eredményeket, részben még a keresztezésekből adódóan, a későbbiekben pedig valószínűleg már egyre inkább csak a mutációknak köszönhetően. A 273. generációt követően még csaknem 200 generációt vártam, de már nem sikerült jobb eredményt elérni, bár érdemes megjegyezni, hogy még itt is sokféle kromoszóma van jelen és nagy területen fedik le a keresési teret, de ez már inkább csak egy véletlen keresésnek tekinthető.

[3] A referencia képek területének jelentős részén nem találhatók sejtmagok, emiatt egy teljesen üres kép is egy elsőre zavarbaejtően jónak tűnő eredményt ér el. Gondolnunk kell azonban azokra a megoldásokra is, amelyek sok *hamis-pozitív* találatot eredményeznek, emiatt az üres eredményt nem tekinthetjük 0%-os pontosságúnak.

Az kromoszómák eloszlásában itt egy érdekes mintázat alakult ki, ahol az alábbiakat láthatjuk:

- Az algoritmus használja az elitizmus módszerét, tehát minden generáció első 30 legjobb kromoszómája automatikusan átjut a következő generációba. Mint látható, a 200. generáció környékén ezek már olyan közel kerültek egymáshoz, hogy a diagramon egy sötét (tehát egy helyen sok azonos elemet tartalmazó) vízszintes sávot alkotnak. Ez nem jelenti azt, hogy az első 30 egyed mindenhol megegyezik, kisebb különbségek megjelennek az egyes génekben, de a kiértékelés mindegyikre ugyanazt az eredményt hozza.
- Ez alatt látható egy szélesebb sáv, ami az újonnan létrehozott (keresztezés és mutáció), változó eredményeket elérő egyedeket tartalmazza. Ez a sáv jellegét tekintve szinte változatlan az egymást követő generációk között, ez annak köszönhető, hogy bár a keresztezésektől azt várjuk, hogy hosszabb távon valamilyen optimum felé konvergáljanak, a mutációk azonban teljesen véletlenszerűek. És ezen a területen már inkább a mutációk dominálnak.
- Pusztán érdekességként érdemes megemlíteni egy ez alatt látható halványabb sávot, ami pusztán annak köszönhető, hogy az egyik paraméter egy bizonyos értéket felvéve fixen kb. 2%-al rontotta az adott kromoszóma pontosságát, ebbe a sávba pedig ezek az elemek kerültek (ez jól látható akkor, ha nem ábrázoljuk ezeket az egyedeket, ugyanis akkor ez a sáv eltűnik).




**13. diagram:** *Pontossági értékek szórásnégyzete generációnként.*

### *3.6.2. Leállási feltétel vizsgálata*

#### 3.6.2.1. Pontosság szórásának vizsgálata

Az eredményeket célszerű megvizsgálni olyan szempontból, hogy meddig érdemes futtatni az algoritmust. Azt már tisztáztuk, hogy nincs kitűzött végcél, aminek elérésekor befejeződik a feldolgozás, de előfordulhatnak állapotok, ahol már érdemes lehet leállítani a keresést. Ilyen lehet az, amikor az egymást követő generációkban lévő egyedek egymáshoz olyan közeli géneket tartalmaznak, hogy a kereszteződések már nem adnak lényegi változásokat.

A jobb áttekinthetőség érdekében célszerű megvizsgálni generációnként az egyedek pontosságának szórásnégyzetét (13. diagram). Amint az várható, a véletlenszerűen generált elemek és az ezekből létrehozott első néhány generációban a szórás még meglehetősen magas. A későbbiekben azonban az életképesebb egyedek előretörésének köszönhetően ez gyorsan lecsökken[4].

Ez a csökkenés azonban egy idő után megáll, és innentől a szórás stabilnak tekinthető. Bár a genetikus algoritmusoktól elsőre azt várnánk, hogy ez továbbra is csökkenjen, miként az elemek egyre közelebb kerülnek egy optimális eredményhez és így egymáshoz is, ez itt nem következett be. Ennek az az oka, hogy a mutációk mértékét meglehetősen nagyra választottam. Mivel egy nagy és nehezen leírható keresési térben kell dolgoznunk, ezért a mutációk meglehetősen nagy arányával próbáltam gyorsítani a tér minél szélesebb körű feldolgozását. A sok mutáció azonban nem engedi, hogy az egyes gének értékei beálljanak egy ideális értékre, így a pontossági értékek szórása sem tud jelentősen csökkenni. Az eredmények így is mutatják, hogy a keresés folyamatosan jó irányba haladt, az elitizmus pedig biztosította, hogy ne is tudjon letérni erről a pályáról.

Az mindenesetre látható, hogy a leállítást nem érdemes ahhoz igazítani, hogy mikor kerülnek az elemek egymáshoz megadott közelségbe, mivel a mutációk miatt ez nem fog bekövetkezni. Emiatt a keresést önkényesen leállítottam akkor, amikor már több száz generáción keresztül nem tudott jobb eredményt felmutatni. Persze nem zárható ki, hogy a mutációk miatt a későbbiekben még tudott volna javítani az eddig elért maximumon, de célszerűbb az erőforrásokat inkább egy új (részben akár a megadott eredményeken alapuló kezdő populációból kiinduló) keresés indítására fordítani.

[4] Kisebb érdekesség, hogy az elsőről a második generációra lépve egyszeri alkalommal növekszik a szórás. Részletesen megvizsgálva az egyes kromoszómákat, látható, hogy ez annak köszönhető, hogy a legelső generált generációban nagyon sok az életképtelen egyed, illetve azok a kromoszómák, amelyekkel lefuttatva a régiónövelést egy üres eredményt kapunk vissza. A sok hasonló egyed (mégha az eredményük nem is tekinthető jónak) alacsony szórást eredményezett.

#### 3.6.2.2. Paraméterek eloszlásának vizsgálata

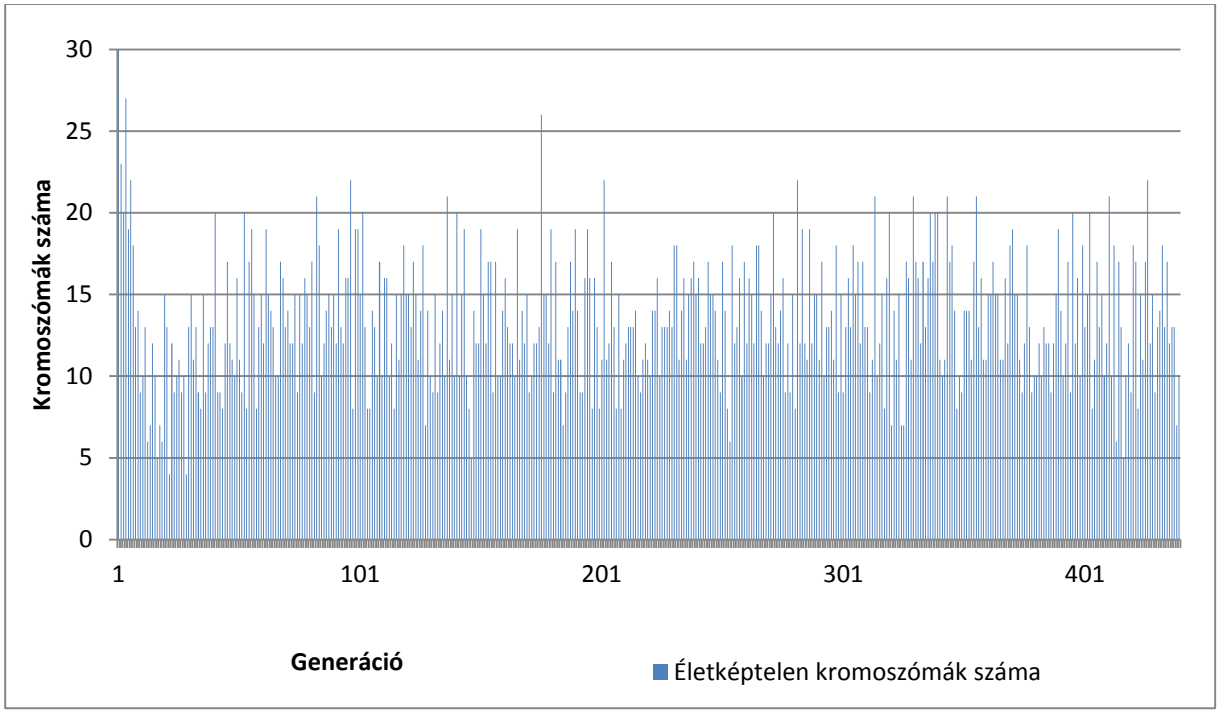
Hely hiányában nincs lehetőségem mind a 27 paraméter részletes bemutatására, emiatt csak néhány érdekesebb, kiragadott példát mutatok itt be (a hivatkozott diagramok a 2. függelékben találhatók meg). A paraméterek időbeli változását az előzőhöz hasonló diagramon ábrázoltam (vízszintes tengely a generáció, függőleges tengelyen pedig egy megadott paraméter esetében a generáció minden egyedéhez tartozó értékét ábrázoltam áttetsző szürke körökkel, a sötétebb részek értelemszerűen itt is egymást átfedő jeleket jeleznek). Elmondható, hogy három fő mintázatot vehetünk észre a paraméterek értékeinek változásában:

- A legtöbb paraméter meglehetősen gyorsan beállt a számára ideális értékre (általában már a 100. generáció környékén ez bekövetkezett), és ezt követően innen nem mozdult. A mutációk miatt természetesen megjelennek az ideális érték alatti és feletti értékek is, ezek azonban mind rövid életűnek bizonyulnak. Erre a mintára jó példa az 1. és a 2. paraméterek (16. diagram, 17. diagram), illetve valamivel hosszabb ideig tart, de ugyanez látható a 3. és 6. paraméterek (18. diagram, 19. diagram) esetében is.
- Kevésbé jellemző, de néhány paraméter esetében megjelenik az, amit a genetikus algoritmusok „iskolapéldájának" tekinthetünk, amikor különféle paraméter értékek (allélok) versenyeznek egymással. Ezekben az esetekben az egyes paraméter értékek meglehetősen hosszú ideig életképesek maradnak, néha 40-50 generáción keresztül valamelyik felerősödik, néha párhuzamosan 2-3 is domináns tud maradni. Végül azonban általában ezekben az esetekben is kialakult egy stabil állapot. Erre jó példa a 13., 19. és 24. paraméter (20. diagram, 21. diagram, 22. diagram).
- Ezek mellett egy-egy példát találhatunk olyan génekre is, amelyeknek az értéke még a leállítás előtti utolsó, 440. generációban sem tudott stabilizálódni. Erre példa a 9. és a 12. paraméterek (23. diagram, 24. diagram). Látható, hogy általában itt is csak két érték maradt versenyben, ezek is általában közvetlenül egymás mellettiek, ezek közül szinte mindegy, hogy melyiket választuk, így csak ezek miatt nem érdemes tovább futtatni a keresést.

### *3.6.3. Életképtelen egyedek számának vizsgálata*

Bár a végeredmény szempontjából ez kevésbé lényeges, de érdemes megvizsgálni az egyes generációkon belül az életképtelen egyedek számát (14. diagram).

Az első generáció generálása, illetve a későbbi kereszteződések és mutációk végrehajtása során nem végeztünk ellenőrzéseket azügyben, hogy az így létrejött egyedek paraméterei egymásnak ellentmondanak-e. Ugyanis éltem azzal a feltételezéssel, hogy ezek az életképtelen egyedek úgyis kihullanak majd a következő szülőválasztáskor, így remélhetőleg a későbbi generációk során egyre kevésbé fognak megjelenni.



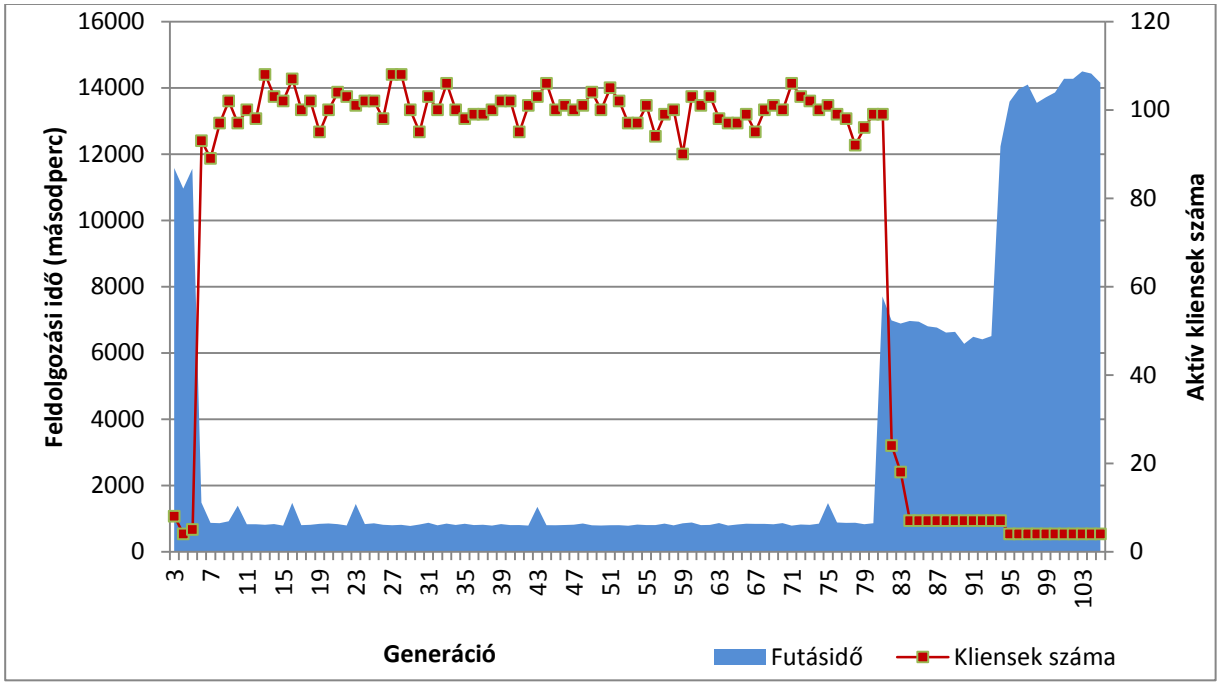

**14. diagram:** *Életképtelen kromoszómák száma generációnként.*

Az utólagos adatok ezt igazolni látszanak, a kezdő (generált) generáció adatai nem lettek ábrázolva, itt ugyanis 1879 volt az életképtelen egyedek száma. Azt ezt követő 5 generációban azonban ez a szám drasztikusan lecsökkent, majd a későbbiekben is átlagosan 13,2 környékén stabilizálódott. Mivel a keresztezések és főleg a mutációk a későbbiekben is okozhatnak életképtelen egyedeket, így ennek további csökkenésére már nem számíthatunk, ez a szám azonban tolerálható, ezek kiértékelése nem igényel drasztikusan sok erőforrást.

### *3.6.4. Futásidő vizsgálata*

Mivel a kutatásnak nem volt elsődleges célja (illetve korlátozott ideig fértünk csak hozzá a szükséges erőforrásokhoz, így célunk a minél nagyobb teljesítmény elérése volt, nem a különböző teljesítmény szintek mérése), ezért külön sebesség méréseket nem végeztem, viszont a kliensek futási naplóinak utólagos elemzéséből így is le lehet vonni néhány következtetést.

A rendszer működése során dinamikusan kapcsolódtak be és ki újabb kliensek, így itt találhatunk adatokat arról, hogy a kliensek számának változásával hogyan változott egy generáció feldolgozási ideje. Az ábrán (15. diagram) az vízszintes tengely az egyes generációkat mutatja, a háttérben lévő *kék terület* az egyes generációk teljes feldolgozási idejét, a *vörös pontok* pedig az adott generáció feldolgozásában résztvevő kliensek számát.




**15. diagram:** *Futásidő és az aktív kliensek száma generációnként.*

Jól látható, hogy a kliensek számának növekedésével azonnal csökkent a feldolgozási idő is, míg az utolsó generációknál az egyes kliensek lekapcsolásával ez ismét növekedni kezdett. Mivel az egyes végrehajtandó feladatok egymástól szinte teljesen függetlenek, illetve maguk a végrehajtó egységek is egymástól függetlenül működnek, joggal várnánk, hogy a kliensek számának növelésével lineárisan csökkenne a generációnkénti végrehajtási idő. Mindez azért nem következett be, mivel fizikailag csak 27 munkaállomással rendelkeztem, így a 100-at is meghaladó kliensszámot úgy értem el, hogy egy munkaállomáson több példányban is futott a kliens alkalmazás, így próbálva kiaknázni azok teljes kapacitását (4 magos processzorral rendelkező gépekhez igazítva, 4 példányban).

Ez ugyan valamelyest növelte a feldolgozási teljesítményt, mivel a kiértékelést végző algoritmus csak egy magot használ ki a rendelkezésre állók közül, így abból 4 példányt futtatva, valóban sikerült jobban kihasználni az erőforrásokat. Részben pedig nem jelentett változást, mivel a régiónövelést futtató algoritmus már eleve optimalizálva volt többmagos

környezetben (főleg az általa intenzíven használt OpenCV [**95**][**96**] függvényhívásokon keresztül), így ezekből egyidőben több példány futtatása nem jelentett különösebb előnyt.

A lineáris növekedés önmagában már azért sem elérhető, mivel az egyes csomagok feldolgozási ideje egymástól jelentősen különbözhet, így mindig előáll egy néhány perces állapot, amíg az egyik kliens még dolgozik az utolsó csomag feldolgozásán, miközben az összes többi már végzett, és a következő generáció indítására vár (a genetikus operátorok lefuttatása előtt szükséges az összes csomag eredményének beérkezése). Emiatt egy későbbi fejlesztés lehet a paraméterek alapján az egyes csomagok feldolgozási idejének becslése, és ez alapján egy kifinomultabb ütemezés [**97**] [**98**][**99**] kidolgozása.

### *3.6.5. Kontrollcsoport vizsgálata*

Mivel a régiónövelés és a kiértékelés meglehetősen számításigényes feladat, emiatt lehetőség szerint meg kellett próbálni csökkenteni a szükséges feldolgozások számát. Ennek egyik módja, hogy nem végezzük el a kiértékelést minden egyes rendelkezésre álló szövetmintára (összesen 41 darab), hanem azok közül kiválasztottam 11 darabot olyan módon, hogy lehetőleg minden típusú (sok-kevés sejtmag, éles-életlen kontúrok stb.) minta legalább egyszer szerepeljen (6. ábra).

A keresést csak erre a szűkebb listára hajtottam végre, ami a gyorsabb egyedfeldolgozás mellett azzal az előnnyel is jár, hogy a maradék szövetmintákat fel tudtam használni kontrollcsoportként. Ennek megfelelően, az összes rendelkezésre álló mintára lefuttatva a sejtmagkeresést, az alábbi eredményekhez jutottam:

- Az átlagos pontosság a régi paraméterkészlettel: 76,83%.
- Az átlagos pontosság az evolúciós algoritmus által talált paraméterkészlettel: 81,15%.

Mivel ezek a minták nem vettek részt az optimalizálásban, így ezzel jobban lehet becsülni az ismeretlen mintákra vonatkozó pontosság javulást, ami jelen esetben: 4,32%. Emellett az is látható mindkét eredményből, hogy a teljes mintakollekcióból kiválasztott 11 minta meglehetősen jól reprezentálta a teljes 41 elemű mintahalmazt (a két csoport pontossági vizsgálata egészen közeli értéket adott, 78,1% és 76,83%).

## 3.7.Eredmények értékelése

A kutatás célja egy olyan rendszer kiépítése volt, amely segítségével meg tudjuk találni az ideális paraméterkészletet az újonnan kifejlesztett GPGPU alapú adatpárhuzamos

régiónövelés számára. A meglévő módszerek áttekintése után egy evolúciós alapokon nyugvó algoritmus implementálása tűnt a legcélravezetőbbnek.

Ennek megfelelően kidolgoztam egy evolúciós algoritmust, amely alkalmas a régiónövelésen alapuló sejtmagkeresési eljárások nagy mennyiségű, egymástól függetlennek tekinthető paramétereinek optimalizálására. A mérési eredmények igazolják, hogy az általam kidolgozott módszer olyan új paraméterkészletet határoz meg, hogy az a korábban ismert legjobb paraméterkészletnél 4,3%-al jobb eredményt szolgáltat.

A genetikus algoritmus indítása előtt szükség volt a kezdő generáció generálásához szükséges intervallumok meghatározására. Ehhez statisztikai módszerekkel a rendelkezésre álló manuálisan annotált minták alapján megállapítottam a sejtmagkeresés paramétereit optimalizáló genetikus algoritmus számára szükséges kezdő generáció paramétereinek ideális értéktartományait.

Végül megterveztem és implementáltam egy olyan keretrendszert, amely alkalmas genetikus algoritmusok elosztott futtatására, miközben moduláris felépítésének köszönhetően támogatja a genetikus operátorok és a kommunikációs protokollok szabad megválasztását. A mérési eredmények igazolják, hogy a régiókeresési eljárás paramétereinek optimalizálása során az elosztott rendszer futásideje jelentősen kisebb, mint a hagyományos szekvenciális megoldás időigénye.

A tapasztalatok alapján egy továbbfejlesztési irányt jelenthet egy specializáltabb mintakörre irányuló paraméteroptimalizálás. Mivel tetszőlegesen kiválasztható, hogy a kliensek mely szövetmintákra végezzék el a kiértékelést és ellenőrzést, esetleg célszerű lehet szűkíteni ezt a kört megadott típusú mintákra (megadott festési mód, adott labor, adott szövet típus stb.), így remélhetőleg található még nagyobb pontosságot eredményező paraméterkészlet a meglévő algoritmusok számára.

Emellett érdemes lehet a evolúciós algoritmus feldolgozási sebességét néhány továbbfejlesztéssel tovább növelni. Az egyik út lehet egy hibrid rendszer felé való elmozdulás, egy másik kézenfekvő lehetőség pedig egy kifinomultabb ütemezés kialakítása, hogy a generációk utolsó kromoszómáját feldolgozó kliensre minél kevesebbet kelljen várni a többi, a feladatával már elkészült kliensnek.

# ÖSSZEGZÉS (TÉZISEK)

Az orvosi célú digitális képfeldolgozás használata napjainkban egyre elterjedtebb a patológusok körében. A legújabb rendszerekben elérhetővé vált, hogy a szövettani vizsgálatok által igényelt lépések nagy része automatizálható (festések, tárgylemezek továbbítása, digitális felvételek készítése stb.), és várhatóan ez a trend a jövőben is folytatódik [**1**]. Az újszerű eszközök a szövetminták feldolgozásának közvetlen előnyei mellett (nagyfelbontású, jó minőségű, jól fókuszált képek) számos új lehetőség előtt nyitották meg az utat, a telepatológiai rendszerekben ugyanis nincs szükség többé a tárgylemezek fizikai továbbítására, hanem hálózaton keresztül is van lehetőség a nagyfelbontású felvételek katalogizálására, megosztására, reprodukciójára, távoli elérésére. Az így kiépült rendszerek elterjedése, a hardvereszközök teljesítményének növekedése, és a képfeldolgozási eljárások fejlődése együttesen készítette elő a következő lépést, amely a képfeldolgozó eljárások megjelenését jelenti a napi rutin diagnosztikában.

A szoftveres környezet változásán túlmenően az utóbbi években jelentősen megváltozott a hardver lehetőségek tárháza is. Az informatikában egészen a 2000-es évekig megszokottá vált, hogy a különböző processzorok teljesítménye évről-évre folyamatosan növekszik, miként ezt a közkedvelt Moore-törvény előre meg is jósolta. Ez a nagyon egyszerű jóslat hosszú éveken keresztül időtállónak bizonyult, napjainkban azonban ez a dinamikus fejlődés megtorpant, vagy legalábbis jelentősen irányt változtatott [**15**]. Az utóbbi években a legszembetűnőbb a többmagos processzorok megjelenése [**16**], amelyek az évenkénti teljesítménynövekedést egy meglehetősen kézenfekvő ötlettel biztosítják: egy processzoron belül bizonyos egységeket egyszerűen megdupláztak (négyszereztek, nyolcszoroztak), amelynek segítségével az eszköz elméleti számítási kapacitása továbbra is növekszik, kielégítve a piac igényeit.

A többmagos processzorok megjelenésén túlmenően azonban időközben megjelentek egészen újszerű architektúrák is, mint például az általános célú számításokra használható grafikus kártyák (GPGPU-k). Az eredetileg a monitoron látható kép megjelenítésére kialakított hardvereszközök idővel különféle 3D számítások végrehajtására lettek alkalmasak, az ipari (és nem kevésbé az otthoni játékos) felhasználók igényeinek megfelelően egyre több egyszerű végrehajtóegységgel rendelkeztek; majd különféle *shader* generációváltások [**17**] után felmerült a lehetőség, hogy általános célú programok futtatására is alkalmasak legyenek. Ennek adott nagy lökést az Nvidia 2007-es lépése, amikor kiadott egy szoftverfejlesztési környezetet is a saját kártyáihoz (*CUDA 1.0*). A fejlesztések azonban meglehetősen nehézkesen haladnak, ugyanis az újszerű eszköz számos korláttal és speciális jellemzővel bír a

hagyományos CPU-khoz viszonyítva, továbbá hiányoznak a kiforrott eszközök, és a több éves tapasztalatok is, amelyek a hagyományos fejlesztéseknél már rendelkezésre állnak.

## 1. téziscsoport

Számos sejtmagkeresési eljárással találkozhatunk, ezek általában jósági vizsgálatokkal is járnak, ezek azonban különféle módszereket alkalmaznak az értékeléshez, így eredményük egymással nem összehasonlítható, másrészt az értékelések csak a hagyományos képfeldolgozó algoritmusokra kidolgozott általános módszereken alapulnak (pl. döntési tábla) és nem veszik figyelembe a sejtmagkeresési eljárások által támasztott speciális követelményeket.

Leggyakrabban a szegmentálási algoritmusok pontosságát egy teszt és egy referencia eredmény pixelenkénti összehasonlításával szokták meghatározni. Ez ugyan egy nagyon jól kezelhető mérőszámot ad arra vonatkozólag, hogy a kép pixeleit egymástól teljesen függetlenül vizsgálva, az egyes pixelekre vonatkozóan az algoritmus jól állapította-e meg, hogy azok egy sejtmag részei vagy sem. A gyakorlatban azonban ez nem ad kielégítő eredményt, hiszen gyakori, hogy több egymáshoz közel álló kisebb sejtmagot a szegmentáló algoritmus egy nagy sejtmagként határoz meg, amit ez az egyszerű ellenőrzés tökéletes eredményként fogadna el. Ehhez hasonlóan, a gyakorlatban nagyobb hibának szeretnénk tekinteni, ha egy rendszer sok kisebb sejtmagot nem talált meg, mintha csak egy darab ugyanilyen területű nagy sejtmag maradt ki.

**1.1. tézis: Eljárást dolgoztam ki, amely ötvözi a pixel- és objektumszintű értékelések előnyeit, így a meglévő módszereknél jobban alkalmazható szövetmintákban sejtmagokat kereső szegmentáló eljárások objektív összehasonlítására.**

Az általam kifejlesztett módszer [**47**] nem csak a fent említett pixel szintű összehasonlítást végzi el, hanem először megpróbálja egymáshoz rendelni a teszt és a referencia eredményben található sejtmagokat, majd ezekre a sejtmagokra végzi el a részletes összehasonlítást. Ezzel a pontossági mérőszám tartalmazza annak az ellenőrzését is, hogy valóban a referencia eredményben meglévő számú és elhelyezkedésű sejtmagot talált-e a szegmentáló algoritmus, illetve ezen túlmenően a párokra a pixelenkénti összehasonlítást alkalmazva információt ad az egyes megtalált sejtmagok alakjának megfelelőségéről is.

**1.2. tézis: Kidolgoztam egy keresési algoritmust, amely nagy mennyiségű, egymást átfedő referencia és teszt sejtmagokat tartalmazó szegmentálási eredmények esetén a**

**hagyományos keresésekhez képest kevesebb lépéssel határozza meg a legnagyobb pontossági értéket nyújtó párosításokat.**

Az előzőleg felvázolt pontossági mutató a gyakorlatban jól használható, azonban kiszámítása meglehetősen erőforrás-igényes. A pixelenkénti vizsgálat előtt ugyanis szükség van az egyes teszt illetve referencia sejtmagok egymáshoz rendelésére, ez pedig egymást átfedő sejtmagok (illetve így az átfedéseken keresztül kialakuló hosszú láncok) miatt nem egyértelmű. A legnagyobb pontosságot nyújtó párosítás megtalálása több ezer sejtmag esetén használhatatlan futásidőt eredményezne, ezért kidolgoztam egy módszert, ami az egymást átfedő sejtmagokat klaszterekbe rendezi [**35**], majd egy speciális visszalépéses keresésnek köszönhetően a lineáris kereséshez képest kevesebb lépéssel meghatározza a legjobb referencia-teszt sejtmag párosítást [**47**].

## 2. téziscsoport

A szöveti mintákon sejtmagok keresésére számos szegmentáló algoritmus létezik, ezek közül az egyik legígéretesebb a különféle régiónövelési eljárások köre. Az algoritmus alapvető lépéseinek (előfeldolgozás, kezdőpont keresés, régiónövelés, utófeldolgozás) egy részét, illetve egyes rész műveleteket (szűrők, konvolúciós műveletek) már sikerült jól párhuzamosítani, a minél rövidebb futásidő érdekében azonban érdemes a kezdőpont keresés és a régiónövelés számára is kidolgozni új, adatpárhuzamos környezetben is működő algoritmusokat.

### 2.1. tézis: Kidolgoztam egy adatpárhuzamos régiónövelési eljárást, amely alkalmas szöveti mintákon párhuzamosan több kiindulópontból kiindulva sejtmagok detektálására.

Számítási kapacitás tekintetében a GPGPU alapú platformok jelentik az egyik legjobb alternatívát napjainkban [**18**], emiatt a megvalósított algoritmus alkalmazkodik a rendszer által megkívánt kötöttségekhez. Mivel a GPGPU esetében több száz végrehajtó egységgel rendelkezünk, és a szálak számát még ennél is jóval nagyobbnak érdemes választani azok teljes leterheléséhez, emiatt az algoritmus a lehető legnagyobb mértékű párhuzamosítást használja több szinten is: régiónövelés során az egyes kontúrpontok adatait külön szálak dolgozzák fel, emellett egy magasabb szinten, egyidőben több régiónövelés is futhat párhuzamosan egy kártyán [**40**].

**2.2. tézis: Kidolgoztam egy adatpárhuzamos régiónövelési kiindulópont keresési eljárást, amely alkalmas a régiónövelési eljárás számára egyidőben több kezdőpont kigyűjtésére.**

Az egyes régiónövelések közötti kezdőpont keresés során szintén érdemes kihasználni a GPGPU által nyújtott lehetőségeket, ez pedig új, adatpárhuzamos keresési módszereket igényel, ami a szinkronizációs műveletek nehézsége miatt jelent különösen nagy kihívást. A jelentős sebességnövekedésen túlmenően ez amiatt is szükséges, mivel az előbb említett új régiónövelési módszer egy időben több növelést is tud kezelni párhuzamosan, így a kezdőpont keresést is úgy valósítottam meg, hogy ne csak egy pontot, hanem egyidőben több lehetséges kiinduló pontot is keressen, mivel csak így tudja maximálisan táplálni az ezt követő növeléseket [**40**].

**2.3. tézis: Igazoltam, hogy az újonnan kifejlesztett adatpárhuzamos régiónövelési eljárás (kiindulópont keresés és régiónövelés) a jelenleg meglévő hagyományos régiónövelési eljárásokhoz képest azonos pontosságot ér el jelentősen kisebb futásidő mellett.**

Implementáltam CUDA környezetben az adatpárhuzamos régiónövelési algoritmust, majd a fent ismertetett módszerekkel ellenőriztem a módszer pontosságát, illetve futásidejét. A vizsgálat eredményeképpen megállapítottam, hogy a CPU és a GPGPU alapú régiónövelés pontossága gyakorlatilag azonos, míg a GPGPU alapú változat futásideje általában fele/harmada a CPU alapúnak [**40**][**8**].

## 3. téziscsoport

**3.1. tézis: Kidolgoztam egy evolúciós algoritmust, amely alkalmas a régiónövelésen alapuló sejtmagkeresési eljárások nagy mennyiségű, egymástól függetlennek tekinthető paramétereinek optimalizálására. A mérési eredmények igazolják, hogy az általam kidolgozott módszer olyan új paraméterkészletet határoz meg, hogy az a korábban ismert legjobb paraméterkészletnél 4,3%-kal jobb eredményt szolgáltat.**

A régiónövelési algoritmus meglehetősen nagy számú paramétert igényel (27 darab), ezek értékkészlete is meglehetősen nagy, illetve ismeretlen az egyes paraméterek egymásra vonatkozó hatása, így az optimális paraméterkészlet manuális megkeresése szinte lehetetlen, de még a hagyományos lineáris kereséssel is valószínűleg évekbe telne. Emiatt kidolgoztam egy genetikai algoritmust, amely alkalmas arra, hogy meghatározzon egy, a gyakorlatban is jól használható paraméterkészletet. A gyakorlati tesztek során kiderült, hogy a módszer beváltotta a hozzá fűzött reményeket, 440. generáció után a módszer által ajánlott

paraméterkészlet már 4,3%-al nagyobb pontosságot nyújtott, mint az általunk használt régiónövelési implementációhoz tartozó jelenleg ismert legjobb paraméterkészlet [**46**].

**3.2. tézis: Statisztikai módszerekkel a rendelkezésre álló manuálisan annotált minták alapján megállapítottam a sejtmagkeresés paramétereit optimalizáló genetikai algoritmus számára szükséges kezdő generáció paramétereinek ideális értéktartományait.**

A genetikai algoritmus számára kritikus lehet az első induló generációban található kromoszómák létrehozása. Hogy ezek minél közelebb kerüljenek az elvárt eredményhez, a rendelkezésre álló, manuálisan annotált mintákon található sejtmagok alapján különféle statisztikai módszerekkel megállapítottam néhány paraméterre olyan határértékeket [**65**] (sejtmag méret, intenzitás, körszerűség stb.), amelyek között célszerű a kezdő generáció génjeinek konkrét értékeit megválasztani.

**3.3. tézis: Megterveztem és implementáltam egy olyan keretrendszert, amely alkalmas genetikai algoritmusok elosztott futtatására, miközben moduláris felépítésének köszönhetően támogatja a genetikai operátorok és a kommunikációs protokollok szabad megválasztását. A mérési eredmények igazolják, hogy a régiókeresési eljárás paramétereinek optimalizálása során az elosztott rendszer futásideje jelentősen kisebb, mint a hagyományos szekvenciális megoldás időigénye.**

A genetikai algoritmusok általában nagy számításigénnyel rendelkeznek, ez ebben az esetben kiemelten igaz, mivel a jósági függvény kiértékeléséhez az egyes paraméterkészletekkel el kell végezni egy régiónövelést illetve egy pontossági kiértékelést egymás után több szövetmintára is. A megoldást ebben az esetben is egy elosztott rendszer jelentette, amely a nagy erőforrás-igényre tekintettel egy hálózatokon keresztül összekapcsolt gépeken futó, több száz kliens egyidejű futtatását is lehetővé teszi. Megterveztem egy ezeknek a feltételeknek megfelelő rendszert [**86**], mindezt olyan szempontok szerint, hogy mind az egyes genetikai operátorok, mind pedig a kliensek es a szerver közötti kommunikáció formája tetszőlegesen megválasztható legyen. A módszernek köszönhetően az evolúciós algoritmus futásideje jelentősen csökkent.

# AZ EREDMÉNYEK HASZNOSÍTÁSA, TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

A képfeldolgozás egy meglehetősen nagy területet lefedő témakör, aminek jelen kutatás csak egy nagyon specializált területét, a szövetmintákon való sejtmagok keresést célozta meg. A vizsgált terület azonban gyakorlati szempontból nagyon fontosnak tekinthető, hiszen napjainkban egyre inkább várhatjuk a különböző automatikus diagnosztikai rendszerek megjelenését. Amikor konkrét betegek diagnosztikájáról beszélhetünk, akkor talán nem is kell részletesen elemezni, hogy milyen sokat jelent akár néhány százalékkal pontosabb kiértékelés, ami az evolúciós algoritmus által nyújtott jobb paraméterkészlet alapján elérhetővé vált.

Az általam kidolgozott GPGPU alapú régiónövelés azonos pontosság mellett kétszer/háromszor gyorsabb futásidőt eredményez. A relatív gyorsulás mellett érdemes figyelembe venni azt is, hogy a hagyományos CPU alapú régiónövelés egy nagyobb szövetmintát közel egy órás futásidővel tud csak feldolgozni, ehhez képest az elérhető 20 perc körüli eredmény nem csak egyszerű gyorsítást jelent, hanem a régiónövelést felveszi a gyakorlatban is használható módszerek közé.

Az általam kidolgozott pontossági mutató pedig a későbbiekben lehetőséget nyújt az újonnan megjelenő sejtmagkeresési eljárások összehasonlítására, ahol külön érdekes lehet az egyes újabb fejlesztések és a megelőző változatok közötti különbségek analízisére. A mutató a gyakorlatban jól használhatónak bizonyult, hiszen ezen a mutatón alapul az általam kidolgozott paraméter optimalizálás, illetve a GPGPU alapú régiónövelés pontosságának igazolása is.

# FELHASZNÁLT IRODALOM


[1] L. Ficsór, "Digital Microscopy in the Diagnosis of Gastrointestinal Histological Samples," Semmelweis University, 2nd Department of Internal Medicine, Budapest, Phd Thesis 2009.

[2] A. Nagy, Z. Vámossy, "Super-resolution for Traditional and Omnidirectional Image Sequences," *Acta Polytechnica Hungarica*, vol. 6, no. 1, pp. 117-130, 2009.

[3] E. Sorantin, E. Balogh, A. Vilanova i Bartrolí, K. Palágyi, L. G. Nyúl, F. Lindbichler, A. Ruppert, "Techniques of Virtual Dissection of the Colon Based on Spiral CT Data," in *Image Processing in Radiology: Current Applications*, 2008, pp. 257-268.

[4] M. E. Adawy, Z. Shehab, H. Keshk, M. E. El Shourbagy, "A Fast Algorithm for Segmentation of Microscopic Cell Images," in *ITI 4th International Conference on Information & Communications Technology (ICICT '06)*, 10-12 Dec. 2006, pp. 1-1.

[5] Y. Surut, P. Phukpattaranont, "Overlapping Cell Image Segmentation using Surface Splitting and Surface Merging Algorithms," in *Second APSIPA Annual Summit and Conference*, 2010, pp. 662-666.

[6] J. Hukkanen, A. Hategan, E. Sabo, I. Tabus, "Segmentation of Cell Nuclei From Histological Images by Ellipse Fitting," in *18th European Signal Processing Conference (EUSIPCO-2010)*, Aalborg, 23-27 Aug. 2010, pp. 1219-1223.

[7] R. Pohle, K. D. Toennies, "Segmentation of medical images using adaptive region growing," *SPIE*, vol. 4322, pp. 1337-1346, Jul. 2001.

[9] Pannon University, "Algoritmus- és forráskódleírás a 3DHistech Kft. számára készített sejtmag-szegmentáló eljáráshoz," Veszprém, 2009.

[10] L. Ficsór, V. S. Varga, A. Tagscherer, Zs. Tulassay, B. Molnár, "Automated classification of inflammation in colon histological sections based on digital microscopy and advanced image analysis," *Cytometry*, vol. 73A, no. 3, pp. 230–237, March 2008.

[11] L. Krecsák, T. Micsik, G. Kiszler, T. Krenács, D. Szabó, V. Jónás, G. Császár, L. Czuni, P. Gurzó, L. Ficsor, B. Molnár, "Technical note on the validation of a semi-automated image analysis software application for estrogen and progesterone receptor detection in breast cancer," *Diagnostic Pathology*, vol. 6, Jan. 2011.

[12] K. E. Emblem, F. G. Zoellner, A. Bjornerud, "A fully automated method for predicting glioma patient outcome from DSC imaging. A second reference to histopathology?," in *International Society for Magnetic Resonance in Medicine*, 2013, p. 281.

[13] A. K. Jain, "Data clustering: 50 years beyond K-means," *Award winning papers from the 19th International Conference on Pattern Recognition* , vol. 31, no. 8, pp. 651-666, June 2010.

[14] E. C. Smochină, "Image Processing Techniques and Segmentation Evaluation," Technical University "Gheorghe Asachi", Doctoral School of the Faculty of Automatic Control and Computer Engineering,.

[15] D. Patterson, "The trouble with multi-core," *Spectrum*, vol. 47, no. 7, pp. 28-32, July 2010.

[16] P. Gepner, M. F. Kowalik, "Multi-Core Processors: New Way to Achieve High System Performance," in *International Symposium on Parallel Computing in Electrical Engineering (PAR ELEC)*, 13-17 Sept. 2006, pp. 9-17.

[17] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, J. C. Phillips, "GPU Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879-899, May 2008.

[19] N. Goswami, R. Shankar, M. Joshi, L. Tao, "Exploring GPGPU workloads: Characterization methodology, analysis and microarchitecture evaluation implications," in *International Symposium on Workload Characterization (IISWC)*, 2-4 Dec. 2010, pp. 1-10.

[20] L. Hu, X. Che, Z. Xie, "GPGPU cloud: A paradigm for general purpose computing," *Tsinghua Science and Technology*, vol. 18, no. 1, pp. 22-23, Feb. 2013.

[21] J. Tianzi, Y. Faguo, F. Yong, D. J. Evans, "A Parallel Genetic Algorithm for Cell Image Segmentation," in *8th International Workshop on Combinatorial Image Analysis (IWCIA 2001)*, vol. 46, Philadelphia, 23-24 Aug. 2001, pp. 214-224.

[22] X. Du, S. Dua, "Segmentation of Fluorescence Microscopy Cell Images Using Unsupervised Mining," *The Open Medical Informatics Journal*, vol. 4, pp. 41-49, May 2010.

[23] W. A. Yasnoff, J. K. Mui, J. W. Bacus, "Error measures for scene segmentation," *Pattern Recognitio*, vol. 9, no. 4, pp. 217-231, 1977.

[24] S. Timmermans, M. Berg, *The Gold Standard-The Challenge of Evidence-Based Medicine and Standardization in Health Care*. Philadelphia: Temple University Press, 2003.

[25] R. Kohavi, F. Provost, "Glossary of Terms," *Machine Learning*, vol. 30, no. 2, pp. 271-274, 1998.

[26] R. E. Precup, S. Preitl S, J. K. Tar, M. L. Tomescu, M. Takács, P. Korondi, P. Baranyi, "Fuzzy control system performance enhancement by iterative learning control," in *IEEE Transactions on Industrial Electronics*, 2008, pp. 3461-3475.

[27] E. Tóth-Laufer, M. Takács, I.J. Rudas, "Conjunction and Disjunction Operators in Neuro-Fuzzy Risk Calculation Model Simplification," in *13th IEEE International Symposium on Computational Intelligence and Informatics (CINTI)*, Budapest, 20-22 Nov. 2012, pp. 195-200.

[28] P. Correia, F. Pereira, "Objective Evaluation of Relative Segmentation Quality," in *International Conference on Image Processing (ICIP00)*, vol. 1, 2000, pp. 308-311.

[29] M. Everingham, H. Muller, B. Thomas, "Evaluating Image Segmentation Algorithms using the Pareto Front," in *7th European Conference on Computer Vision*, 28-31 May 2002, pp. 34-48.

[30] H. J. G. Bloom, W. W. Richardson, "Histological Grading and Prognosis in Breast Cancer," *Pathol. Annu*, vol. 15, pp. 359-377, 1980.

[31] W. A. Christens-Barry, A. W. Partin, "Quantitative Grading of Tissue and Nuclei in Prostate Cancer for Prognosis Prediction," *Johns Hopkins APL Technical Digest*, vol. 18, no. 2, pp. 226-232, 1997.

[32] H. S. Wu, J. Gil, "A Biased Median Filtering Algorithm for Segmentation of Intestinal Cell Gland Images," *The Scientific World Journal*, vol. 6, pp. 200-220, Feb. 2006.

[34] J. Han, M. Kamber, *Data Mining. Concepts and Techniques*.: Elsevier Inc., 2001.

[36] M. T. Goodrich, R. Tamassia, *Algorithm Design*.: John Wiley & Sons, Inc., 2002.

[37] B. Selman, K. McAloon, C. Tretkoff C. P. Gomes, "Randomization in Backtrack Search: Exploiting Heavy-Tailed Profiles for Solving Hard Scheduling Problems," in *Fourth International Conference on Artificial Intelligence Planning Systems*, Menlo Park, Jun. 1998.

[38] V. P. Guddeti, B. Y. Choueiry, "An Empirical Study of a New Restart Strategy for Randomized Backtrack Search," in *CP 2004 Workshop on CSP Techniques with Immediate Application (CSPIA)*, Toronto, 27 Sept. 2004, pp. 66-82.

[39] D. E. Knuth, "Estimating the efficiency of backtrack programs," *Mathematics of Computation*, vol. 29, pp. 122-136, 1975.

[41] A. Troelsen, *Pro C# 2010 and the.NET 4 Platform*, 5th ed.: Apress, 2010.

[42] J. Richter, *CLR via C# (Microsoft, Developer Reference)*, 4th ed.: Microsoft Press, 2012.

[43] C. Nagel, B. Evjen, J. Glynn, K. Watson, M. Skinner, *Professional C# 2012 and.NET 4.5*.: Wrox, 2012.

[45] V. Podlozhnyuk, "Image Convolution with CUDA," NVIDIA, 2007.

[48] S. Sergyán, L. Csink, "Automatic Parameterization of Region Finding Algorithms in Gray Images," in *4th International Symposium on Applied Computational Intelligence and Informatics*, Timisoara, 17-18 May. 2007, pp. 199-202.

[49] Z. Krizsán, Sz. Kovács, "Gradient based parameter optimisation of FRI "FIVE"," in *9th International Symposium of Hungarian Researchers on Computational Intelligence (CINTI)*, Budapest, 6-8 Nov. 2008, pp. 531-538.

[50] D. G. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston: Addison-Wesley Longman Publishing Co., 1989.

[51] C. Carrick, K. MacLeod, "An evaluation of genetic algorithm solutions in optimization and machine learning," in *21st Annual Conference Canadian Association for Information Science CAIS/ACSI'93*, Antigonish, 12-14 July 1993, pp. 224–231.

[52] T. H. Cormen, C. E. Leiserson, R. L. Rivest,.: Mcgraw-Hill College, 1990, ch. 17, p. 329.

[53] Z. Blázsik, Cs. Imreh, Z. Kovács, "Heuristic algorithms for a complex parallel machine scheduling problem," *Central European Journal of Operations Research*, pp. 379-390, 2008.

[54] J. L. Noyes, *Artificial Intelligence With Common Lisp*.: D.C. Health and Company, 1992.

[55] S. N. Sivanandam, S. N. Deepa, *Introduction to Genetic Algorithms*.: Springer, 2008.

[56] M. Gendreau, J.-Y. Potvin, *Handbook of Metaheuristics*.: Springer, 2010.

[57] D. Bersimas, J. Tsitsiklis, "Simulated Annealing," *Statistical Science*, vol. 8, no. 1, pp. 10-15, 1993.

[58] A. Álmos, S. Győri, G. Horváth, A. Várkonyiné Kóczy, *Genetikus algoritmusok*.: Typotex, 2002.

[59] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.

[60] C. Reews, "Genetic algorithms," Coventry University, Heuristics in Optimization course 2012.

[61] M. Mitchell, "Genetic algorithms: An Overview," *Complexity*, vol. 1, no. 1, pp. 31-39, 1995.

[62] Y.-S. Choi, B.-R. Moon, "Parameter Optimization by a Genetic Algorithm for a Pitch Tracking System," in *International conference on Genetic and evolutionary computation: PartII, GECCO'03*, 2003, pp. 2010-2021.

[63] A. Bevilacqua, R. Campanini, N. Lanconelli, "A Distributed Genetic Algorithm for Parameters Optimization to Detect Microcalcifications in Digital Mammograms," in *EvoWorkshop 2001*, 2001, pp. 278-287.

[64] L. Budin, M. Golub, A. Budin, "Traditional Techniques of Genetic Algorithms Applied to Floating-Point Chromosome Representations," in *41st Annual Conference KoREMA*, Opatija , 1996, pp. 93-96.

[66] M. Mitchell, *An introduction to genetic algorithms*. Cambridge: Bradfork Book The MIT Press, 1999.

[67] D. Gupta, S. Ghafir, "An Overview of methods maintaining Diversity in Genetic Algorithms," *International Journal of Emerging Technology and Advanced Engineering*, vol. 2, no. 5, pp. 56-60, May 2012.

[68] W. M. Spears, D. Jong, D. Kenneth, "On the Virtues of Parameterized Uniform Crossover," Washington, Naval Research Lab 1995.

[69] D. Jong, A. Kenneth, W. M. Spears, "An analysis of the interacting roles of population size and crossover in genetic algorithms," in *Parallel problem solving from nature*, 1991, pp. 38-47.

[70] K. Messa, M. Lybanon, "Curve Fitting Using Genetic Algorithms," Naval Oceanographic and Atmospheric Research Lab., Stennis Space Center MS., NTIS issue number 9212,.

[71] R. Murphy, "A Generic Parallel Genetic Algorithm," University of Dublin, Department of Mathematics, M.Sc. Thesis in High Performance Computing.

[72] E. Alba, M. Tomassini, "Parallelism and Evolutionary Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 443-462, Oct. 2002.

[73] D. Sima, "Decisive aspects in the evolution of microprocessors," *Proceedings of the IEEE*, vol. 92, no. 12, pp. 1896-1926, Dec 2004.

[74] D. Sima, T. Fountain, P. Kacsuk, *Advanced Computer Architectures: A Design Space Approach*.: Addison-Wesley, 1997.

[75] E. Alba, J. M. Troya, "A Survey of Parallel Distributed Genetic Algorithms," *Complexity*, vol. 4, no. 4, pp. 31-52, March/April 1999.

[76] M. Nowostawski, R. Poli, "Parallel genetic algorithm taxonomy," in *Third International Conference Knowledge-Based Intelligent Information Engineering Systems*, Dec. 1999, pp. 88-92.

[77] P. Adamidis, "Parallel Evolutionary Algorithms: A Review," in *4th Hellenic-European Conference on Computer Mathematics and its Applications*, 1998.

[78] A. Muhammad, A. Bargiela, G. King, "Fine-Grained Parallel Genetic Algorithm: A Stochastic Optimisation Method," in *The First World Congress on Systems Simulation*, 1997, pp. 199-203.

[79] S. Baluja, "Structure and Performance of Fine-Grain Parallelism in Genetic Search," in *5th International Conference on Genetic Algorithms*, 1993, pp. 155-162.

[80] J. D. Lohn, S. P. Colombano, G. L. Haith, D. Stassinopoulos, "A Parallel Genetic Algorithm for Automated Electronic Circuit Design," NASA Ames Research Center, 2000.

[81] M. Golub, L. Budin, "An Asynchronous Model of Global Parallel Genetic Algorithms," in *Second ICSC Symposium on Engineering of Intelligent Systems EIS2000*, University of Paisley, 2000, pp. 353-359.

[82] M. Dubreuil, C. Gagne, M. Parizeau, "Analysis of a master-slave architecture for distributed evolutionary computations," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 36, no. 1, pp. 229-235, Feb. 2006.

[83] M. Babbar, B. S. Minsker, "A Multiscale Master-Slave Parallel Genetic Algorithm with Application to Groundwater Remediation Design," in *Genetic and Evolutionary Computation Conference (GECCO-2002)*, New York, 9-13 July 2002, pp. 9-16.

[84] U. Kohlmorgen, H. Schmeck, K. Haase, "Experiences with Fine-Grained Parallel Genetic Algorithms," *Annals of Operations Research*, pp. 203-219, 1996.

[85] I. Rudas, "Hybrid Systems: Integration of Neural Networks, Fuzzy Logic, Expert Systems, and Genetic Algorithms," *Encyclopedia of Information Systems, Boston: Academic Press*, vol. 114, pp. 1-8, 2002.

[87] I. Foster, Z. Yong, I. Raicu, L. Shiyong, "Cloud Computing and Grid Computing 360-Degree Compared," in *Grid Computing Environments Workshop (GCE '08)*, 12-16 Nov. 2008, pp. 1-10.

[88] I. Foster, C. Kesselman, S. Tuecke, "The anatomy of the Grid: Enabling scalable virtual organization," *The Intl. Jrnl. of High Performance Computing Applications*, vol. 15, no. 3, pp. 200-222, 2001.

[89] M. Sarnovský, P. B., J. Paralič, "Grid-based Support for Different Text Mining Tasks," *Acta Polytechnica Hungarica*, vol. 6, no. 4, pp. 5-27, 2009.

[90] M. Kozlovszky, A. Balasko, P. Kacsuk, "Enabling JChem on the Grid," in *The International Symposium on Grids and Clouds and the Open Grid Forum*, Taipei, 19-25 March 2011, p. 119.

[91] D. Thain, T. Tannenbaum, M. Livny, "Distributed Computing in Practice: The Condor Experience," *Concurrency and Computation: Practice and Experience*, vol. 17, no. 2-4, pp. 323-356, Feb-Apr. 2005 2005.

[92] N. S. Arora, R. D. Blumofe, C. G. Plaxton, "Thread Scheduling for Multiprogrammed Multiprocessors," *Theory of Computing Systems*, vol. 34, pp. 115-144, 2001.

[93] J. Postel, J. Reynolds, "File Transter Protocol (FTP)," Network Grouping Group, RFC 959, 1985.

[94] R. Isermann, *Fault-Diagnosis Systems*.: Springer, 2006.

[95] G. Bradski, A. Kaehler, *Learning OpenCV*.: O'Reilly Media Inc., 2008.

[96] A. Nagy, Z. Vámossy, "OpenCV C# Wrapper-based Video Enhancement Using Different Optical Flow Methods in the Super-resolution," in *Intelligent Systems and Informatics*, Subotica, 26-27 Sept. 2008, pp. 1-6.

[97] C. Banino, O.Beaumont; L. Carter, J. Ferrante, A. Legrand, Y. Robert, "Scheduling strategies for master-slave tasking on heterogeneous processor platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 4, pp. 319-330, Apr. 2004.

[98] D. K. Kiss, A. Rovid, "Multi-core processor needs from scheduling point of view," in *8th International Symposium on Intelligent Systems and Informatics (SISY)*, 10-11 Sept. 2010, pp. 219-224.

[99] Cs. Imreh, "Online scheduling with general machine cost functions," *Discrete Applied Mathematics*, vol. 157, no. 9, pp. 2070-2077, 2009.

# SAJÁT PUBLIKÁCIÓK


[8] A. Reményi, S. Szénási, I. Bándi, Z. Vámossy, G. Valcz, P. Bogdanov, Sz. Sergyán, M. Kozlovszky, "Parallel Biomedical Image Processing with GPGPUs in Cancer Research," in *3rd IEEE International Symposium on Logistics and Industrial Informatics (LINDI 2011)*, Budapest, 25-27 Aug. 2011, pp. 245–248.

[18] S. Szénási, "GPGPU - Új eszközök az információbiztonság területén," Hadmérnök, vol. 4, no. 4, pp. 362-373, Dec. 2009.

[33] D. Kotsis, G. Légrádi, G. Nagy, S. Szénási, Többnyelvű programozástechnika. Budapest: Panem, 2007.

[35] S. Szénási, D. Jankó, "Black spot treatment system using a „hunting for irregular pattern" process and a safety knowledge-base" On safe roads in the XXI. Century,Budapest, 2006.

[40] S. Szénási, Z. Vámossy, M. Kozlovszky, "GPGPU-based data parallel region growing algorithm for cell nuclei detection," in 12th IEEE International Symposium on Computational Intelligence and Informatics (CINTI), Budapest, 21-22 Nov. 2011, pp. 493-499.

[44] S. Szénási, "Java programozási nyelv oktatása C# alapokon," in Informatika a felsőoktatásban 2008 konferencia, Debrecen, 27-29 Aug. 2008, p. 118.

[46] S. Szénási, Z. Vámossy, "Evolutionary algorithm for optimizing parameters of GPGPU based image segmentation," Acta Polytechnica Hungarica, 2013.

[47] S. Szénási, Z. Vámossy, M. Kozlovszky, "Evaluation and comparison of cell nuclei detection algorithms," in IEEE 16th International Conference on Intelligent Engineering Systems (INES), Lissboa, 13-15 June 2012, pp. 469-475.

[65] S. Szénási, Z. Vámossy, M. Kozlovszky, "Preparing initial population of genetic algorithm for region growing parameter optimization," in 4th IEEE International Symposium on Logistics and Industrial Informatics (LINDI), 5-7 Sept. 2012, pp. 47-54.

[86] S. Szénási, Z. Vámossy, "Implementation of distributed genetic algorithm for parameter optimization in cell nuclei detection project," *Acta Polytechnica Hungarica*, 2013.

# FÜGGELÉK 1 – PONTOSSÁGI VIZSGÁLAT RÉSZLETES ADATAI

| Kép | File neve | Szélesség | Magasság | Nem annotált pixelek száma | RG-C idő | RG-G idő | KM idő |
|---|---|---|---|---|---|---|---|
| 1 | B2007_00259_ES_01_validation | 1280 | 1024 | 1148460 | 218360 | 48696 | 25888 |
| 2 | B2007_00259_ES_02_validation | 1024 | 1280 | 1240977 | 149105 | 46250 | 25996 |
| 3 | B2007_00259_ES_03_validation | 1024 | 1024 | 964557 | 80534 | 33386 | 21837 |
| 4 | B2007_00259_PR_01_validation | 1024 | 1024 | 963045 | 150231 | 50647 | 10513 |
| 5 | B2007_00259_PR_02_validation | 1280 | 1024 | 1227946 | 137505 | 40785 | 25999 |
| 6 | B2007_01031_ES_01_validation | 1536 | 1280 | 1735247 | 287405 | 61304 | 42205 |
| 7 | B2007_01031_ES_02_validation | 1792 | 1792 | 3092771 | 478357 | 93773 | 28962 |
| 8 | B2007_01429_PR_02_validation | 1536 | 1536 | 2258348 | 258423 | 66477 | 58446 |
| 9 | B2007_02224_PR_01_validation | 1792 | 1536 | 2500717 | 88894 | 73720 | 59407 |
| 10 | B2007_02224_PR_02_validation | 1792 | 1792 | 3054592 | 134440 | 92421 | 36680 |
| 11 | B2007_02225_ES_01_validation | 1792 | 1792 | 2926504 | 173936 | 116405 | 70963 |
| 12 | B2007_02225_ES_02_validation | 1792 | 1536 | 2528358 | 119015 | 98113 | 60741 |
| 13 | B2007_02225_ES_03_validation | 1536 | 1792 | 2304288 | 143450 | 111043 | 60984 |
| 14 | B2007_02508_PR_01_validation | 1792 | 1792 | 2894079 | 124036 | 101106 | 69292 |
| 15 | B2007_02508_PR_02_validation | 1792 | 1536 | 2657515 | 107322 | 93090 | 59822 |
| 16 | B2007_02819_ES_01_validation | 1280 | 1280 | 1512268 | 40465 | 46374 | 39243 |
| 17 | B2007_02819_ES_02_validation | 1280 | 1536 | 1709903 | 59836 | 61052 | 42288 |
| 18 | B2007_02819_ES_03_validation | 1792 | 1280 | 2224792 | 82972 | 68412 | 24742 |
| 19 | B2007_02819_PR_01_validation | 1280 | 1280 | 1546638 | 34944 | 44942 | 38818 |
| 20 | B2007_02819_PR_02_validation | 1280 | 1280 | 1549452 | 41283 | 54961 | 20488 |
| 21 | B2007_02819_PR_03_validation | 1536 | 1280 | 1808054 | 45302 | 52063 | 41950 |
| 22 | B2007_02856_ES_01_validation | 1280 | 1280 | 1500477 | 34791 | 55477 | 23702 |
| 23 | B2007_02856_ES_02_validation | 1280 | 1024 | 1199165 | 20427 | 35677 | 31364 |
| 24 | B2007_02857_ES_01_validation | 1536 | 1536 | 2054726 | 99293 | 105185 | 46225 |
| 25 | B2007_02857_ES_02_validation | 1280 | 1280 | 1473205 | 51784 | 73143 | 29768 |
| 26 | B2007_02857_ES_03_validation | 1536 | 1280 | 1735043 | 45182 | 63782 | 45360 |
| 27 | B2007_02924_PR_01_validation | 1536 | 1536 | 2096868 | 70998 | 74302 | 51381 |
| 28 | B2007_02924_PR_02_validation | 1536 | 1536 | 2108679 | 83969 | 79731 | 28080 |
| 29 | B2007_03019_PR_01_validation | 1536 | 1536 | 2182888 | 91942 | 75689 | 22765 |
| 30 | B2007_03019_PR_02_validation | 1280 | 1280 | 1539163 | 46784 | 51380 | 34170 |
| 31 | B2007_03019_PR_03_validation | 1536 | 1536 | 2047810 | 88166 | 75568 | 48562 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **32** | B2007_03381_ES_01_validation | 1280 | 1280 | 1462345 | 47001 | 56461 | 40676 |
| **33** | B2007_03381_ES_03_validation | 1536 | 1536 | 2101740 | 80679 | 78844 | 52593 |
| **34** | B2007_03381_PR_01_validation | 1536 | 1536 | 2162112 | 88790 | 88046 | 46098 |
| **35** | B2007_03381_PR_02_validation | 1536 | 1280 | 1693046 | 55064 | 62949 | 44265 |
| **36** | B2007_03381_PR_03_Validation | 1280 | 1280 | 1444294 | 90745 | 95382 | 14956 |

**16. táblázat:** *Képek mérete, feldolgozási idők.*

| Kép | Igaz-pozitív | Igaz-negatív | Hamis-pozitív | Hamis-negatív | Nem annotált | Pontosság |
|---|---|---|---|---|---|---|
| **1** | 0 | 141751 | 20509 | 0 | 1148460 | 87,36% |
| **2** | 10438 | 39196 | 1906,69 | 18613 | 1240977 | 70,75% |
| **3** | 8454 | 56150 | 3938,19 | 16509,59 | 964557 | 75,96% |
| **4** | 2305 | 66690 | 181,92 | 16290,73 | 963045 | 80,73% |
| **5** | 4598 | 64359 | 747,44 | 13285,35 | 1227946 | 83,09% |
| **6** | 519 | 216904 | 1583,26 | 11739,17 | 1735247 | 94,23% |
| **7** | 11629 | 80065 | 2524,07 | 24313,39 | 3092771 | 77,36% |
| **8** | 32362 | 42312 | 16508,48 | 15500,12 | 2258348 | 70,00% |
| **9** | 33557 | 171967 | 17063,25 | 32411,85 | 2500717 | 80,60% |
| **10** | 21759 | 105146 | 9619,43 | 21720,28 | 3054592 | 80,20% |
| **11** | 9173 | 231364 | 13166,8 | 32472,16 | 2926504 | 84,05% |
| **12** | 6218 | 176992 | 4610,77 | 36096,08 | 2528358 | 81,82% |
| **13** | 7107 | 354392 | 21650,13 | 66516,05 | 2304288 | 80,39% |
| **14** | 63934 | 205441 | 28169,39 | 24230,07 | 2894079 | 83,72% |
| **15** | 30250 | 47219 | 6815,78 | 12009,73 | 2657515 | 80,45% |
| **16** | 2687 | 84231 | 260,72 | 39101,95 | 1512268 | 68,83% |
| **17** | 7512 | 189964 | 1608,57 | 56932,68 | 1709903 | 77,13% |
| **18** | 3987 | 40814 | 1987,79 | 23700,72 | 2224792 | 63,56% |
| **19** | 22991 | 49738 | 9232,64 | 12214,05 | 1546638 | 77,23% |
| **20** | 24117 | 47662 | 8974,72 | 9545,18 | 1549452 | 79,49% |
| **21** | 31275 | 91858 | 20074,76 | 19239,12 | 1808054 | 75,80% |
| **22** | 47939 | 52518 | 20920,27 | 22042,43 | 1500477 | 70,04% |
| **23** | 36192 | 50300 | 14324,91 | 13071,94 | 1199165 | 75,94% |
| **24** | 50786 | 229769 | 9509,8 | 13659,93 | 2054726 | 92,37% |
| **25** | 46478 | 90009 | 18552,66 | 10224,2 | 1473205 | 82,59% |

| | | | | | | |
|---|---|---|---|---|---|---|
| **26** | 55444 | 136768 | 19499,41 | 19697,54 | 1735043 | 83,06% |
| **27** | 48270 | 186238 | 15126,49 | 13961,24 | 2096868 | 88,97% |
| **28** | 51820 | 162252 | 19294,32 | 24494,74 | 2108679 | 83,02% |
| **29** | 11401 | 134138 | 2240,9 | 28459,53 | 2182888 | 82,58% |
| **30** | 5385 | 74650 | 1304,97 | 17925,18 | 1539163 | 80,63% |
| **31** | 17159 | 272334 | 4295,41 | 17049,15 | 2047810 | 93,13% |
| **32** | 0 | 143317 | 0 | 32761 | 1462345 | 81,39% |
| **33** | 149 | 194682 | 0 | 62766,11 | 2101740 | 75,63% |
| **34** | 1325 | 127604 | 1202,68 | 68000,46 | 2162112 | 65,07% |
| **35** | 2941 | 194966 | 2444,92 | 73143,01 | 1693046 | 72,36% |
| **36** | 0 | 137755 | 0 | 57136 | 1444294 | 70,68% |

**17. táblázat:** *CPU alapú régiónövelés részletes eredményei.*

| Kép | Igaz-pozitív | Igaz-negatív | Hamis-pozitív | Hamis-negatív | Nem annotált | Pontosság |
|---|---|---|---|---|---|---|
| **1** | 0 | 141155 | 21105 | 0 | 1148460 | 86,99% |
| **2** | 9831 | 39191 | 2083,24 | 19252,28 | 1240977 | 69,68% |
| **3** | 8130 | 56225 | 4236,5 | 16818,78 | 964557 | 75,35% |
| **4** | 2403 | 66614 | 268,92 | 16191,61 | 963045 | 80,74% |
| **5** | 4193 | 64378 | 744,46 | 13706,97 | 1227946 | 82,59% |
| **6** | 762 | 216892 | 1782,07 | 11397,86 | 1735247 | 94,29% |
| **7** | 11708 | 79756 | 2889,09 | 24264,2 | 3092771 | 77,11% |
| **8** | 32170 | 42240 | 17503,38 | 15608,11 | 2258348 | 69,20% |
| **9** | 33863 | 171916 | 16869,32 | 32027,39 | 2500717 | 80,80% |
| **10** | 21742 | 105094 | 9618,96 | 21717,91 | 3054592 | 80,19% |
| **11** | 9227 | 231488 | 13049,55 | 32417,65 | 2926504 | 84,11% |
| **12** | 5280 | 176738 | 4688,69 | 37027,65 | 2528358 | 81,35% |
| **13** | 7355 | 353772 | 22176,62 | 66287,58 | 2304288 | 80,32% |
| **14** | 63867 | 205283 | 28726,93 | 24137,69 | 2894079 | 83,58% |
| **15** | 30295 | 47025 | 7584,2 | 11927,92 | 2657515 | 79,85% |
| **16** | 2659 | 84266 | 229,21 | 39126,99 | 1512268 | 68,83% |
| **17** | 7857 | 190228 | 1715,37 | 56579,33 | 1709903 | 77,26% |
| **18** | 3810 | 40860 | 1883,26 | 23866,57 | 2224792 | 63,43% |
| **19** | 23249 | 49363 | 9869,03 | 11929,06 | 1546638 | 76,91% |

| | | | | | | |
|---|---|---|---|---|---|---|
| **20** | 24403 | 47399 | 9845,37 | 9242,31 | 1549452 | 79,00% |
| **21** | 31105 | 92195 | 20576,08 | 19498,46 | 1808054 | 75,47% |
| **22** | 48032 | 52141 | 21804,26 | 21915,48 | 1500477 | 69,62% |
| **23** | 36341 | 49745 | 15323,49 | 12834,47 | 1199165 | 75,35% |
| **24** | 49921 | 229865 | 10613,23 | 14531,06 | 2054726 | 91,75% |
| **25** | 46311 | 89489 | 19312,5 | 10475,29 | 1473205 | 82,01% |
| **26** | 55943 | 136253 | 20124,61 | 19211,24 | 1735043 | 83,01% |
| **27** | 48679 | 185669 | 16781,13 | 13536,48 | 2096868 | 88,54% |
| **28** | 51837 | 162238 | 20474,92 | 24453,55 | 2108679 | 82,65% |
| **29** | 11498 | 134070 | 2631,81 | 28336,61 | 2182888 | 82,46% |
| **30** | 5471 | 74639 | 1304,19 | 17823,17 | 1539163 | 80,73% |
| **31** | 16823 | 272480 | 4151,75 | 17391,51 | 2047810 | 93,07% |
| **32** | 0 | 143317 | 0 | 32761 | 1462345 | 81,39% |
| **33** | 143 | 194682 | 0 | 62773,39 | 2101740 | 75,63% |
| **34** | 1323 | 127624 | 1178,79 | 67967,38 | 2162112 | 65,09% |
| **35** | 2927 | 195116 | 2256,64 | 73179,09 | 1693046 | 72,42% |
| **36** | 0 | 137755 | 0 | 57136 | 1444294 | 70,68% |

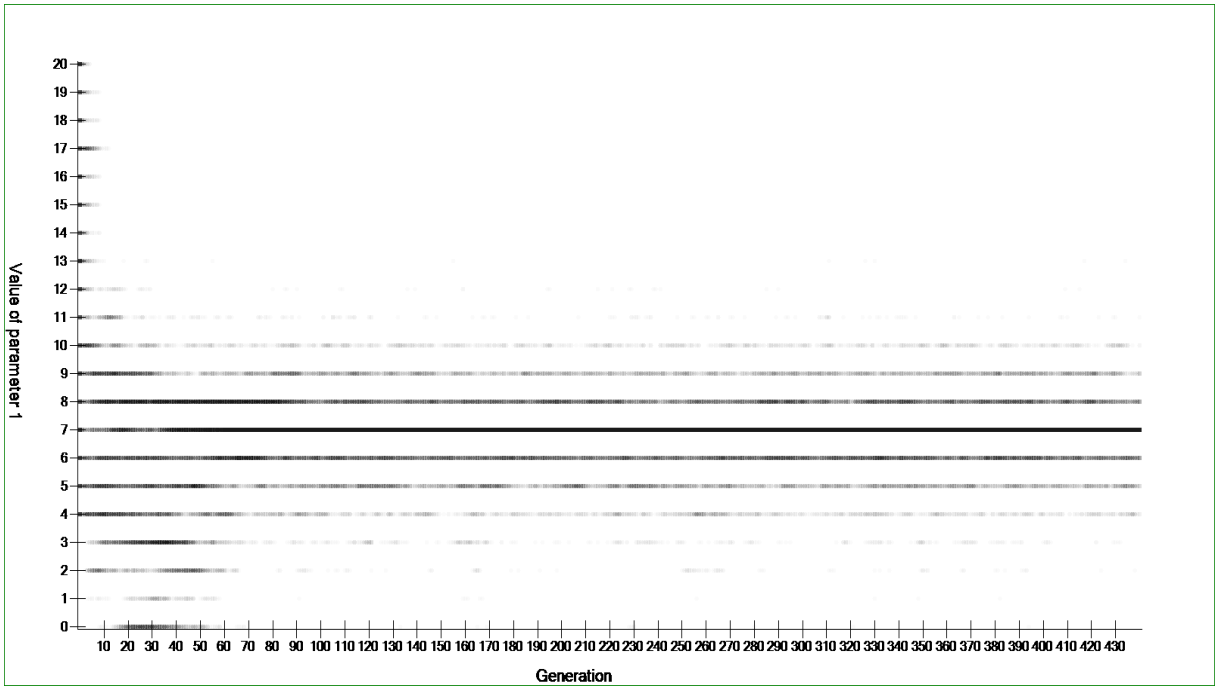
**18. táblázat:** *GPGPU alapú régiónövelés részletes eredményei.*

| Kép | Igaz-pozitív | Igaz-negatív | Hamis-pozitív | Hamis-negatív | Nem annotált | Pontosság |
|---|---|---|---|---|---|---|
| **1** | 0 | 96336 | 65924 | 0 | 1148460 | 59,37% |
| **2** | 18722 | 32341 | 13266,35 | 9290,7 | 1240977 | 69,36% |
| **3** | 16807 | 41390 | 22044,28 | 7420,26 | 964557 | 66,39% |
| **4** | 11100 | 64022 | 5037,16 | 6556,42 | 963045 | 86,63% |
| **5** | 13171 | 45097 | 22573,2 | 4272,02 | 1227946 | 68,46% |
| **6** | 6418 | 110495 | 113011,87 | 5016,06 | 1735247 | 49,76% |
| **7** | 16725 | 75883 | 8686,78 | 18139,48 | 3092771 | 77,54% |
| **8** | 32381 | 42999 | 18286,04 | 14422,83 | 2258348 | 69,74% |
| **9** | 39397 | 132480 | 69398,71 | 24568,68 | 2500717 | 64,65% |
| **10** | 18754 | 106362 | 14558,64 | 23093,64 | 3054592 | 76,87% |
| **11** | 24859 | 142483 | 112314,05 | 15322,84 | 2926504 | 56,73% |
| **12** | 22521 | 108755 | 87675,65 | 17486,59 | 2528358 | 55,52% |
| **13** | 37726 | 216423 | 184453,72 | 32008,85 | 2304288 | 54,00% |

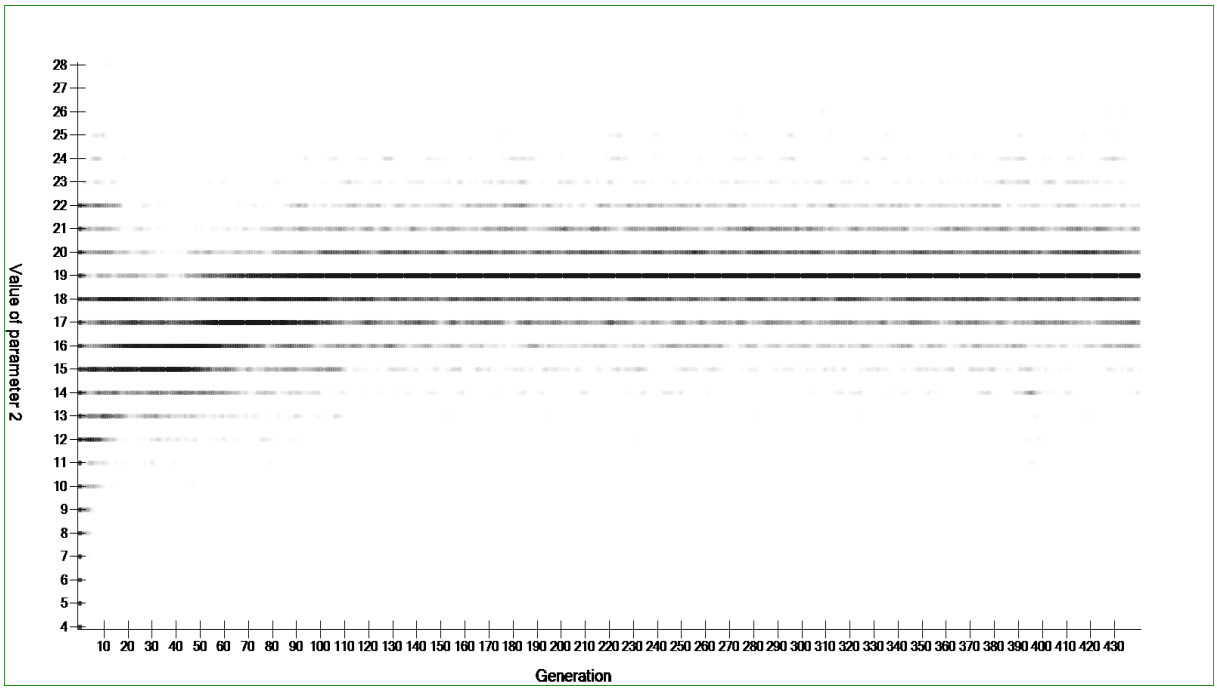
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **14** | 55765 | 163819 | 82884,49 | 30101,28 | 2894079 | 66,03% |
| **15** | 25167 | 45157 | 16036,03 | 15882,19 | 2657515 | 68,78% |
| **16** | 28164 | 60581 | 31941,99 | 12153,2 | 1512268 | 66,81% |
| **17** | 42283 | 131364 | 74261,17 | 19663,79 | 1709903 | 64,90% |
| **18** | 10961 | 38877 | 6897,36 | 15497,79 | 2224792 | 69,00% |
| **19** | 22112 | 46210 | 18070,29 | 12045,66 | 1546638 | 69,41% |
| **20** | 14485 | 51006 | 10410,18 | 18012,06 | 1549452 | 69,74% |
| **21** | 30780 | 77839 | 42092,56 | 18390,65 | 1808054 | 64,23% |
| **22** | 37313 | 62865 | 9642,13 | 30932,82 | 1500477 | 71,17% |
| **23** | 33843 | 47999 | 21851,23 | 14560,08 | 1199165 | 69,21% |
| **24** | 45432 | 178588 | 73917,55 | 17701,25 | 2054726 | 70,97% |
| **25** | 33040 | 100832 | 12785,64 | 21205,97 | 1473205 | 79,75% |
| **26** | 52584 | 119864 | 47283,53 | 20671,65 | 1735043 | 71,73% |
| **27** | 36598 | 142916 | 74788,29 | 23737,17 | 2096868 | 64,56% |
| **28** | 23798 | 160111 | 36542,04 | 49622,11 | 2108679 | 68,10% |
| **29** | 12467 | 120051 | 24435,17 | 25839,6 | 2182888 | 72,50% |
| **30** | 9985 | 48200 | 36308,53 | 12041 | 1539163 | 54,62% |
| **31** | 19213 | 163594 | 124499,16 | 13880,51 | 2047810 | 56,92% |
| **32** | 16196 | 80662 | 75168,42 | 14202,43 | 1462345 | 52,01% |
| **33** | 24827 | 108943 | 112234,76 | 33107,47 | 2101740 | 47,93% |
| **34** | 21135 | 84018 | 77515,2 | 42656,23 | 2162112 | 46,67% |
| **35** | 37353 | 121423 | 104038,51 | 33513,34 | 1693046 | 53,58% |
| **36** | 7996 | 122068 | 26233,84 | 45624,16 | 1444294 | 64,41% |

**19. táblázat:** *K-közép módszeren alapú régiónövelés részletes eredményei.*

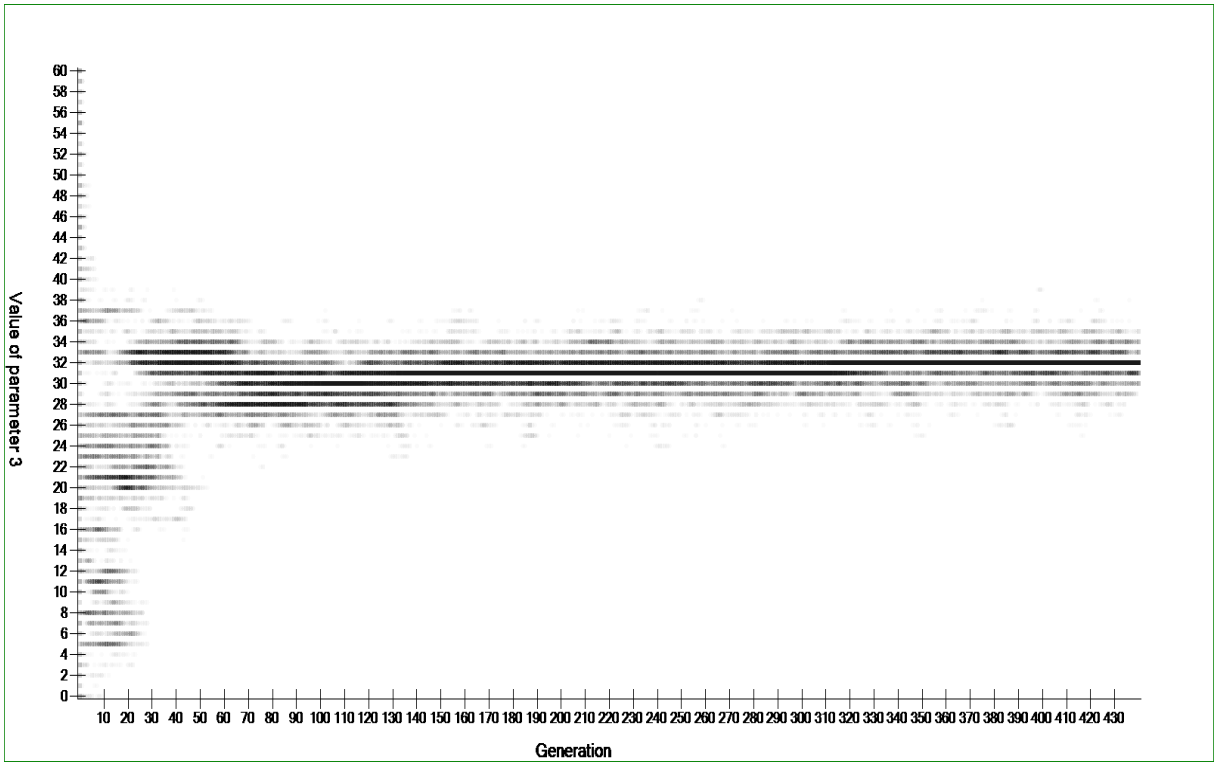
# FÜGGELÉK 2 – PARAMÉTEREK ÉRTÉKEINEK GENERÁCIÓNKÉNTI VÁLTOZÁSA



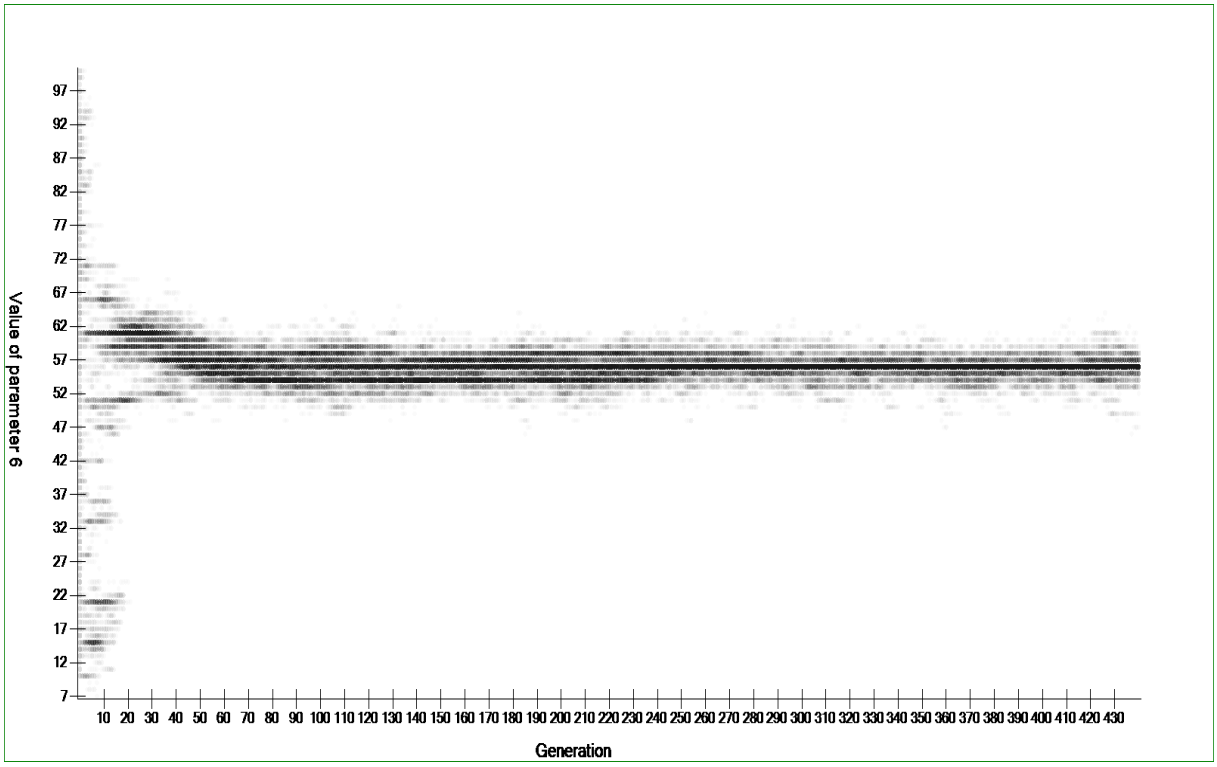

**16. diagram:** *1. paraméter értékei generációnként.*



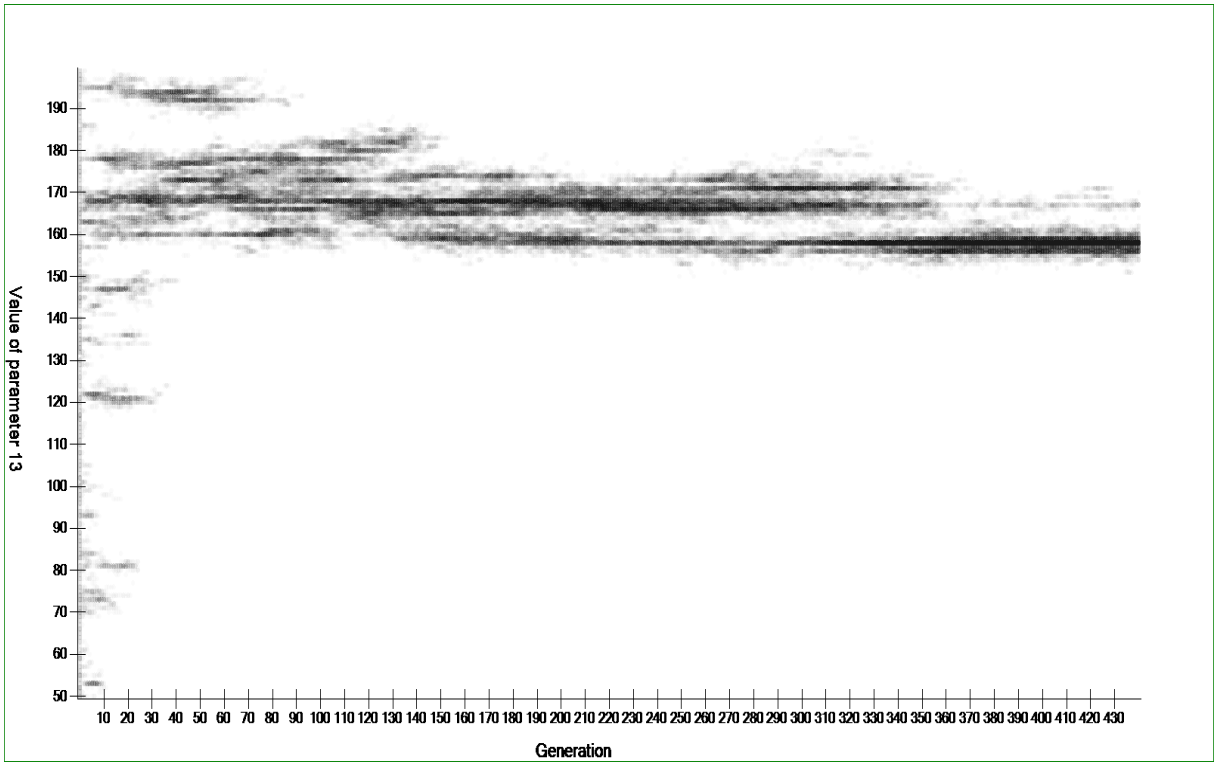

**17. diagram:** *2. paraméter értékei generációnként.*



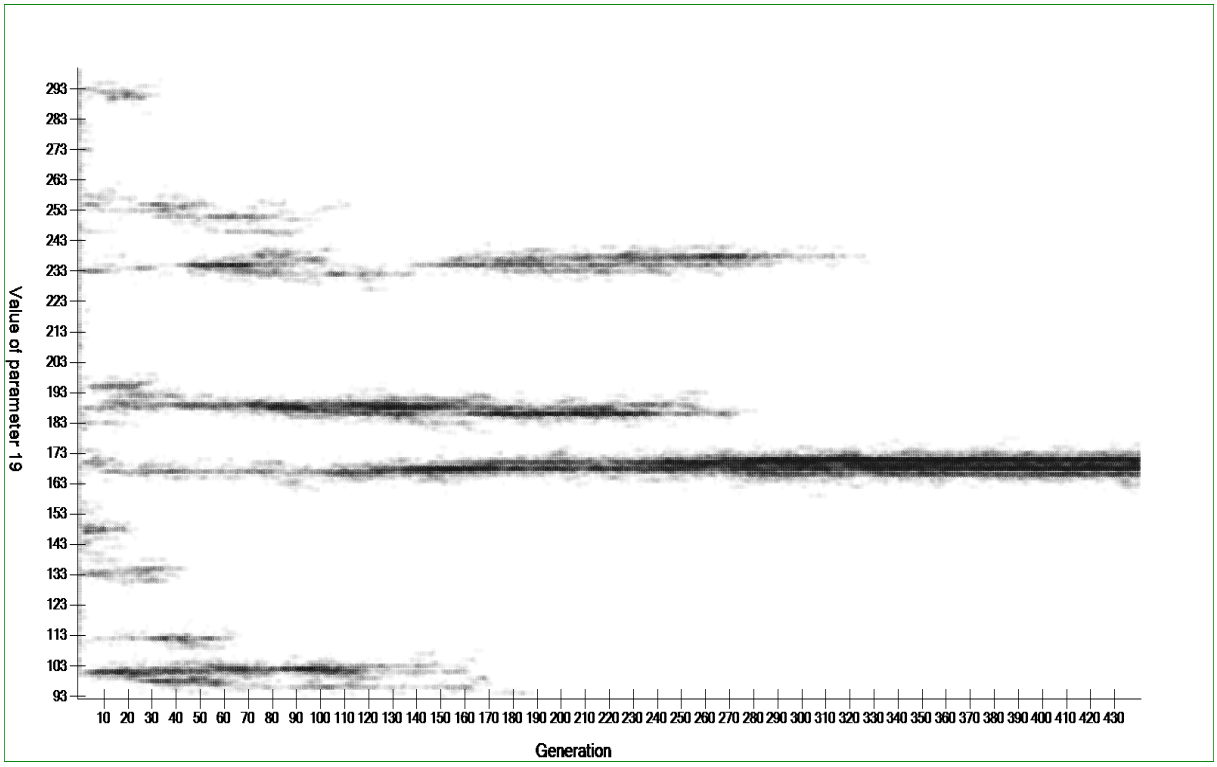

**18. diagram:** *3. paraméter értékei generációnként.*



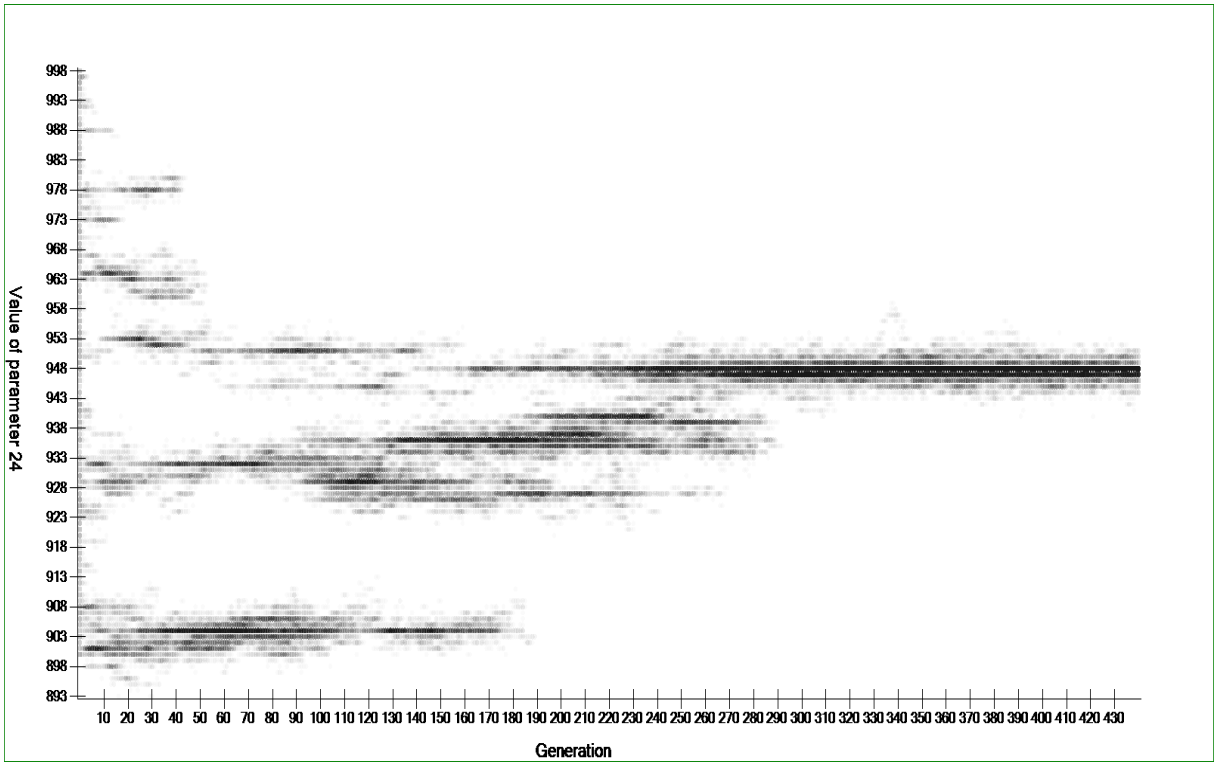

**19. diagram:** *6. paraméter értékei generációnként.*



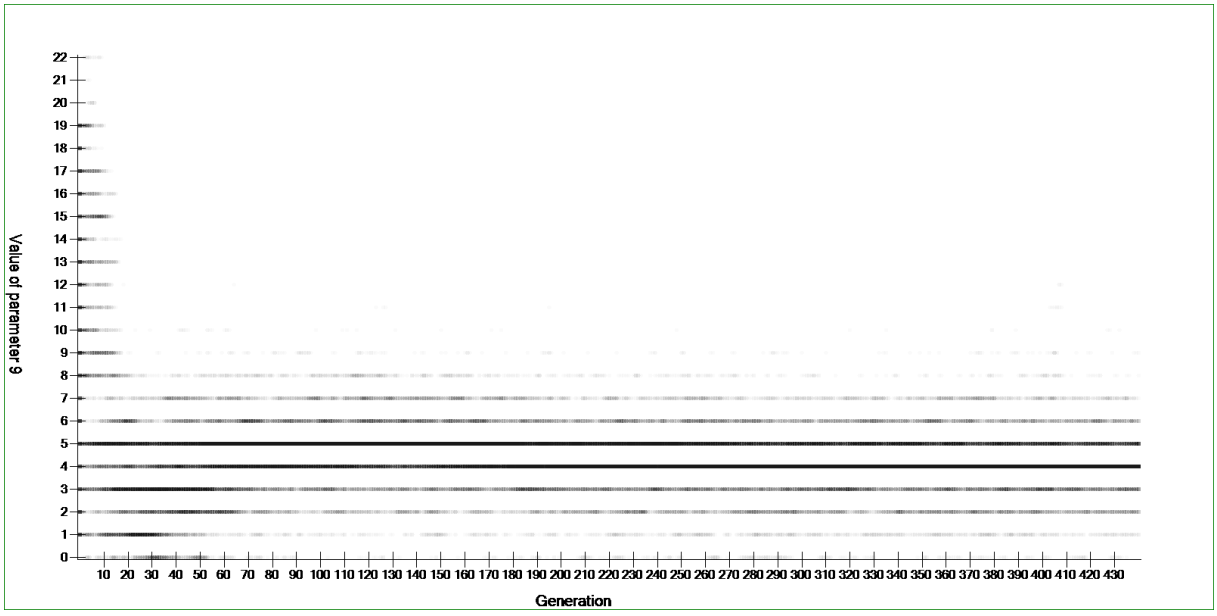

**20. diagram:** *13. paraméter értékei generációnként.*



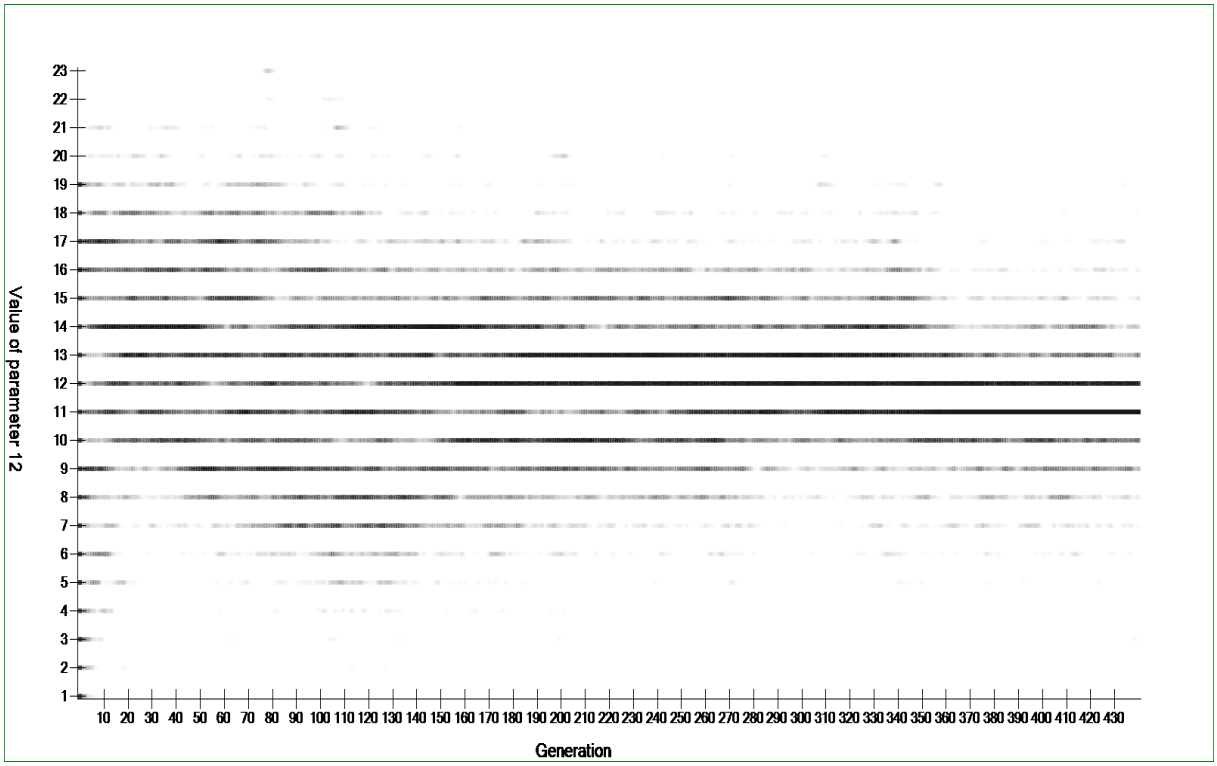

**21. diagram:** *19. paraméter értékei generációnként.*




**22. diagram:** *24. paraméter értékei generációnként.*




**23. diagram:** *9. paraméter értékei generációnként.*




**24. diagram:** *12. paraméter értékei generációnként.*