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

Nagy Balázs, Tóth Ádám Aladin

PÁROSÍTÁSI ALGORITMUSOK AZ ALÁÍRÁS-HITELESÍTÉSBEN

KONZULENS
DR. KŐVÁRI BENCE

BUDAPEST, 2016

Tartalomjegyzék

Összefoglaló

Az aláírás-hitelesítés a legősibb biometrikus azonosítási módszerek egyike. Az utóbbi évtizedekben számos kísérlet történt az folyamat teljes automatizálására. A legszélesebb körben felhasználható (off-line) megoldások pusztán az aláírások szkennelt képeiből indulnak ki, ám a korlátozott információtartalom miatt még sok teret hagynak a fejlődésnek. Dolgozatunkban ezen belül egy olyan megközelítést választottunk, mely a törvényszéki írásszakértők vizsgálati módszereit modellezi. Míg azonban az aláírásokból kinyert jellemzők összerendelése egy ember számára egyszerűbb feladat, egy gép számára komoly kihívást jelent. Dolgozatunkban olyan párosítási algoritmusokat valósítottunk és vizsgáltunk meg, melyek megoldást nyújthatnak a felmerülő problémákra. Elemezzük és értelmezzük az egyes algoritmusok karakterisztikáját és következtetéseket vonunk le a továbbfejlesztési irányokról. A mások által megvalósított rendszerek mellett 2 teljesen saját megközelítést is bemutatunk melyek a jövőben jelentősen hozzájárulhatnak a jellemzőpárosítás javításához az aláírás hitelesítés területén.

Abstract

Signature verification is among the oldest biometric identification methods. There has been a large number of attempts in the last few decades to fully automate the verification process. The off-line approach, which promises the widest field of applicability makes its decision solely based on the scanned images of the signatures. Because of the limited information content there is still much room for improvement in this field. Our solution mimics the approach of forensic handwriting experts, by identifying, matching and comparing smaller features of the signatures. Although the matching of features is an easy task for a human it is much harder for a computer to tackle. In our work we have realized and examined several matching algorithms that may provide solutions for this problem. We analyze the different approaches and provide conclusions about future development directions. Beside the works of others, we have created 2 new algorithms that may contribute to the constant improvement of feature matching in the field of signature verification.

1 Bevezetés

Az aláírás az egyik legrégebbi biometrikus azonosítási módszer. Évszázadok óta hétköznapi életünk szerves részét képezi, gondoljunk csak arra, hogy egy banki ügyintézésnél, postai küldemény átvételénél, egy igazolás kiállításánál vagy akár csak egy kérdőív kitöltésénél is az aláírásunkkal jelezzük beleegyezésünket vagy támogatásunkat. Pontosan az ilyen esetek miatt kulcsfontosságú, hogy az eredetiség – tehát hogy az aláírás valóban az adott személytől származzon – helyesen megállapítható legyen.

Természetesen nem elvárható, hogy minden egyes esetben rendelkezésre álljon egy szakértő, aki képes egy-egy aláírásról eldönteni, hogy eredeti vagy hamisítvány, éppen ezért fontos kutatási terület a verifikáció folyamatának automatizálása, amelynek segítségével sokkal nagyobb mennyiségben tudunk adatot feldolgozni, töredékére csökkentve ezzel a hitelesítés idejét.

Ugyanakkor fontos megemlíteni, hogy néha még a gyakorlott íráselemző szakemberek is tévednek, az ő hibaarányuk is átlagosan 0,5 és 7% közé tehető [1]. Ezek a számok remekül szemléltetik, hogy rengeteg nyitott kérdés van a szakterületen, ami megoldásra vár.

1.1 Aláírás-hitelesítés a gyakorlatban

Az automatizált aláírás-hitelesítés célja, hogy egy aláírásról minél nagyobb bizonyossággal és korlátos időn belül, programozottan eldöntsük, hogy az valóban az állítólagos személyhez tartozik. A döntéshez azonban szükségünk van néhány korábbi aláírás mintára, amiről biztosan tudjuk, hogy eredetiek. A minták típusa alapján kettő nagy kategóriára oszthatjuk a tudományterületet: on-line (dinamikus) és off-line (statikus) megközelítés.

Amíg az on-line megközelítéshez szükség van valami speciális segédeszközre (pl. kamera, digitalizáló tábla, stylus), hogy több információhoz jutva pontosabb képet kapjunk az aláírásról, addig az off-line esetben elegendő valamilyen szkennelt kép, ezért ez utóbbi esetben nincsen szükség komoly beruházásokra, alkalmazása sokkal költséghatékonyabb.

Tekintve, hogy az off-line rendszerek csak kétdimenziós képek alapján dolgoznak, nem érhetők el az írás módjára vonatkozó információk, az aláírás dinamikáját, sebességére és erősségére vonatkozó információkat csupán algoritmikus módon próbálhatjuk kinyerni.

A felsorolt hátrányok ellenére azonban az off-line megközelítés gyakorlati alkalmazhatósága mégis jónak mondható: remekül illeszkedik már létező munkafolyamatokba pl. sok pénzügyi intézmény digitalizált dokumentumain használható, mindösszesen szoftveres fejlesztésre lenne szükség.

1.2 Jellemzőpárosítás

A jellemző alapú hitelesítési eljárások egyik legfontosabb eleme a jellemzők megfelelő párosítása: ez a lépés szükséges ahhoz, hogy a további feldolgozó és mérlegelő egységek az aláírások felépítését helyesen kapják meg, egymással összerendelhető jellemzőket hasonlítsanak össze és helyes döntést hozzanak. Az itt alkalmazott algoritmusnak van ráadásul utoljára esélye arra, hogy elfedje a képfeldolgozásból származó hibákat, és olyan adathalmazt hozzon létre, mely a korlátos információtartalom ellenére is a lehető legjobb eredmények előállítását teszi lehetővé (*1. ábra*).

1. ábra: A jellemző alapú hitelesítés lépései [2]

A Budapesti Műszaki és Gazdaságtudományi Egyetem Automatizálási és Alkalmazott Informatikai Tanszékén fejlesztett moduláris felépítésű Aláírás Hitelesítő Rendszer (AHR) egy ezen a folyamaton alapuló aláírás-hitelesítést valósít meg. A rendszer moduláris felépítésű, így könnyedén cserélhetők benne az egyes lépéseiket megvalósító modulok. Ennek köszönhetően könnyedén tudunk a folyamat harmadik részével, a jellemzőpárosítással foglalkozni úgy, hogy a többi résszel csak előre definiált interfészen keresztül érintkezünk.

Ennek megfelelően a párosítás kezdetén a már beolvasott képből megállapításra kerültek a különböző jellemzők és hasonlósági értékeik, bemenetül szolgálva egy párosító algoritmusnak, amely eredménye a hitelesség megállapítását fogja segíteni.

Jellemző alatt az aláírás egy olyan részét értjük, amit mérhető tulajdonságokkal definiálhatunk. Kettő nagy típusát különböztetjük meg: lokális (helyi) és globális jellemző. Előbbire példa lehet a dolgozatunkban vizsgált alapvonalak vagy hurkok, míg utóbbi legegyszerűbb példája az aláírás szélessége vagy magassága.

A jellemző alapú módszerek kettő bemeneti halmazt feltételeznek, a cél a kettő halmaz (referencia és vizsgált aláírás) közti hasonlóságok megtalálása legyen szó akár alapvonalakról, hurkokról, görbékről, metszéspontokról stb.

Ahhoz, hogy ezt elérjük, egy úgynevezett pont-pont párosítást használunk. Az aláírásokat párba állítva hasonlítjuk össze, kettő fázisban: az első, tanulási fázisban az eredeti aláírások között keresünk leképzést, majd a teszt fázisban pedig a bemenetül kapott minta aláírást jellemzőit próbáljunk párosítani az előzőekhez. Ez a többdimenziós párosítási probléma NP-nehéz [3], ezért speciális heurisztikákkal közelítettük az optimális eredményt.

1.3 Dolgozat felépítésének ismertetése

A dolgozat további részében az elméleti hátteret, követelményeket és kihívásokat mutatjuk be. Ezután rátérünk az eddig kidolgozott megoldásokra és az általunk implementált kettő új algoritmusra. Működésüket mérési eredményekkel szemléltetjük és hasonlítjuk össze, majd a végén összefoglaljuk az elvégzett munkát és egy kitekintést adunk a jövőre nézve.

2 Irodalmi áttekintés

Szerencsére a párosítási algoritmusoknak és a velük ekvivalens problémáknak legalább akkora irodalma van, mint amekkora a jelentőségük az offline aláírás-hitelesítés területén. A jellemzőket egy gráf csúcsaival és a közülük kiválasztott párokat a csúcsok között húzott élekkel modellezve a jellemzőpárosítás könnyen megfeleltethető egy többdimenziós, súlyozott párosításnak, de sok más algoritmikus megoldást is alkalmazhatunk a megoldása során. Most azonban a probléma konkrét megoldása nélkül ─ hisz arra több különböző példát fogunk látni a Párosítási algoritmusok című fejezetben ─ részletezni szeretnénk azokat az ismert algoritmusokat, melyeket dolgozatunk további részében használni fogunk, még ha nagy részüket átalakítva is, hogy a pontos feladatra megoldást adhasson.

2.1 Algoritmikus probléma

A jellemzőpárosítás megfeleltethető a gráfelméletben ismert n-dimenziós párosításnak, de akár a set packing problémán keresztül is közelíthetjük a megoldását: ez az átírás pedig megengedi azt, hogy teljesen új szemszögből beszélhessünk róla. Az előbbi problémáról például tudjuk, hogy NP-nehéz [3], azaz nem tudjuk polinomiális időben megoldani. Ez azt jelenti, hogy a jellemzőpárosításhoz nem tudunk hatékony teljes megoldást találni, és hogyha gyakorlatban használható idejű algoritmust akarunk készíteni, mindenképpen veszíteni fogunk a pontosságából.

A NP-nehézség egy speciális esete az NP-teljesség, ami magában foglalja azt is, hogy minden NP-nehéz probléma visszavezethető rá. Ilyen például a set packing [4], aminek ezt a tulajdonságát több megoldásban is kihasználtuk, visszavezetve rá az n-dimenziós párosítást.

2.2 Párosítási problémák

2.2.1 Párosítások kétdimenziós páros gráfokon

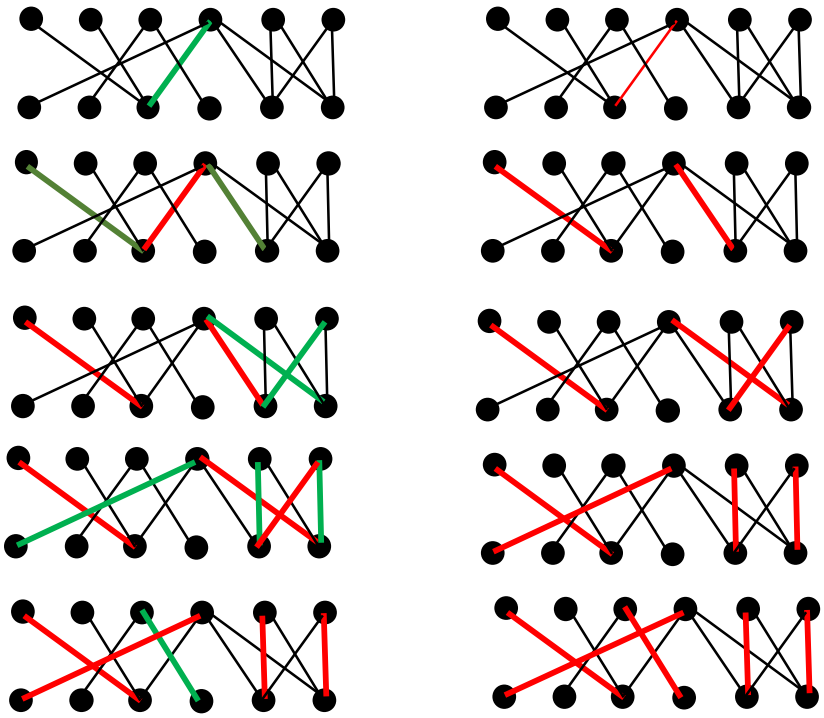
Definíció: *Egy* $G(V, E)$ *gráfban párosításnak nevezünk egy olyan* $M \subseteq E$ *élhalmazt, mely páronként nem szomszédos éleket tartalmaz, azaz semelyik két élnek nincsen közös végpontja.*

Definíció: *Egy **G(V, E)** gráfot páros gráfnak nevezünk, hogyha **V** elemei felosztathatók egy **A** és **B** halmazra úgy, hogy minden **E**-beli él egyik végpontja az **A** halmazból, másik végpontja pedig a **B** halmazból kerül ki.*

Egy párosítás maximális (vagy maximális elemszámú), hogyha a gráfban nincsen nála nagyobb elemszámú párosítás. A páros gráfokban történő maximális párosítás keresést hívjuk két dimenziós párosításnak. [5] [6]

2.2.1.1 Súlyozatlan páros gráfokon

A súlyozatlan páros gráfokon történő párosítás egy régóta ismert probléma, melyre az ismert javító utas algoritmus tud megoldást adni.



2. ábra: Két dimenziós párosítás javító utak módszerével

Ez az algoritmus polinomiális komplexitású, egyik legegyszerűbb megvalósítása, mely a Ford-Fulkerson algoritmuson alapul, ***O(V*E)*** futási idejű, de mátrixszorzáson alapuló megoldással például $O(V^{2,376})$ nagyságrendet érhetünk el [7].

2.2.1.2 Súlyozott páros gráfokon

Súlyozott gráfokban minden élhez rendelve van egy számérték, amit az él súlyának nevezünk. Egy ilyen gráfban maximális súlyú párosításnak nevezzük azt a párosítást, ahol a párosításban szereplő élekhez tartozó súlyok összege maximális.

A két dimenziós párosítás súlyozott esetének megoldásával először Harold Kuhn dolgozta ki Kőnig Dénes és Egerváry Jenő munkája alapján [8], ezért ez az eljárás azóta is magyar módszer néven ismert az irodalomban. A feladatot magát másképp hozzárendelési problémának is nevezik.

A magyar módszer a súlyozatlan problémához hasonlóan polinomiális idejű algoritmus: a Bellman-Ford algoritmusra épülő $O(V^3 * E)$ komplexitású változata könnyen megvalósítható, de $O(V^2 * logV + V * E)$ implementáció is ismert, mely Dijkstra algoritmusával és Fibonacci halmok használatával éri el ezt az eredményt [9].

2.2.2 Többdimenziós párosítás

A több dimenziós párosítások egyik fontos fogalma a hiperél: ezek olyan élek, melyek több mint két csúcsot kötnek össze. A segítségükkel könnyen leírhatjuk az n-dimenziós párosítást, melynek lényege, hogy n hosszú hiperélek legnagyobb elemszámú (súlyozott esetben legnagyobb súlyú) halmazát keressük egy hipergráfban, ahol a hiperélek páronként nem metszik egymást [10].

A páros gráfoknak is van általános, több-dimenziós megadása, a k-partite gráf, melyet úgy definiálhatunk, hogy k csúcsosztállyal rendelkeznek, melyeken belül nem vezethet él, csupán közöttük.

Sajnos a párosítás több dimenzióra való kiterjesztése nem polinomiális komplexitású algoritmus [3]. Már a legkisebb, 3-dimenziós párosítás is NP-nehéz algoritmus, így teljes megoldása olyan időigényű, ami szinte biztos nem engedhető meg az off-line aláírás hitelesítés során.

2.2.3 A jellemzőpárosítás átírása többdimenziós párosítási problémává

Az általunk ismertetett algoritmusok az aláírások jellemzőit használva egy k-dimenziós páros gráfot építenek fel a jellemzőpárosítás megoldásához, melyben:

- az egyes jellemzők a gráf csúcsaiként jelennek meg
- az élek a különböző aláírások jellemzői közti hasonlóságot jelentik, ami egyben azt is biztosítja, hogy egy aláíráson belül függetlenek a pontok. Az élek súlya mutatja, hogy milyen jó a párosítás, a jellemzők hasonlósága alapján.

Ennek megfelelően a probléma tényleg egy többdimenziós párosításként írható le, ahol a végeredmény hiperélek egy halmaza lesz, ami az optimális párosítást adja meg. Mi nekünk azonban még további fogalmakkal is meg kell ismerkednünk. Ez egyrészről a többdimenziós párosítás NP-nehéz volta miatt van, másrészről pedig mivel az nem elég általános: nem tudja kezelni például azt a problémát, hogyha egy aláírásban nem találunk egy jellemzőhöz párt (mivel például abban az aláírásban a képfeldolgozás során egy alapvonalat érzékeltünk csak a kettő helyett), ezért nem tudunk ***k*** hosszú hiperélt létrehozni.

2.3 Set packing

A set packing probléma az n-dimenziós párosítás egy még általánosabb megadása: a megoldandó feladat lényegében az, hogy ha van egy ***S*** alaphalmaz, és annak néhány megadott részhalmaza, ki lehet-e azok közül választani ***k*** darabot úgy, hogy azok páronként ne mentszék egymást.

A problémának természetesen létezik optimalizálási változata is, mely esetén minél több egymást nem metsző halmaz kiválasztása a cél, és súlyozott halmazokon is értelmezhető: ekkor természetesen a részhalmazok legnagyobb összsúlyú csoportjának megtalálása a cél.

Könnyű látni, hogy a set packing probléma valóban általánosítása a többdimenziós párosításnak, hisz a hiperéleket könnyen megfeleltethetjük az algoritmusban használt részhalmazoknak. Ugyanakkor ezt használva leírhatunk egyéb eseteket is: a legfontosabb, hogy a részhalmazok nem mindig ugyanakkora elemszámúak, míg a hiperélek igen (ez pedig fontos tulajdonság a jellemzőpárosításban).

A set packing probléma egyike a legkorábban ismert NP-teljes problémáknak [4].

2.4 Független csúcshalmaz

Egy ***G*** gráfon vett független csúcshalmaz a csúcsok egy olyan S halmaza, melyben egyik két csúcshoz se tartozik köztük futó él. A maximális független csúcshalmaz az a független csúcshalmaz, melynél nagyobb elemszámút már nem lehetne a ***G*** gráfban találni. Az ehhez tartozó optimalizálási probléma, amely arra a

kérdésre keresi a választ, hogy mi a gráfban található legnagyobb független részhalmaz, egy NP-nehéz optimalizációs feladat [11].

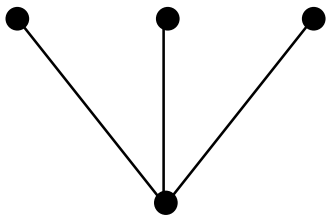
2.4.1 A set packing és a maximális független csúcshalmaz kapcsolata

A set packing és a maximális független csúcshalmaz probléma könnyen találhatunk egy polinomális idejű egyértemű megfeleltetést (kétirányú PTAS redukciót) [12]:

- Egy S halmazon értelmezett set packing problémához generálhatunk egy gráfot úgy, hogy minden $U \in S$ részhalmazhoz létrehozunk egy $v_U \in V$ csúcsot, és v_U akkor van összekötve v_T-vel, hogyha $U \cap T \neq \emptyset$.
- Egy $G(V, E)$ gráfon értelmezett maximális független csúcshalmaz problémához generálhatunk egy S halmazt, melynek elemei a gráf élei, és minden v csúcshoz tartozik egy S_v részhalmaz, mely a csúcshoz kapcsolódó éleket tartalmazza.

2.4.2 Maximális független csúcshalmaz karom-mentes gráfokon

Bár a maximális súlyú független csúcshalmaz keresése NP-nehéz probléma, speciális gráfokon polinomiális idejű algoritmusok is léteznek a megoldására. A P_5-mentes gráfok (P_5-free graph) [13], tökéletes gráfok (perfect graph) [14], és karom-mentes gráfokon (claw-free graph) [15] ilyen esetek, míg húrgráfokon (chordal graph) lineáris idejű megoldás is ismert [16]. Ezek tanulmányozásával azonban arra jutottunk, hogy a jellemzőpárosításhoz a legnagyobb segítséget a karom-mentes gráfok jelentik, mivel ez a szerkezet feleltethető meg legkönnyebben a mi problémánknak.



3. ábra: Karom struktúra egy gráfban

Definíció: *Karom-mentesnek nevezünk egy gráfot, hogyha abban semelyik három csúcs sem független egymástól, ha van közös szomszédjuk (3. ábra).*

A karom-mentes gráfok esetének megoldásával először Minty [15] és Shibi [17] foglalkozott, és ők készítették el rá az első polinomiális algoritmust. Alapja Edmonds

algoritmusa volt [18], mely általános gráfokon találja meg a maximális párosítást javító utakat keresve, polinomiális időben: ezt általánosítva sikerült egy minden karom-mentes gráfra jó algoritmust létrehozni. Az 1980-as publikálás után 21 évvel később Nakamura és Tamura [19] készített hozzá egy javítást, amivel már teljessé vált az algoritmus.

Minty algoritmusa a karom-mentes gráfok tulajdonságára épít: az, hogy semelyik csúcs szomszédjainak halmaza nem tartalmazhat olyan független csúcshalmazt, aminek mérete három vagy annál nagyobb, biztosítja, hogy két független csúcshalmaz szimmetrikus differenciája olyan részgráfot alkot, amiben minden csúcs fokszáma legfeljebb kettő (azaz a részgráf egymással nem összefüggő körökből és utakból áll).

Legyen $G(V, E)$ egy karom-mentes gráf, amihez a $w: V \rightarrow R$ súlyfüggvényt definiáltunk, és rendelkezik egy $S \subset G$ független csúcshalmazzal. Ekkor Minty algoritmusa szerint az S elemeit fekete, a G többi elemét pedig fehér csúcsnak nevezhetjük. A gráf karom-mentes tulajdonsága miatt egy fehér csúcsnak legfeljebb két fekete szomszédja lehet: a csúcsot pedig kötöttnek nevezzük, ha ez teljesül, szabadnak, hogyha csak egy fekete szomszédja van, és szuper szabadnak, ha egy se. Egy utat a gráfban alternáló útnak nevezünk, hogyha abban a fekete és fehér csúcsok váltakozva szerepelnek, és nem tartalmaz két fehér csúcsot, amik szomszédosak egymással. Egy alternáló út lehet fehér (fekete), hogyha mindkét végpontja fehér (fekete), egyébként pedig fehér-fekete. Egy P alternáló út súlya egyenlő a benne lévő fehér csúcsok összegével, kivonva belőle a fekete csúcsok összegét (ezt jelöljük $W(P)$-vel): hogyha ez a súly nagyobb, mint 0, és az út végpontjai nem kötöttek, akkor az utat javító útnak nevezzük.

Ezek ismeretében Minty algoritmusa a következőképp írható le:

1. $S \leftarrow \emptyset$;
*2. **Ha** nincs fehér javító út S-ben, **akkor visszatérés** S-sel; **Egyébként** keressük meg a legnagyobb súlyú P* fehér javító utat;*
*3. $S \leftarrow S \Delta P^*$; **vissza** a 2. lépésre*

Ahol Δ a szimmetrikus differenciát jelöli. Az algoritmus helyességét biztosítja a tény, hogy $S \Delta P^*$ független csúcshalmaz lesz, és $w(S \Delta P) = w(S) + W(P^*)$, valamint a következő lemma:

Lemma: *Ha* ***S*** *egy szemi-optimális, de nem maximális súlyú független csúcshalmaz, akkor a maximális súlyú* ***P**** *fehér javító útjával* ***S*** Δ ***P**** *szintén egy szemi-optimális megoldás, melynek számossága* $|S| + 1$.

Minty algoritmusa nem a legújabb, ami a karom-mentes gráfok maximális súlyú független csúcshalmazának megkeresésével foglalkozik: az elmúlt években sok gyorsabb megoldás született a problémára. Ilyen például a 2011-ben publikált $O(n^3)$ komplexitású megoldás [20], vagy a legújabb, 2015-ös $O(n^2 * \log n)$ idejű algoritmus [21], melyek alkalmazása mind tovább gyorsíthatja az ezekre a módszerekre épülő párosítási algoritmusokat.

2.5 GRASP

A GRASP (*Greedy Randomized Adaptive Search Procedure*) alapgondolata, hogy felhasználunk egy véletlenszerűsített mohó, valamint utána egy lokális keresést is. Működését az alábbi rövid pszeudó kód szemlélteti:

```
1. Megoldások ← ∅;
2. ismétlés
3.    kezdetiMegoldás ← véletlenMohó(problémahalmaz, alfa)
4.    javítottMegoldás ← lokálisKeresés(kezdetiMegoldás)
5.    [javítottMegoldás ←intens.(javítottMegoldás)]
6.    Megoldások ← Megoldások ∪ {javítottMegoldás}
7. amíg leállási feltétel
8. [Megoldások ← posztOptimalizálás(Megoldások)]
9. VégsőMegoldás ← legjobb(Megoldások)
```

Rövid magyarázat a fentiekhez: a leállási feltételig folyamatosan keresünk új megoldásokat, úgy hogy először a véletlenszerűsített mohó algoritmust meghívva egy lehetséges megoldást kapunk. Kettő bemeneti paramétere a problémahalmaz és egy alfa 0 és 1 közti szám, amit mohósági faktornak is szoktunk hívni. Ez utóbbival állíthatjuk be, hogy mennyire legyen az algoritmusunk mohó: 1 esetén teljesen mohó, 0 esetén tiszta véletlen a kimenet.

```
4.1 procedúra lokálisKeresés(S)
4.2   amíg S nem lokálisan optimális ismétlés
4.3         Keressük S' ∈ N(S) amire f(S') < f(S)
4.4         S <- S'
4.5   visszatérés S-sel
4.6 vége
```

A kapott eredményt tovább finomítjuk egy lokális keresést hívva: itt megint csak tervezői döntés, hogy mekkora helyi környezetet (mennyi szomszédot) látogatva próbálkozzunk egy kedvezőbb megoldást találni. A pszeudó kódjában látható, hogy szükségünk van egy költségfüggvényre, ami alapján el tudjuk dönteni, hogy kedvezőbb e a szomszéd értéke, mint a kezdőponté.

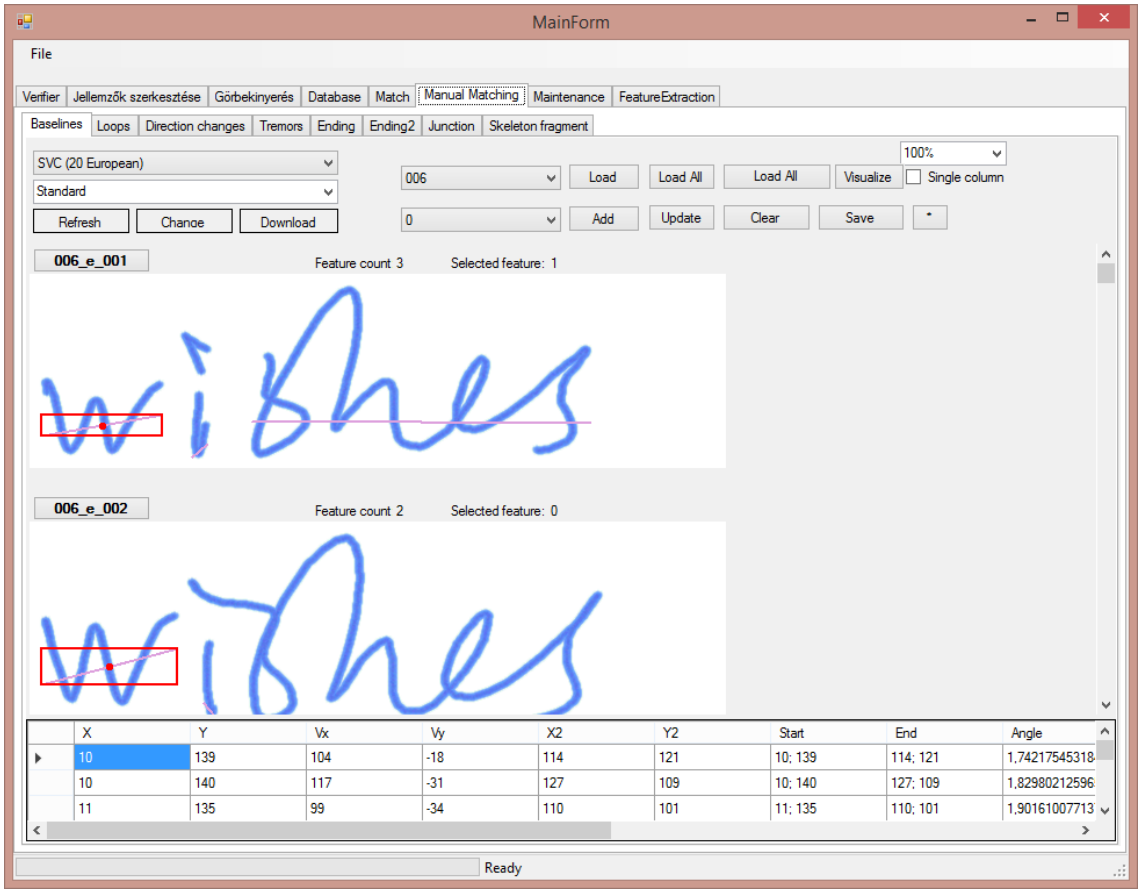
Ezután egy opcionális fázis, az *intesification* következik egy *path relinking* algoritmus megvalósítással. A kapott eredményt ezután hozzávesszük az eredményhalmazunkhoz, amiből aztán a legvégén kiválasztjuk a legjobb megoldást. A ciklus után szintén egy nem kötelező fázis jön, a posztoptimalizálás. Itt egy végső, tetszőleges szűrést iktathatunk be az utolsó kiválasztás előtt. Itt megjegyeznénk, hogy a saját implementációnkban kihagytuk az opcionális lépéseket, de erről még később írunk.

A mohó algoritmus megvalósítására többféle ajánlott konstrukció is létezik set packing probléma esetén, ennek pontos működését is hamarosan részletezzük.

2.6 Aláírás hitelesítő rendszer (AHR)

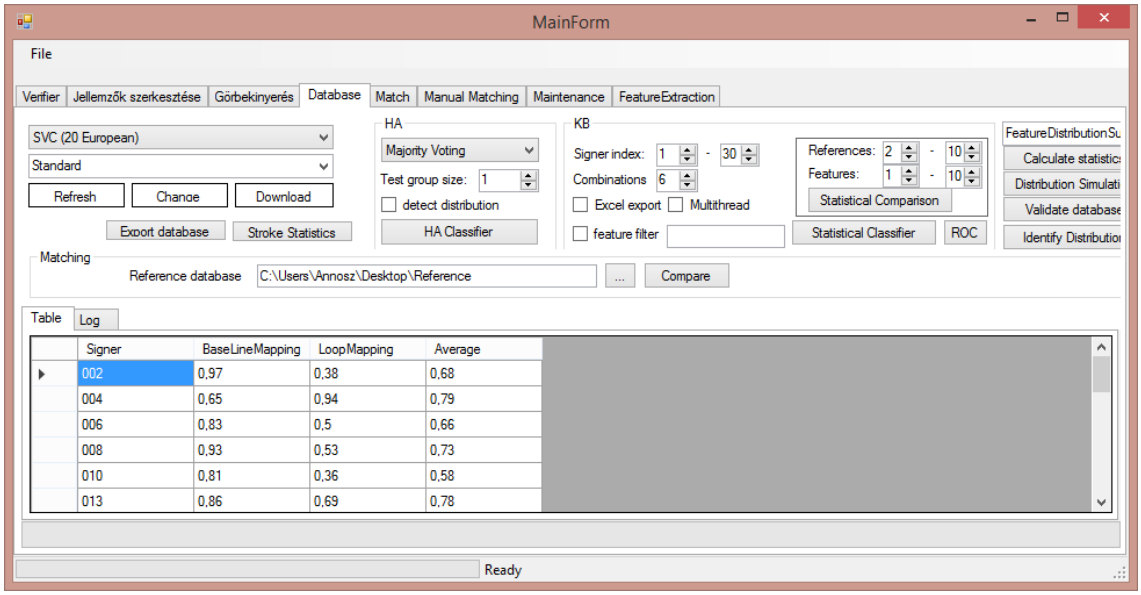
A párosítási algoritmusok pontosságának tesztelésére rendelkezésünkre áll az AHR keretrendszer. Ez lehetővé teszi a párosítási algoritmusok könnyű beillesztését az aláírás hitelesítési folyamatba, és az eredményeinek összehasonlítását egy kézzel létrehozott referencia párosítással.

A párosítások adatainak tárolása XML dokumentumban történik, amit majd az összehasonlító elemez. A mentett adatok elemzése és párosítása az AHR Manual Matching fülén elérhető (*4. ábra*), ahol minden egyes jellemzőpárra végignézhetjük, hogy melyik aláírásból melyik jellemzőt vette bele az algoritmusunk az általa helyesnek vélt megoldásba. Ez a nézet hasznos az aláírások egyenkénti elemzéséhez és a referencia párosítás létrehozásához.



4. ábra: Manual Matching az AHR keretein belül

A referenciával történi összehasonlítás a Database fülön elérhető (*5. ábra*). A művelet egy áttekintő nézetet fog adni a minta és a saját megoldásunk százalékos hasonlóságáról, aláírásonként és jellemzőnként lebontva és aggregálva. Ez a lehetőség hasznos az eredmények nagy mennyiségű, statisztikai elemzéséhez.



5. ábra: Összehasonlítás az AHR keretein belül

3 Párosítási algoritmusok

Ebben a fejezetben bemutatásra kerül 5 különböző párosítási algoritmus, melyek közül az első 3 mások munkája, míg az utolsó kettőt mi készítettük. Dolgozatunk fontos célja nem csupán saját munkánk bemutatása, de más megoldásokkal történő összehasonlítása és ezáltal kontextusba helyezése is volt.

A megoldások közül alfejezetenként egy-egy kerül pontos leírásra a következő formában: Először áttekintést nyújtunk az algoritmusról, majd folyamatosan egyre közelebbről, részleteiben vizsgáljuk meg: milyen motivációk hívták életre, hogyan közelíti meg a problémát, mik a főbb lépései (néhol pszeudó kóddal vagy ábrával magyarázva). Másodkörben az általánosságban bemutatott algoritmus konkrét megvalósításáról lesz szó: hogyan néz ki az AHR-ben, miket kellett megfontolni, milyen kihívásokkal kellett megküzdeni a programozása során stb. Végső soron pedig a mérési eredményeinket mutatjuk be: adott aláírónál milyen paraméterezéssel futott le a leggyorsabban vagy legpontosabban, átlagban és összességében milyen eredményeket ért el stb. Ennek pontos módszertanáról a következő alfejezetben írunk.

3.1 Mérési módszertan

Mielőtt az algoritmusok vizsgálatára rátérnénk, az alábbi alfejezetben röviden összegezzük a mérési módszertanunkat. Bemutatásra kerül az adatbázis, ahonnan a mintákat vettük, felsoroljuk a vizsgált jellemzőket, tisztázzuk a pontosságot definiáló mérőszámokat (pl. mit jelent a gyakorlatban, hogy egy algoritmus futásideje „jó”), majd végül azokat az egyéb tényezőket és általunk meghatározott szabályokat, amik egy-egy mérés lefutás kimenetelére hatással voltak és meghatározták, hogy az adott eredményt végül elfogadtuk vagy elvetettük.

3.1.1 Adatbázis

A mérések során bemenetként felhasznált szkennelt aláírásképek az SVC20 adatbázisból származnak. Ez egy nagyobb méretű adatbázisnak, az SVC2004-nek a része: 20 aláíró 20 darab eredeti és 20 darab hamisított kézjegyét tartalmazza [22].

Az adatbázis kutatási célokra szabadon felhasználható, teljesen publikus és ingyenes, valamint on-line információkat tartalmaz.

3.1.2 Vizsgált jellemzők

Egy aláírást számtalan, már korábban említett jellemző alapján vizsgálhatunk, amelyek közül megkülönböztetünk globális és lokális jellemzőket. Globálisnak nevezzük az egész aláírásra vonatkozó adatokat, mint például az aláírás magasság, szélessége vagy lefedett területe. A lokális jellemzők pedig az aláíráson belül találhatók, az aláírás egy-egy különleges pontjára vonatkoznak.

Dolgozatunkban kettő jellemzőre fókuszáltunk a mérések során: alapvonalak és hurkok párosítása.

Az alapvonalak az alábbi tulajdonságokkal rendelkeznek: x és y tengelyen vett kezdeti valamint végpont, alapvonal hossza és dőlésszöge a vízszinthez képest. Egy-egy aláírásnak jellemzően nem csak egyetlen alapvonala van, hanem az írásképtől függően több is (*6. ábra*).

6. ábra: Több alapvonallal is rendelkező aláírások

Hurok alatt egy görbével zárt területet értünk és a következő tulajdonságokkal jellemezhetjük: hurok területe, kerülete, középpontjának koordinátái, köré írt téglalap vagy kör (*7. ábra*). A hurkok azonosítása korántsem triviális feladat: még emberként is nehezen döntjük el, hogy például az ’o’ betűnél hány darab hurkot találunk, pláne ha több ilyen, kicsit eltérő aláírásunk van és köztük kell az azonos vagy csak hasonló jellemzőket megtalálni.

7. ábra: Hurkok egy-egy aláíráson szemléltetve

Ennek megfelelően a képelemző és jellemzőpárosító algoritmusok is általában az alapvonalakon érnek el jobb eredményeket, így ezeket a mérésekben mindig külön kezeljük, egyenkénti és összesített eredményeikkel megjelenítve.

Az algoritmusok által kimenetként adott párosítások pontosságának eldöntése fontos kérdés, ugyanis ezzel dönthető el, hogy egy algoritmus mennyire „jó”. A mostani metodika szerint az eredményeket az általunk optimálisnak vélt referenciapárosításokkal hasonlítjuk össze mind a 20 vagy 40 minta esetén. Minél hasonlóbb az előállított kimenet az általunk létrehozotthoz, annál közelebb van az algoritmus megoldása az optimálishoz.

Jelen pillanatban kettő távolságfüggvénnyel állapítjuk meg az eredmény „jóságát”, egyik az alapvonalak, másik pedig a hurkok vizsgálatára, viszont működési elvük közös: meg kell határozni, hogy hány azonos elem van a két, vagyis kimenet és referencia halmazban az összes elemhez képest. Tehát ha minden egyes elem megegyezik, akkor 100% a hasonlóság.

3.1.3 Futtatási módszertan

A tesztek során kapott futásidőket és pontosságokat egy táblázatba gyűjtöttük ki, ahol a további számolásokat végeztünk.

A mérések során egy-egy teszt futási idejét 10 percben maximalizáltuk, – néhány kivételtől eltekintve – az olyan teszteket, ahol egy aláíróra az algoritmus futása több mint 10 percig tartott volna, elvetettük. Ez nem azt jelenti, hogy az algoritmus

rossz eredményt ad, (majd mutatunk példát egy-két e feletti futásidőre is), hanem csupán így próbáltuk maximalizálni a vizsgált esetek számát. Ahogy azt látni fogjuk (pl. GRASP algoritmusnál), a leállást biztosító iterációs számot meghatározó paraméterek kisebb értékre vétele általában a pontosság rovására ment, így nem lehet ezeket a végtelenségig csökkenteni. Természetesen az ilyen paraméterekből minél nagyobbat állítunk be, annál pontosabb eredményt kell, hogy kapjunk, azonban valahol meg kellett húzni a határt a futási idő és pontosság között.

Minden egyes aláíróra 10-10 darab tesztet futtattunk (alapvonalakra és hurkokra egyaránt), hogy releváns méretű alaphalmazunk legyen. Ekkora nagyságrendnél már látható az algoritmus helyes működése, ugyanakkor ez a szám nem eredményez kezelhetetlenül nagy adathalmazt sem.

Ha ezektől a szabályoktól eltérünk (például nem teljesítette elegendő adat a feltételt), azt mindig az adott algoritmus mérési eredményeinél fogjuk jelezni.

3.1.4 Az eredmények értelmezése

Az eredményeket minden esetben az átlagos jellemzőszám függvényében ábrázoltuk, mivel ez az elsődleges tényező, ami befolyásolja a futási időt és a pontosságot.

Fontos azonban kitérni arra, hogy a diagramok nagyon ritka esetben ábrázolnak szép, sima egyenest, amin egyszerűen megfigyelhető az exponenciális jelleg. Ennek oka, hogy bár tényleg a jellemzőszám az, ami a legnagyobb hatással van a különböző mért eredményekre, számtalan más tényező is módosíthatja ezt. Az algoritmusok például egyértelműen nehezebb döntési helyzetben vannak, és kisebb esélyjel találják meg az ideális párosítást, ha a jellemzők közel vannak egymáshoz, vagy nagyon hasonlóak, mintha ugyanazon jellemzőszám mellett azok jól szétszórva vannak az aláírásban (*8. ábra*).

8. ábra: A 4,675 átlagos jellemzőszámú Beny és az 5,35 átlagos jellemzőszámú Debbie, melyek közül egyértelműen a nagyobb jellemzőszámú hurkainak azonosítása a könnyebb feladat

3.2 Véletlenszerűsített mohó algoritmus

Tóth Csaba szakdolgozata alapján készült [23], elsőként bemutatott algoritmusunk egy paraméter alapján tetszőleges mértékben mohó vagy véletlenszerű megoldásként működhet.

3.2.1 Algoritmus bemutatása

Az implementáció a második fejezetben bemutatott set packing problémára épül. A véletlenszerűsítés miatt azonban a mohóság nem teljes, és a futás során nem biztos, hogy minden iterációban a legjobbnak tűnő megoldást választja ki a program: így szerencsés esetben elkerülhető a tisztán mohó megoldás legnagyobb problémája, vagyis hogy nem globálisan optimális eredményt ad.

A véletlenszerűsített keresést tetszőleges sokszor megismételve a kapott megoldások közül a legjobbat választjuk ki, így növeljük az esélyét annak, hogy az optimális megoldást találjuk meg. Ugyanezt a gondolatot vezeti tovább a GRASP algoritmus, amit később mutatunk be ebben a fejezetben.

Tóth Csaba megvalósítása két fő lépésből áll: jellemzőkből halmazok létrehozása, majd e halmazok, mint bemenet átadása a párosító algoritmusnak, ami elvégzi magát a párosítást. A halmazok létrehozásának pontos működésére nem térnénk ki, viszont a szempontokat röviden bemutatjuk, hiszen a GRASP is ezeket a halmazokat használja, így ennek a gyorsítására is szükség lehet a jövőben. Itt a legfőbb nehézség, hogy a kinyert jellemzőkből minél kevesebb számú halmazt hozzunk létre, de úgy hogy ne hagyjunk el olyan lehetséges halmazt, amit esetleg később kiválasztana az algoritmus, mint lehetséges megoldást.

Az alkalmazott heurisztikák a következők:

- Két élt nem kötünk össze egymással (vagyis nem kerülnek azonos halmazba), ha azok x koordinátája nagyban eltér, vagyis az átlagnál nagyobb a távolság a két jellemző között.
- Ha találunk két nagyon hasonló elemet, akkor nem nézünk további pontokat a kiinduló és vizsgált pont aláírásán belül. (A hasonlóság egy becsült hasonlóság 0,625-szörösén belül van)

- Ha az előbb említett becslés 0,9-szeresén belül van a jellemzőhöz talált másik jellemző, akkor nem keresünk tovább párt a kiinduló ponthoz.

Maguk a halmazok tartalmazzák az aláírások sorszámát és a benne éppen megtalálható jellemzőt páronként, illetve egy pozitív egész számot, amely a halmazt jellemzi a benne lévő párok hasonlóságát kifejezve (minél kisebb ez a szám, annál hasonlóbbak az elemek).

3.2.2 Megvalósítás

A bevezetés után most pedig térjünk rá magára a párosító algoritmusra. A véletlenszerűsített mohó algoritmus első lépése, hogy hasonlóság szerint növekvő sorrendbe teszi a fentebb bemutatott halmazokat és ebből választja ki a 3 legjobbat eltérő valószínűségekkel (legnagyobb valószínűséggel a legjobbnak tűnőt, kisebb valószínűséggel a következőt és így tovább). A valószínűségek csökkenők a jelöltek között, néhány iteráció alatt előfordulhat, hogy nem a legjobbat választja be, így elkerülhető a mohó keresés problémája.

Az, hogy pontosan mi minősül jobbnak, egy lehetséges eredmény megtalálásakor, azt a következő heurisztika írja le:

- Ha nagyobb kardinalitású párosítást találtunk VAGY
- Talált élek hasonlósági értékének összege osztva élek száma + 1 kisebb, mint az ilyen eddig talált legjobb hányados

Az algoritmus leáll, ha elérte a keresésre szánt iterációszámot. Ez alapesetben 20, viszont paraméterezhető a jelöltek számával egyetemben, ami az előbb említett 3 alapértéken értelmezett. A mérési eredményeknél pontosan feltüntettük, hogy aktuálisan milyen paraméterezéssel volt futtatva az algoritmus, mert előfordult, hogy a paramétert túl nagynak választva nem futott le adott időlimiten belül.

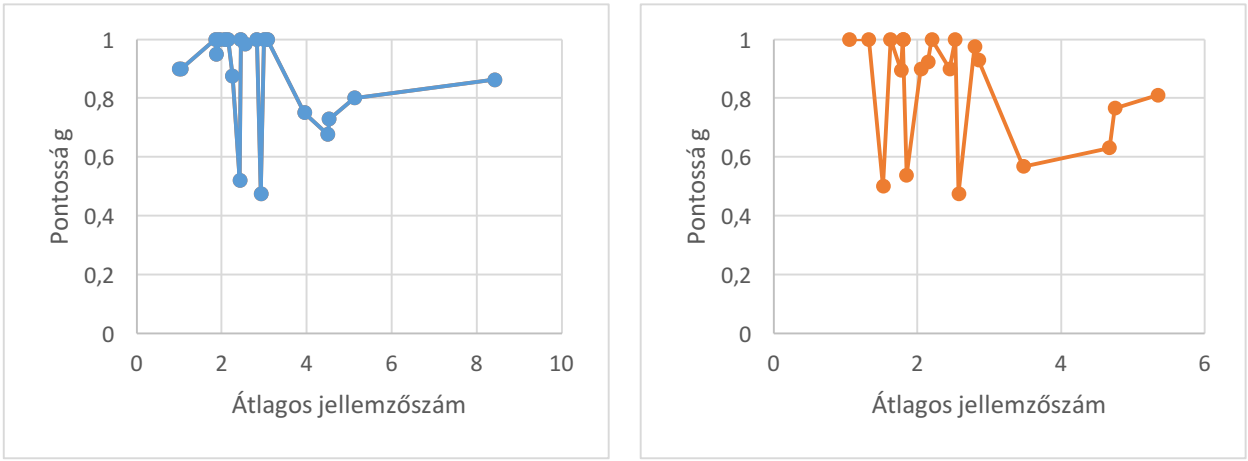
Tapasztalatunk szerint az algoritmusról elmondható, hogy ha a halmazok alapból „jó” értékeket tartalmaznak, („jó” alatt pedig azt értjük, hogy a referenciamegoldáshoz legjobban hasonlító élek hasonlóságai az elsők között vannak sorba rendezés után), akkor gyorsan megtalálja a 100% vagy ahhoz közeli megoldást, viszont ha rosszabb (vagyis nagyobb) ez a hasonlósági érték, akkor nagy valószínűséggel sosem fogja választani. Itt sajnos előjön, hogy az algoritmus inkább mohó és lokális finomítás nélkül sokszor nem fogja megtalálni a referencia megoldást.

3.2.3 Mérési eredmények

Ez az algoritmus a – GRASP-hoz hasonlóan és a többi algoritmussal ellentétben – 40 helyett csak 20 mintán lett tesztelve. Ennek az oka, hogy 4-5 jellemző esetén már nagyságrendekkel lassabb, mint a többi algoritmus, valamint emiatt az eredeti projektjében is 20 mintára volt beállítva a vizsgálat. Természetesen így csak a GRASP-pal összemérhető, a 40 mintán futókkal nem fair a mért eredmény.

A véletlenszerűsített mohó keresést kíváncsiságból azért futtattuk a 40 mintás környezetben is, itt azonban általában az eredmény nem haladja meg az 50%-os pontosságot, hiszen 20 darab minta hátrányában indul.

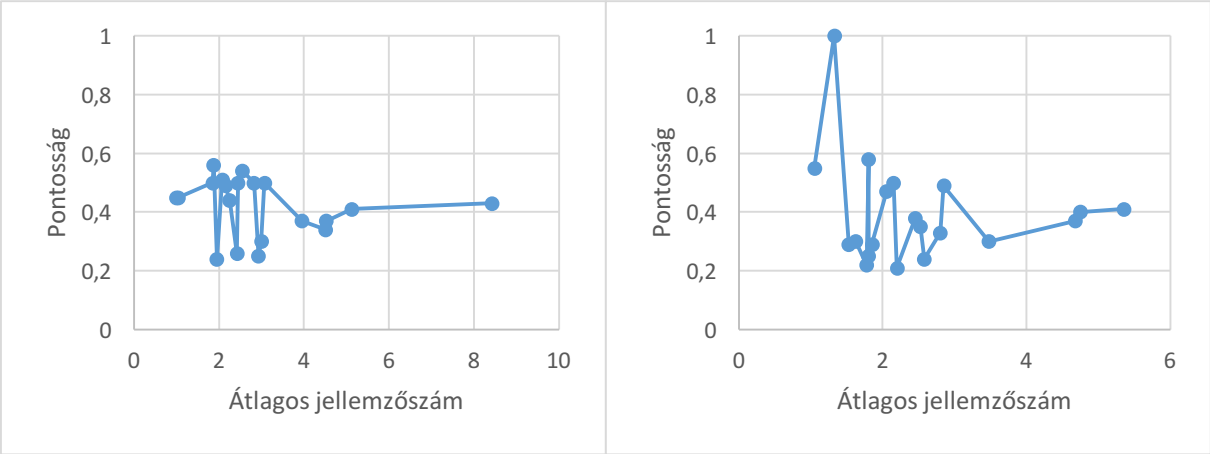
Először a pontosságot vizsgáltuk meg az eredeti, 20 mintás környezetben, az alapvonalakra és hurkokra az alábbi eredményeket kaptuk:



9. ábra: Pontosság alapvonalak és hurkok vizsgálata esetén

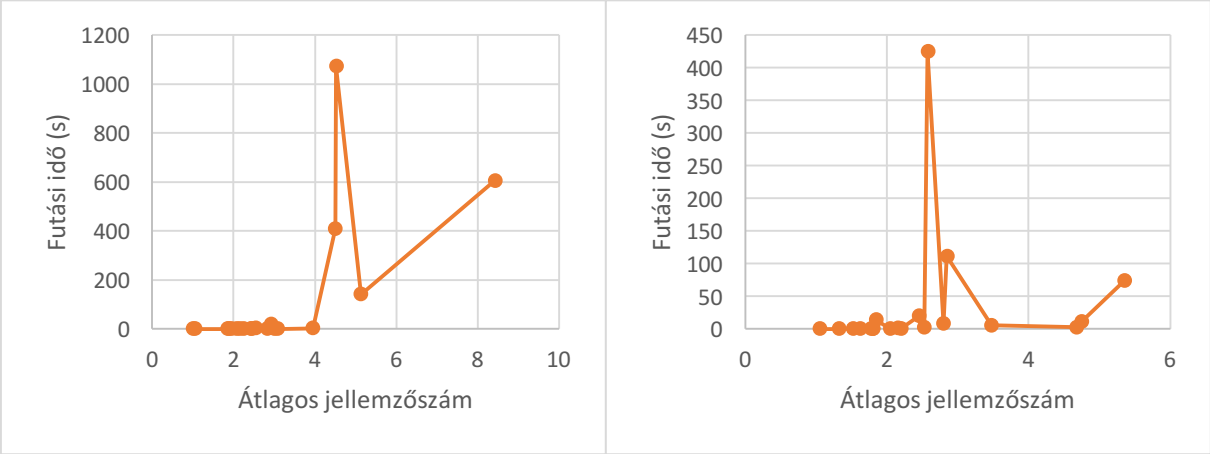
Mindkettő esetben jól látható, hogy átlagban a jellemzőszám növekedésével a pontosság romlik, ugyanakkor egyfajta „pergés” figyelhető meg a kimenetek eredményében. Ez egyértelműen a mohóságnak köszönhető, a lényegesen rosszabb kb. 50%-os eredmények esetén az optimális megoldás nem a halmaz elején van, így a mohó megoldás által legjobbnak tűnő választás globálisan csak egy ilyen gyengébb eredményt jelent.

Ugyanezt a mérés elvégeztük a 40 mintás teszten is, de itt természetesen a 20 mintás hátrány miatt összességében lényegesebben rosszabb eredmények születtek, ugyanakkor a „pergés” itt is megfigyelhető:



10. ábra: Alapvonalak és hurkok esetén mért pontosság 40 minta esetén

A pontosságok mérése utána térjünk át a futásidők mérésére. Ahogy már korábban említettük, ez az algoritmus nagyságrendekkel lomhább vetélytársainál, így 20 mintával is lényegesebben lassabb időt futott, mint pl. a karom-mentes párosítás.



11. ábra: Futásidők alapvonalak és hurkok párosítása esetén

Ahogy az alapvonalas méréseknél is látható, kétszer is átlépte a 10 perces korlátot, a jellemzők növekedésével a futásidő is nő. A 24-es, illetve 10-es aláírásnál van egy tüske, ennek az egyszerű oka az, hogy a párosító algoritmus bemenetéül szolgáló halmazok itt a legnagyobb méretűek az alapvonalak, illetve hurkok esetén. Az alapvonalaknál a kiugró érték 3 071 127, a hurkoknál ez 2 432 797 halmazt jelent. Az egyszerűbb aláírások esetén ez a szám még a 10 000-et sem éri el. Az egyes aláírásokhoz tartozó pontos halmazméretek a függelékben találhatók.

3.3 Teljes gráfbejárás

A jellemzőpárosítás során létrejött teljes gráf bejárására épülő algoritmus Berceli Zoltán diplomaterve alapján készült [2], és két különböző változata is a rendelkezésünkre áll a mérések elvégzéséhez.

3.3.1 Algoritmus bemutatása

Az algoritmus az aláírásokból a korábban ismertetett módon elkészíti a problémát leíró gráfot, melynek azt a jellemzőjét használja ki elsősorban, hogy egy aláírásonként nézve teljes gráf jön létre: minden aláírás minden jellemzője párosítva van az összes többi aláírás jellemzőivel. Nem fogja él összekötni azonban az egy aláíráson belüli jellemzőket, így nem fordulhat elő, hogy amikor a gráfunkat valamilyen formában bejárjuk, egy aláírásból választunk egy jellemzőnek párt.

Az algoritmus ezután kétféleképpen tudja bejárni a gráfot, és kiválasztani a megfelelő párosítást: az első módszer egy útkeresés, mely mindig az eddigi jellemzőkhöz legjobban hasonlító jellemzőt veszi be a párba, így lényegében egy hiperél mentén próbál végignavigálni, míg a másik, csillagszerű bejárás egyetlen csúcsból indul el minden másik aláírás felé, és veszi az ahhoz legjobban hasonlítókat.

3.3.2 Megvalósítás, változatok

Mindkét bejáráshoz el kell dönteni, hogy melyik aláírásból induljunk ki, mivel ez nagyban meghatározza az eredményt: például a jellemzők száma annyi lesz, amennyi a kiinduló aláírásban volt. Emiatt a legkézenfekvőbb megoldás kiválasztani az aláírások jellemzőszámának móduszát, és az első olyant alapul venni, ami ennyi jellemzővel rendelkezik.

Ezek után a kezdő aláírás minden jellemzőjéhez megpróbáljuk azokat párosításba helyezni. Az első, útkereső megoldásnál ehhez mindig a gráf összes éle közül keressük meg azt, amelyik a legkisebb súlyú, és olyan aláíráshoz tartozik majd, amelyikből eddig még nem találtunk jellemzőt. A második módszer ettől annyiban különbözik, hogy ahelyett, hogy egy utat végigjárva, mindig a legutolsó „állomáshoz” leghasonlóbb jellemzőt szeretnénk megtalálni, végig az eredetileg kiválasztott alap aláíráshoz keressük a megfelelő párokat minden aláírásból.

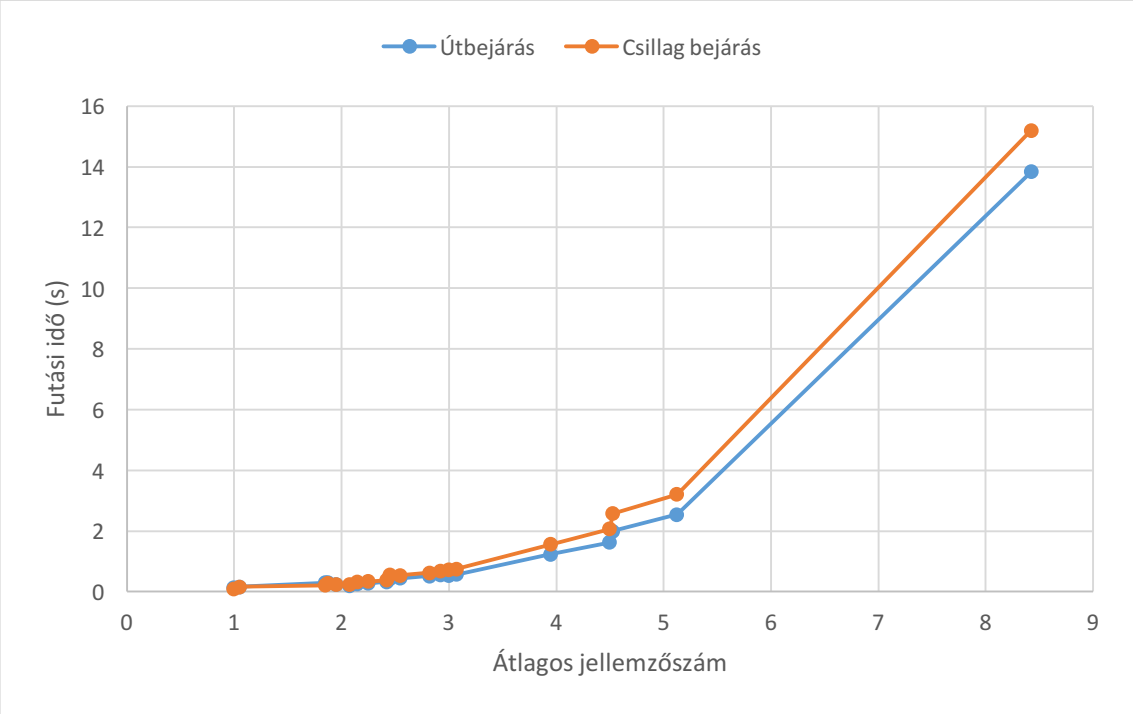
Amikor bizonyos jellemzők már bekerültek a párosításba, azokat eltüntetjük a gráfból, és tovább folytatjuk a megmaradt csúcsokkal, addig, amíg az első aláírás

minden jellemzőjéhez nem találtunk párt. Az algoritmus determinisztikus, azaz mindig ugyanazt az eredményt találja meg.

3.3.3 Mérési eredmények

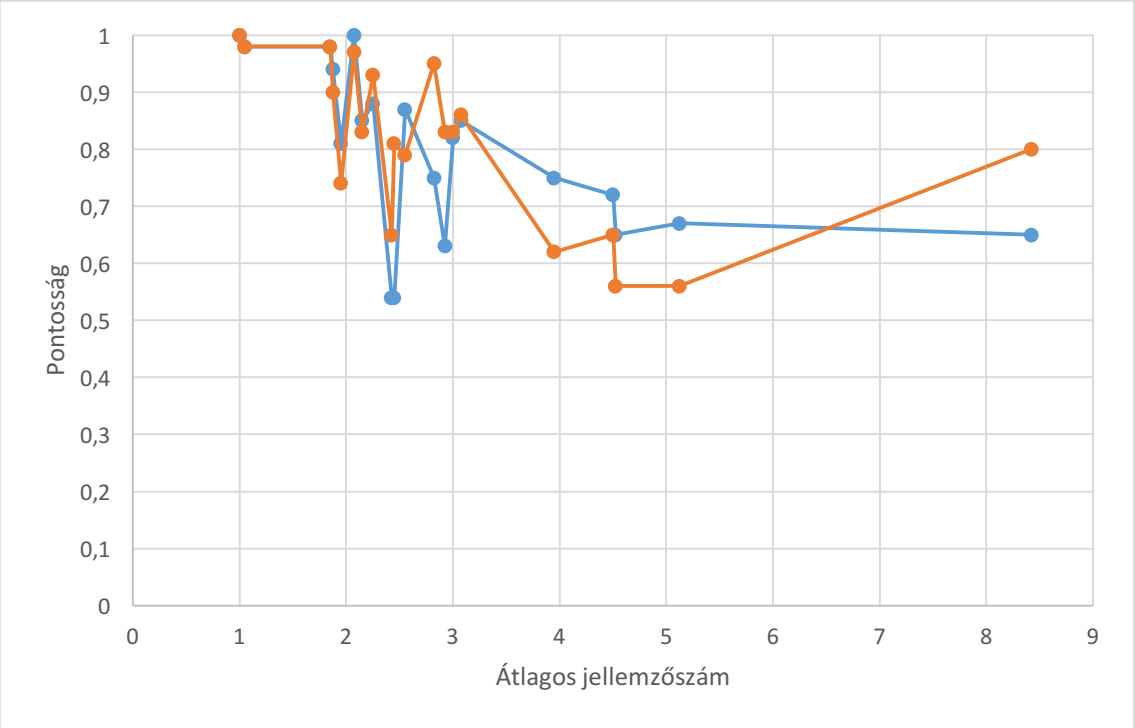
Az algoritmus eredménye átlagosan csak 65,35%-os és 66,25%-os egyezést mutat a referenciánkkal (az első és második változatra megfelelően), és futási ideje is meghaladja egyes aláírásokra a 10 másodpercet, így nem mondható a leghatékonyabb megoldásnak. Általánosságban azonban jól szemlélteti, hogy hogyan viselkednek az egyszerűbb, mohó algoritmusok egy komplexebb problémával szemben, ahol nem feltétlenül elég olyan egyszerű, lokális információk alapján dönteni, hogy melyik két jellemző hasonlít egymásra legjobban. A megállapításainkat alátámasztó mérések közül most az alapvonalakra vonatkozóakat mutatjuk be, mivel a hurkokkal ehhez teljesen hasonló eredményeket értünk el.

A futási időről megállapíthatjuk, hogy bár minimális mértékben, de a csillag bejárás lassabban végez, mint az első változat. Mindkettőnél megfigyelhető azonban a meredek exponenciális jelleg, ami a nagy jellemzőszámnál jelenik meg. Szerencsére a futási idő ennek ellenére se lépi túl sose a 15 másodpercet (*12. ábra*).



12. ábra: A teljes gráfbejárás futási ideje az átlagos jellemzőszám függvényében alapvonalak vizsgálata esetén

Az eredmények pontossága ezzel szemben jóval váltakozóbb: bár a második változat 1%-kal jobb átlagos hasonlóságot ad a referencia aláírással, nem mondható meg egyértelműen, hogy mely aláírásoknál teljesít jobban (*13. ábra*). Az eltérések fő indoka egyszerűen az algoritmus működése lehet: míg az útbejárás könnyebben „eltér" attól, amihez eredetileg hasonlót keresünk, így könnyebben talál rossz megoldásokat is, ugyanakkor pont emiatt kevésbé érzékeny egy rosszabb kiinduló aláírás kiválasztására. A csillag bejárás ennek pont ellentéte, az eredmények váltakozása pedig ebből a kettősségből ered. Külön pozitívum azonban utóbbinál, hogy a magasabb jellemzőszámmal rendelkező aláírásra jobb eredményt tudott adni, hisz a legtöbb hasonló módszer hátránya, hogy minél összetetted egy aláírás, annál kevésbé mondható helyesnek a létrejött párosítás.



13. ábra: A teljes gráfbejárás pontossága az átlagos jellemzőszám függvényében alapvonalak vizsgálata esetén

3.4 Kiterjesztett magyar módszer

A következő bemutatott módszer Dr. Kővári Bence doktori disszertációja alapján készült [24], és a magyar módszer használatát terjeszti ki több dimenzióra.

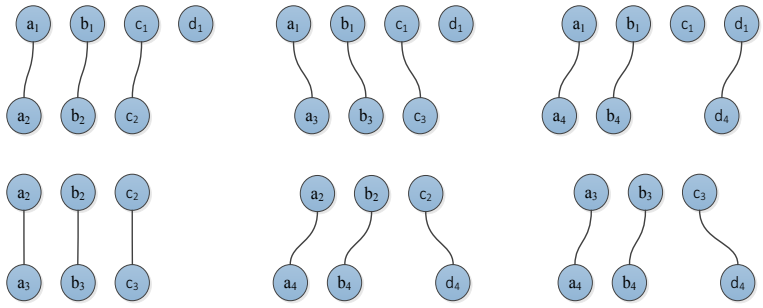
3.4.1 Algoritmus bemutatása

Miután a jellemzőpárosítási problémát leíró adatokat átalakítottuk egy többdimenziós páros gráffá, az algoritmus a magyar módszert alkalmazza. Ennek segítségével két aláírás közt mindig meg tudjuk állapítani az optimális párosítást. Ez természetesen csak lokális információkat ad, az, ami ezt összefogja, és egy globális párosítást hoz létre belőle, az a magyar módszer segítségével megadott párok megfelelő kombinálása: ehhez a két jellemző közti kapcsolatokat csoportosítjuk jellemzőnkként, majd a „legjobban összekapcsolt" csoportot választjuk ki egy párosítást leíró hiperélnek.

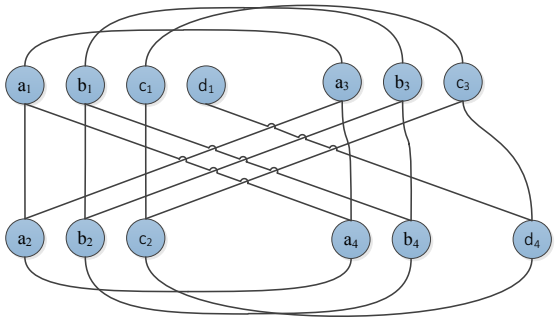
3.4.2 Megvalósítás

Az algoritmus futása a következő lépésekből áll:

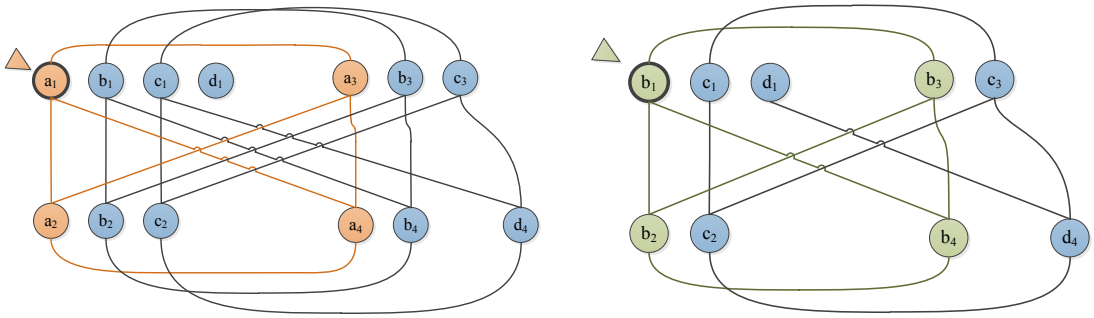
- 1. Minden aláírás-párra számoljuk ki a magyar módszerrel az ideális lokális párosítást (*14. ábra*).
- 2. A gráf minden egyes csúcsára számoljuk meg, hogy hány másik csúccsal van összekötve, és jegyezzük ez meg, mint az adott részgráf élszámát (*15. ábra*).
- 3. A megkapott legnagyobb élszámú részgráfot vegyük el a gráfból – ez lesz az ebben az iterációban megtalált párosított jellemző -, és amíg a megmaradt csúcsok fokszáma nem elég kicsi, térjünk vissza a 2. ponthoz (*16.* ábra*, 17. ábra*).



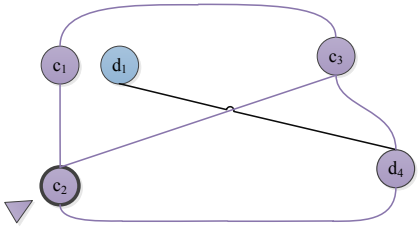
14. ábra: Párosítás minden aláírás között [24]



15. ábra: A teljes gráf [24]



16. ábra: A 3. lépés 1. iterációja és 2. iterációja

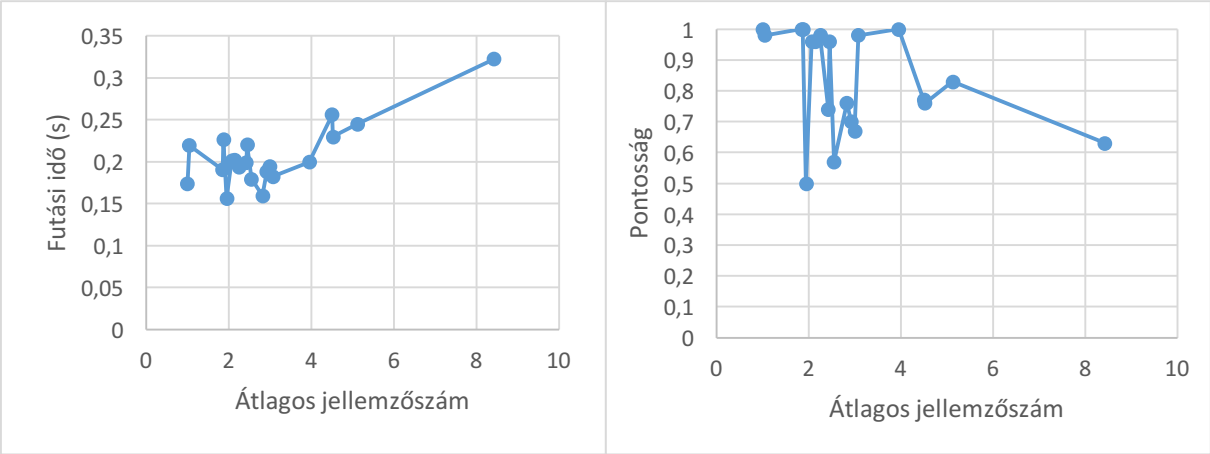


17. ábra: A 3. rész 3. iterációja (ahol tévesen a d_4 jellemző is bekerül a párosításba) [24]

3.4.3 Mérési eredmények

A magyar módszert használó algoritmus alapötlete egyszerű, és ehhez képest gyors és pontos eredményeket tud elérni. Mind a 20 tesztesetre, 2-2 jellemzőt vizsgálva is átlagosan 7,45 másodperces futási idővel létrehozza a párosításokat, amik az AHR értékelője szerint 76,375%-ban felelnek meg a referencia aláírásnak. A megoldás determinisztikus, minden futás után ugyanazt az eredményt kapjuk, amiket ismét az alapvonalakhoz tartozó mérésekkel szemléltetünk, mivel a hurkokkal mért teljesítmény ezeknek megfelelő.

A futási idő a vizsgált jellemzők számának növelése esetén növekszik, de nem meredeken: a leghosszabb ideig tartó párosítás is 0,32 másodperc alatt befejeződött (*18. ábra*).



18. ábra: A kiterjesztett magyar módszer futási ideje és pontossága az átlagos jellemzőszám függvényében alapvonalak vizsgálata esetén

Az algoritmus pontossága néhány pozitívabb értéktől az egyértelműbb aláírások esetén nagyjából mindig az átlagos érték körül marad. Itt is megfigyelhető azonban az a jellemző trend, hogy az összetettebb aláírásokra rosszabb eredmények születnek.

3.5 GRASP algoritmus

A korábbi alfejezetekben, Tóth Csaba véletlenszerűsített mohó algoritmusánál már utaltunk esetleges továbbfejlesztési lehetőségekre, most pedig elérkezett az idő, hogy ezeket részletesen bemutassuk.

3.5.1 Algoritmus bemutatása

A GRASP a mohóságot kívánja kiküszöbölni az említett megoldásból. A véletlenszerűsített mohó keresés mellett egy lokális finomítást is tartalmaz, így képes a probléma szempontjából kedvezőbb megoldást nyújtani. Működésének főbb lépéseit már a második fejezetben ismertettük.

3.5.2 Megvalósítás, változatok

Ahogy korábban említettük az algoritmus bemenete ugyanazok a halmazok, amiket Tóth Csaba implementációja is létrehozott, itt nem változtattunk semmit.

A GRASP teljes egészében paraméterezhető az alábbi változókkal: alapiterációk száma, szomszédossági iterációk száma lokális keresésnél, mohósági faktor és egy opcionális jelöltszám limit. Az alapiterációk számával biztosítjuk a leállási feltételt, itt igyekeztünk megtartani a 20 lépést, azonban bonyolultabb (több jellemzővel rendelkező) aláírásoknál ez a szám túl nagy volt és nem futott le időlimiten belül. A

szomszédossági iterációk száma a lokális keresőfüggvénynek adja meg, hogy legfeljebb hányszor próbálkozzon a kiválasztott él, mint kezdőpont környezetében új eredmények találásával.

A mohósági faktor helyes megválasztása már egy bonyolultabb kérdés: ha 1-et vagy ahhoz közelit választunk, akkor előfordulhat, hogy gyengébb hasonlóságú, viszont referencia szempontjából helyes megoldásokat elvet az algoritmus. Ellenben ha túl alacsony, 0 vagy ahhoz közeli, akkor pedig a véletlenszerűsített keresés miatt gyengébb eredményeket kapunk. Pontosan ezen okok miatt mértünk más-más alfa értékekkel.

Az opcionális jelöltszám bevezetésére a bonyolultabb párosításoknál volt szükség. Ez a plusz feltétel azt garantálja, hogy a véletlenszerűsített keresés a hasonlóság alapján növekvő sorrendbe rendezett jelölthalmazból nem az összeset, hanem csak az első jelöltszám darabot veszi figyelembe.

A paraméterek ismertetése után most térjünk rá a pontos futásra. A GRASP is hasonlóan indul, mint Tóth Csaba implementációja: hasonlóságok alapján sorrendbe teszi a halmazokat, erre a már előbb említett mohósági faktor és jelöltszám miatt van szükség. Utána a legjobb megoldást inicializáljuk egy véletlen kiválasztott halmazzal, majd belépünk a ciklusba, ahol a megadott lépésszámig keresünk megoldást. Itt minden egyes lépésben kettő fázist veszünk végig: keresünk egy megoldást a véletlenszerűsített mohó kereséssel, majd a kapott eredményt bemenetül adva meghívjuk a lokális keresést, ez után pedig egy feltétel alapján eldöntjük, hogy a kapott kimenetet eddigi legjobbnak vesszük fel vagy eldobjuk.

A GRASP-nak többféle verziója is létezik a jelenlegi megvalósításban, előbb bemutatjuk az alap implementációt, majd rátérünk a verziók közötti különbségekre.

Az alap megvalósításban a véletlenszerűsített mohó keresés a mohósági faktort figyelembe véve definiál egy limitet, ami fölött nem keres megoldásokat. Azok a halmazok, amik megfelelnek a limit támasztotta követelményeknek, bekerülnek a jelölthalmazba, amiből aztán véletlenszerűen sorsolva (tehát a jelöltek közül bármelyiknek ugyanakkora esélye van a kiválasztásra) választ egyet, majd következőnek egy vele nem metszőt és így tovább, amíg el nem fogynak a jelöltek, vagyis amíg tud egymással nem metsző halmazokat találni.

Az eredményül kapott halmazt vagy halmazokat aztán a lokális keresés próbálja javítani, ami ebben a verzióban a mohó algoritmus újbóli meghívásait jelenti az alábbi heurisztikákkal:

- Ha eredményül eddig egyetlen halmazt találtunk, akkor ezt kizárva próbáljunk új megoldásokat keresni az összes halmazból VAGY
- Ha több megoldáshalmazt találtunk, akkor az összesen végig iterálva dobjuk el az aktuálisat és helyette próbáljunk meg egy globálisan jobbat találni.

De mit jelent pontosan az, hogy globálisan jobbat találni? Erre az alábbi feltételt dolgoztuk ki:

- Ha nagyobb kardinalitású párosítást találtunk VAGY
- Ha nagyobb kardinalitású párosítást találtunk ÉS a talált élek hasonlósági értékének összege osztva élek száma + 1 kisebb, mint az ilyen eddig talált legjobb hányados

Jól látható, hogy Tóth Csaba implementációjához képest ez a verzió nem fog eldobni olyan, már megtalált megoldásokat, ahol a számosság ugyan nagyobb viszont a hányados kedvezőtlenebb. Ennek a döntésnek a gyakorlati okát a méréseknél fogjuk pontosan látni, ahol az algoritmus így meg fogja találni az olyan párosításokat, ahol a referenciában kettő vagy annál több a kiválasztott halmaz.

A GRASP második verziójában lokális keresés esetén kikapcsolható a mohóság, vagyis a limit figyelembevétele, így megtalálhatunk olyan korábban kizárt halmazokat, amik globálisan nagyobb pontosságot érhetnek el, ezzel viszont a futási idő jelentősen nő.

A harmadik verzió a már említett opcionális jelöltszám limit használatát jelenti, míg a negyedik verzióban egy kiegészített heurisztikával próbálunk jobb párosítást találni: nem csak a talált halmazok számát, hanem a bennük található jellemzők számát is figyelembe vesszük, tehát például ha az egyik halmaz 9, a másik halmaz 10 jellemzőt tartalmaz, viszont előbbinek kedvezőbb a hasonlósági értéke, akkor is a másodikat fogja választani a nagyobb számosság miatt.

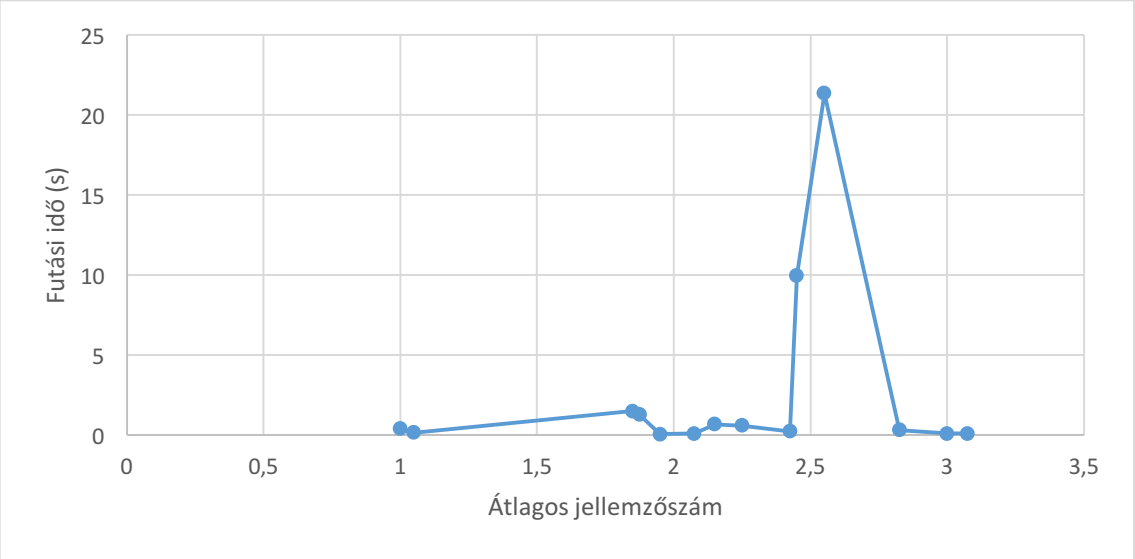
A felsorolt verziók által produkált eredményeket a mérések résznél fogjuk részletesen elemezni.

3.5.3 Mérési eredmények

Fontos az elején leszögezni, hogy a GRASP-ot – a véletlenszerű mohó implementációhoz hasonlóan – nem 40, hanem csupán 20 mintára futtattuk.

A GRASP-ot az előző pontban taglaltak szerint több verzióval is teszteltük. Alaphelyzetben 10-10 futtatást próbáltunk az alábbi paraméterekkel: iterációszám = 20, szomszédossági iteráció = 20, mohósági faktor = 0.5.

Az alapvonalak párosításáról elmondható, hogy ezzel a paraméterezéssel a bonyolultabb, már 4-5 jellemzőnél rendelkező aláírásoknál a GRASP nem futott le 10 percen belül, ugyanakkor a kisebb jellemzőszámú aláírásoknál viszont meglehetősen pontos, átlagosan 94,7%-os értéket produkált (*19. ábra*).

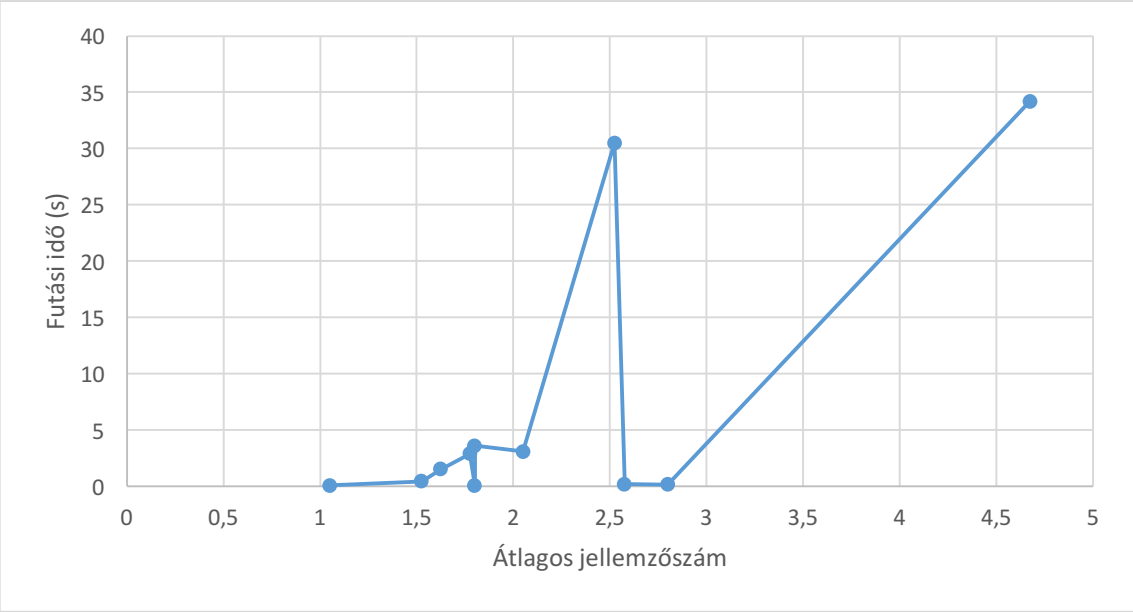


19. ábra: A GRASP algoritmus futási ideje az átlagos jellemzőszám függvényében alapvonalak vizsgálata esetén

A 25-ös aláírásnál találhatunk egy nagyobb kiugrást a futásidőben. A 4 vagy annál nagyobb jellemzőszámúak tekintve, hogy 10 percen felül futottak, nem kerültek rá a diagramra, mert eltorzítanák a megjelenítést.

Természetesen, ha az iterációszámokat csökkentjük, akkor már le fog futni az algoritmus időlimiten belül, viszont a pontossága lesz oda, a nyújtott kimenet eléggé bizonytalan lesz: 10 futásból előfordult, hogy csak 1-2-nél találta meg az optimális közelit, a többinél pedig nagy szórással hozott csak eredményeket. Tekintve, hogy itt a cél a minél nagyobb pontosság, így ezeket a megoldásokat elvetettük.

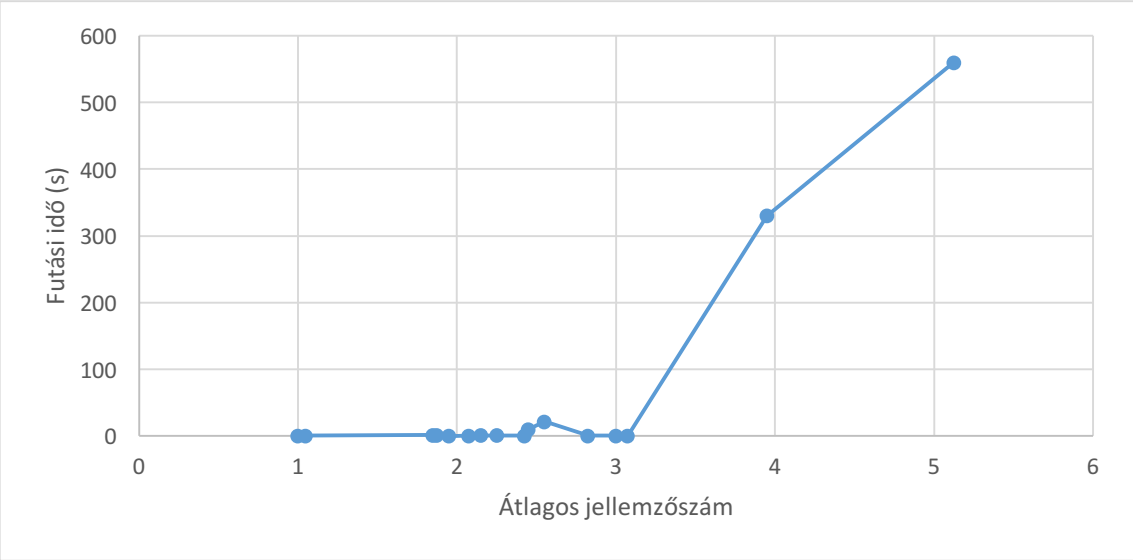
Más a helyzet a hurkok esetében. Az előbb említett paraméterezés itt sokkal több esetben (pontosan 11 aláírásnál) nem tudta tartani a 10 perces határt, itt viszont az iterációszámok 10-re való csökkentése sem rontotta jelentősen a végeredményt (*20. ábra*). Ennek oka a bemeneti halmazok méretében keresendő: korábban említettem, hogy a GRASP is változatlanul Tóth Csaba megvalósítását használja a halmazok (hiperélek) létrehozására. A nagyobb jellemzőszám potenciálisan nagyobb halmazszámot eredményez: míg 3-4 jellemzőnél ez egy 100-10000-es nagyságrendet jelent, addig 6-8 jellemző esetén túllépjük a milliós halmazméretet. Hogy ezt lecsökkentsük, bevezettük a GRASP-nál is a jelöltlimitet, amit a hurkoknál 3-ra állítottunk be. A kapott eredményeket az alábbi grafikonon tudjuk megtekinteni.



20. ábra: A GRASP futási ideje az átlagos jellemzőszám függvényében hurkok vizsgálata esetén 20-20-0.5 paraméterekkel

Az ábrán látható, hogy a 33-as aláírásnál tapasztalható egy nagyobb kiugrás, majd az 5, illetve ettől nagyobb jellemzőszámú aláírásoknál (ebben az esetben már csak 5 db ilyen aláírás van az adathalmazban) nem futott le időlimiten belül.

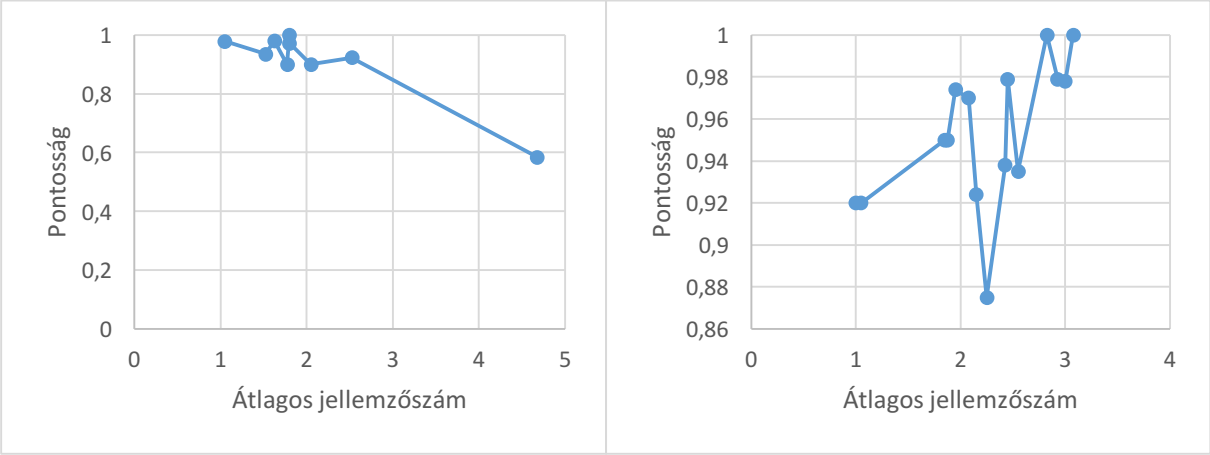
Ha azonban a különböző paraméterezések futásidejének egymáshoz való viszonyára vagyunk kíváncsiak, azt az alábbi ábrán megnézhetjük: jól látszik a jellemzőszám növekedésével az exponenciális futásidő (*21. ábra*).



21. ábra: GRASP futásidő növekedés 20,20,0.5 majd 10,10,0.5 paraméterekkel alapvonalak mérése esetén

A futásidők után térjünk rá a pontosságok vizsgálatára. Az alapvonalak esetén egy viszonylag stabil 85% feletti pontosságot sikerült azoknál az aláírásoknál elérni, ami lefutottak, ez elég jó eredménynek számít.

A hurkok esetében is sikerült tartani a pontosságot, egyedül a 35-ös aláírásnál hozott az algoritmus átlagban gyengébb eredményt, de a többinél jól teljesített (*22. ábra*).



22. ábra: A GRASP pontossága az átlagos jellemzőszám függvényében alapvonalak és hurkok vizsgálata esetén

Ahogy az előző alfejezetben írtuk, megpróbálkoztunk egy olyan verzióval is, ahol a mohósági limitet figyelmen kívül hagyjuk lokális keresés esetében, azonban az

eredmények egyáltalán nem javultak, viszont a futásidők további aláírásoknál is túllépték a 10 percet, így ezt a verziót végül elvetettük.

A negyedik verzióban ismertetett új heurisztikát kettő aláírás esetén mértük újra, ahol pontosan tudtuk, hogy ez javítani fog a pontosságon. Ez így is történt, a 2-es és 6-os aláírásoknál az átlagos pontosság alapvonalak esetén (csak erre vizsgáltuk) szintén 10 lefutást tekintve a 2-esnél 97%-ról 99,5%-ra, a 6-osnál 97,8%-ról 100%-ra nőtt. Ezek ugyan alacsony jellemzőszámú aláírások, azonban jól esett látni, hogy egy apró módosítás a heurisztikában is jelentősen tudja a kimenet stabilitását és pontosságát növelni.

Most pedig a mohósági paraméter megválasztásáról néhány szót. Több paraméterrel is próbálkoztunk, végül a 0.5 bizonyult általánosságban a legmegfelelőbbnek. Ha 0-át választottunk, akkor a véletlenszerűség miatt nagy szórással kaptunk eredményeket és ezen a lokális keresés sem tudott sokat javítani, 1-et választva a mohóság miatt pedig megint csak nem megfelelő eredményeket kaptunk, hiszen az olyan jellemző párosításokat, amelyek nem a legjobb halmazhoz tartoztak, mind elvetett az algoritmus, annak ellenére, hogy az esetek többségében az optimális megoldás nem a sorba rendezett halmaz legelejéről volt való.

Voltak olyan aláírások is, ahol a 0.5 sem bizonyult elegendőnek, az ilyen esetekben a bemeneti halmazok létrehozásán kellene még finomítani.

3.6 Karom-mentes párosítás

Az algoritmus a két egymással ekvivalens problémára, a set packingre, illetve a maximális súlyú független részhalmazok problémájára épül. Az új lépés az eddigi hasonló megoldásokhoz képest azonban az, hogy a párosítási problémát a karom-mentes gráfok használatával próbálja megoldani – a maximális súlyú független részhalmazok keresése ezáltal polinomiális komplexitásúvá tehető, cserébe azonban olyan bemeneti gráfot kell neki biztosítani, ami megfelel a feltételeknek. A célunk ezzel az algoritmussal, hogy kiderüljön, mennyire sikeres ez a csere az időigény és a pontosság szempontjából.

3.6.1 Algoritmus bemutatása

A karom-mentes párosítás két részfeladat iteratív ismétléséből áll, amíg javulást érünk el az eredményen. Először a jellemzők S halmazán halmazokat generálunk

véletlenszerűen, melyek egyben a többdimenziós párosítási probléma hiperélei is (ez a probléma set packing része), és azokból felépítjük a maximális súlyú független csúcshalmaz megkereséséhez szükséges gráfot (úgy, hogy végig figyelünk arra, hogy a karommentes maradjon). Második lépésként pedig ezek közül Minty polinomiális komplexitású algoritmusával kiválasztjuk azokat a független csúcsokat, melyek a legnagyobb összsúllyal rendelkeznek: a set packing problémáról való áttérés miatt ezek egyben azok a halmazok is, melyek a legnagyobb összsúlyú párosítást adják.

A fenti két lépésből álló sorozatot addig lehet ismételni, amíg javulást nem érünk el. Ennek időigényét és pontosságát két fontos paraméter határozza meg: a tryNumber, amivel azt jelöljük, hogy hány olyan iteráció után álljon le az algoritmus, amiben nem történt javulás, és a setNumber, ami azt adja meg, hogy egy iterációban hány halmazt generáljunk, amin futtatjuk Minty algoritmusát. Értelem szerűen mindkét szám növelése az eredmény finomodásával jár – hisz minél kitartóbban, minél több lehetséges érték közül válogatva próbáljuk megtalálni a tökéletes párosítást, annál jobb az eredményünk, de annál több időt is igényel.

3.6.2 Megvalósítás

3.6.2.1 Halmazok létrehozása

Ahhoz, hogy a maximális súlyú független csúcshalmazt megtaláló algoritmusunk olyan eredménnyel állhasson elő, ami a lehető legközelebb van a párosítási probléma megoldásához, a lehető legjobb bemenetet kell neki biztosítani: mindig igaz, hogy a legjobb megoldást csak akkor tudjuk kiválasztani, ha az a lehetőségek között van. Emiatt nagyon fontos az, hogy a heurisztikák, amik alapján véletlenszerűen létrehozzuk a halmazokat, azokat a hiperéleket, amiket le akarunk tesztelni, hogy megtaláljuk köztük a legjobb párosítást, jók legyenek.

Az első iterációban, amikor még nincsenek korábbi eredményeink, amiket felhasználgatunk, a legjobb, amit tehetünk, hogy minden aláírásból az első, majd a második, harmadik, és így tovább jellemzőket vesszük bele egyetlen halmazba, nagyjából úgy, mint ahogy a létező legnaivabb algoritmus tenné. Mivel itt egyelten aláírás egyetlen hiánya is nagy elcsúszást okozhat, ezért az alaphalmaz létrehozásához ugyanezt a folyamatot megismételjük az utolsó jellemzőktől kezdve is. Az így előállított hiperélek elég jók ahhoz, hogy ezek módosításával állítsuk elő vizsgálni kívánt

halmazokat. Későbbi iterációkban pedig e helyett a lépés helyett egyszerűen az eddig legjobbnak talált megoldást vesszük alapul és módosítjuk.

Miután van néhány halmaz, amiből elindulhatunk, meg kell alkotni azokat a heurisztikákat, ami alapján módosítjuk őket: ez az a pontja az algoritmusnak, ahol rengeteg optimalizálási lehetőség áll a rendelkezésünkre. Az tesztelések során eddig is nagyon sok kezdeti heurisztika lett elvetve, és új alkalmazva helyette, vagy pusztán az változtatva, hogy melyik lépést hányszor és milyen sorrendben használjuk. A jelenlegi változatban az eddig legjobbnak ítélt heuriszikák a következők:

- Egy véletlenszerű halmazból elhagyjuk a többitől legjobban elütő jellemzőt.
- Egy véletlenszerű halmaz egyik olyan sorában, ami az átlagosnál több jellemzőt tartalmaz, az egyik jellemzőt egy szomszédjára változtatjuk.
- Egy véletlenszerű halmaz egyik olyan sorában, ami az átlagosnál kevesebb jellemzőt tartalmaz, az egyik jellemzőt egy szomszédjára változtatjuk.
- Egy véletlenszerű halmaz egyik olyan sorában az egyik jellemzőt egy szomszédjára változtatjuk.

Mint említettük, az egyik fontos paramétere az algoritmusnak, hogy hány különböző halmaz készítése után állunk neki a maximális súlyú független csúcshalmaz keresésének. Ezt a setNumber nevű paraméterrel jelöljük, melyet megszorzunk az átlagos jellemzőszámmal: ezzel biztosítja, hogy a nagyobb aláírásokra nagyobb lehetőséghalmazból keressük meg az optimális megoldást, míg kisebbeknél, ahol jóval egyszerűbb szokott lenni az ideális párosítás, ne töltsünk felesleges időt a javítással.

A halmazok generálása közben azokból folyamatosan építjük a gráfot is, ami ahhoz kell, hogy megkereshessük benne a megoldást. Míg a probléma első fele a set packing szemléletében kerül megoldásra, ugyanis az egymást nem metsző, a jellemzőket minél jobban fedő halmazokat keresünk, itt történik az áttérés a maximális súlyú független csúcshalmaz keresésének problémájára. Hogyha a második fejezetben ismertetett polinomális igényű módszerrel építjük a gráfunkat, akkor azonban minden új csúcs beszúrása után azt is tesztelni kell, hogy ne jöjjön létre új háromszög, hisz Minty algoritmusának helye működéséhez ez a szükséges feltétel. Ha egy új set létrehozása és beillesztése a gráfba ilyen esetet eredményezne, akkor azt eldobjuk – ebben az iterációban már nem kerülhet az eset vizsgálatra.

3.6.2.2 Maximális súlyú független csúcshalmaz megállapítása karom-mentes gráfban

Ennek a problémának a megoldásához Minty algoritmusát [15] használjuk, ami a második fejezetben került ismertetésre. Az ottani pszeudokóddal megmutattuk, hogy ez a módszer lényegiben a legnagyobb súlyú P* fehér javító utak megkereséséből, és azoknak az eddigi legjobb megoldással vett szimmetrikus differenciáljának új megoldásként való vételéből tevődik össze. Most részletesebben arra térnénk ki, hogy a 2. pontként feltűntetett lépés, a bizonyos P* javítóút megkeresése hogyan történik.

2-1. Generáljunk minden 1 és 3 hosszú fehér alternáló utat;
2-2. ***Minden*** *nem szomszédos a és b szabad csúcshoz*
2-3. Legyen x_a *és* x_b *a hozzájuk csatlakozó fekete csúcs;*
2-4. ***Ha*** $x_a \neq x_b$ ***akkor***
2-5. Keressünk egy maximális súlyú fehér alternáló utat a kettő közt;
2-6. ***Iteráció vége;***
2-7. ***Ha*** *minden így generált fehér alternáló útnak negatív a súlya* ***akkor visszaadjuk*** *S-t*
2-8. Válasszuk ki a legnagyobb súlyú alternáló utat, ami P javítóút lesz*

A fenti sémában szereplő 1 és 3 hosszú javító utak megkeresése lényegében triviális lépés, hisz előbbi egy szuper szabad csúcsot jelent, míg utóbbi egy fekete csúcs két szomszédja közül könnyen kiválasztható. Az ennél hosszabb javító utak megtalálása már bonyolultabb feladat, és pontos megoldását Minty, Nakamura és Tamura [19] részletei, a mi problémánknak azonban nem feltétlenül képezi részét.

Az algoritmus legelső változata még úgy készült, hogy Minty teljes algoritmusát futtatta, azonban későbbi mérések alapján kiderült, hogy erre nincs szükség. Mivel két jellemző hasonlóságának hatalmas részét adja az x tengelyen való elhelyezkedésük, ezért a szóba jöhető optimális megoldások halmaza igen egyedien strukturált: természetes benne, hogy két jellemzőpáros az aláírás minél távolabbi részén helyezkedik el, annál „távolabb” van egymástól hasonlóság tekintetében is. Mivel így az általunk alkotott gráfban is megfigyelhető egy, a térbeli elrendezésre hasonló elrendezés, ezért nagyon ritkán fordul elő, hogy 3-nál hosszabb javító út mentén lehetne változást találni az optimális megoldásban.

Ez a tulajdonsága a jellemzőpárosítási feladatnak pedig a későbbiekben is fontos szempont lehet: ahogy Minty algoritmusából is elég egy sokkal egyszerűbb változatot használni a megoldáshoz, úgy lehetséges, hogy más, alapesetben komplexebb algoritmusokat is olyan formára tudunk hozni, ami a mi problémánkat gyorsan és pontosan meg tudja majd oldani.

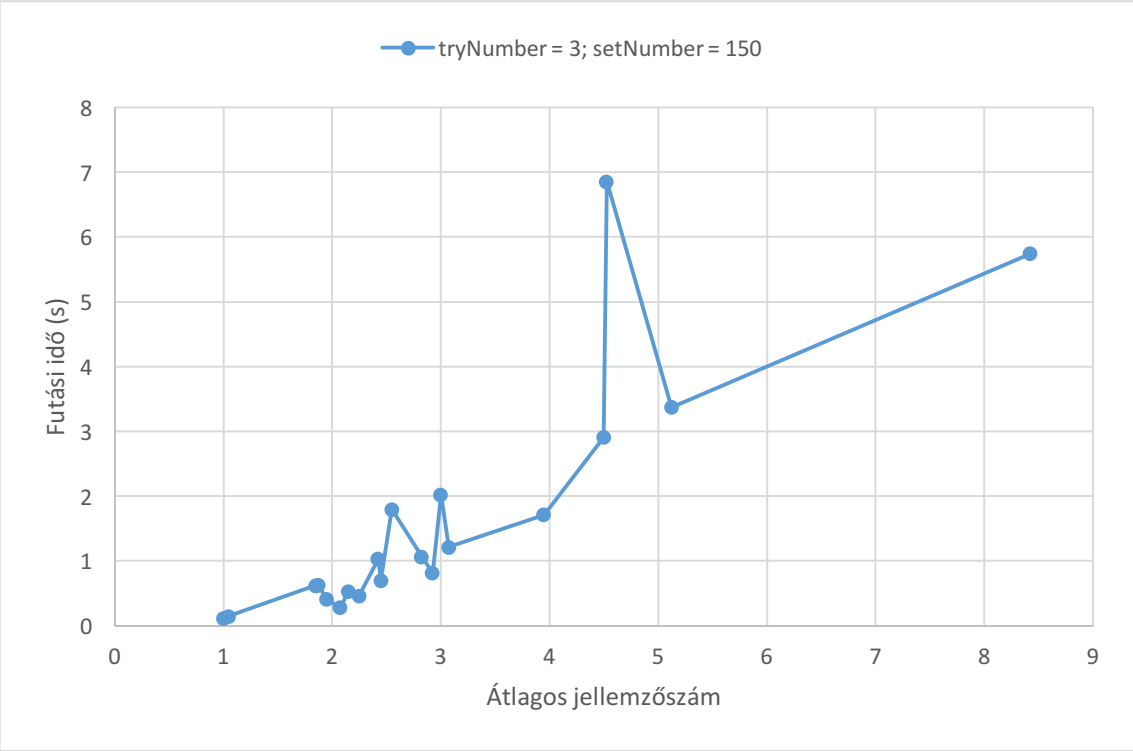
3.6.3 Mérési eredmények

A karom-mentes párosítás futási ideje és pontossága nagymértékben függ a paraméterektől, amiket megadtunk neki. Mi most a mérések során három féle különböző paraméterezést használtunk:

- tryNumber = 2; setNumber = 75: Legnagyobb futási idő < 0,5 sec; Átlagos pontosság = 82,6%
- tryNumber = 3; setNumber = 150: Legnagyobb futási idő < 10 sec; Átlagos pontosság = 83,2%
- tryNumber = 5; setNumber = 250: Legnagyobb futási idő: < 1,5 perc; Átlagos pontosság = 84,8%

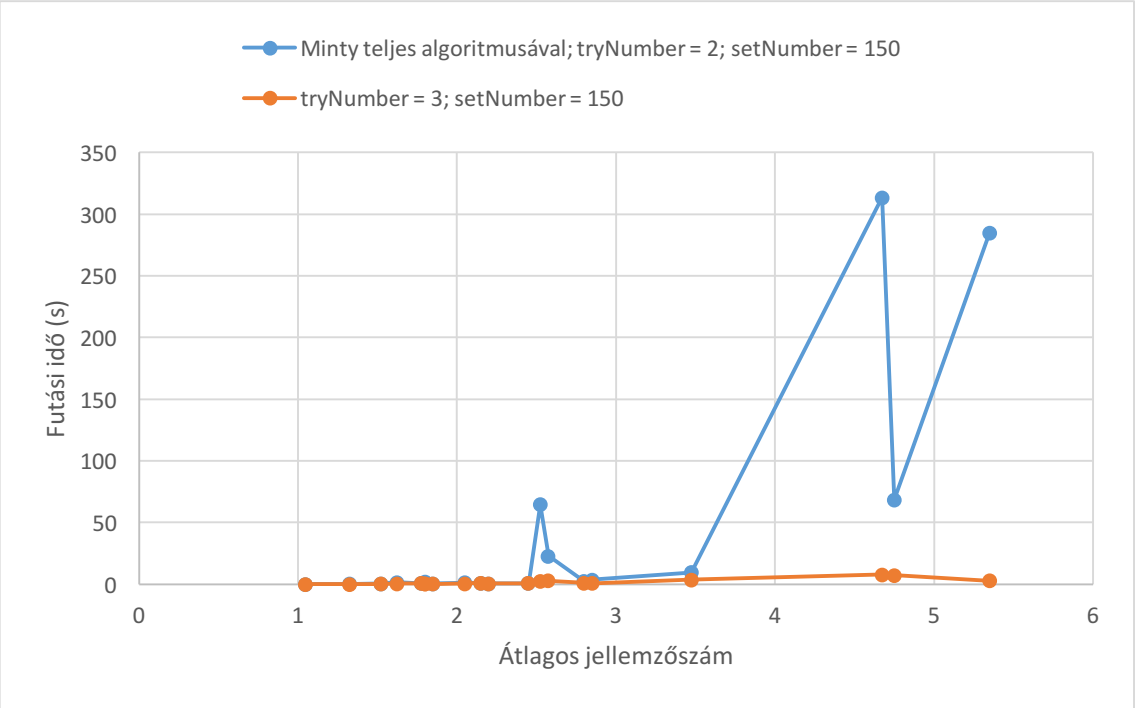
Az eredményekből az látható, hogy a más algoritmusokhoz hasonló sebesség mellett is képes 82,6% pontosságú párosítást létrehozni a referenciaaláíráshoz képest, ami egy gyors és pontos eredmény. Ennek pedig még további javítására is lehetőség van extra idő rááldozásával, így akár 85%-os pontosság felé is eljuthatunk.

A futási idő megfigyeléséhez a jellemzőszám függvényében a második felparaméterezést használjuk, mivel gyorsabb futás esetén kevésbé volt megfigyelhető az exponenciális jelleg: itt azonban jól látható, hogy míg alacsony jellemzőszám mellett képes tartani az egyszerűbb algoritmusok alacsony futási idejét, a jellemzőszám növelésével exponenciálisan nő a szükséges idő (*23. ábra*). Azonban ebben az esetben is legfeljebb 7 másodpercre volt szükség a legnagyobb aláírás vizsgálatánál is, gyakorlati használhatóságát ez az eredmény nem fogja korlátozni.



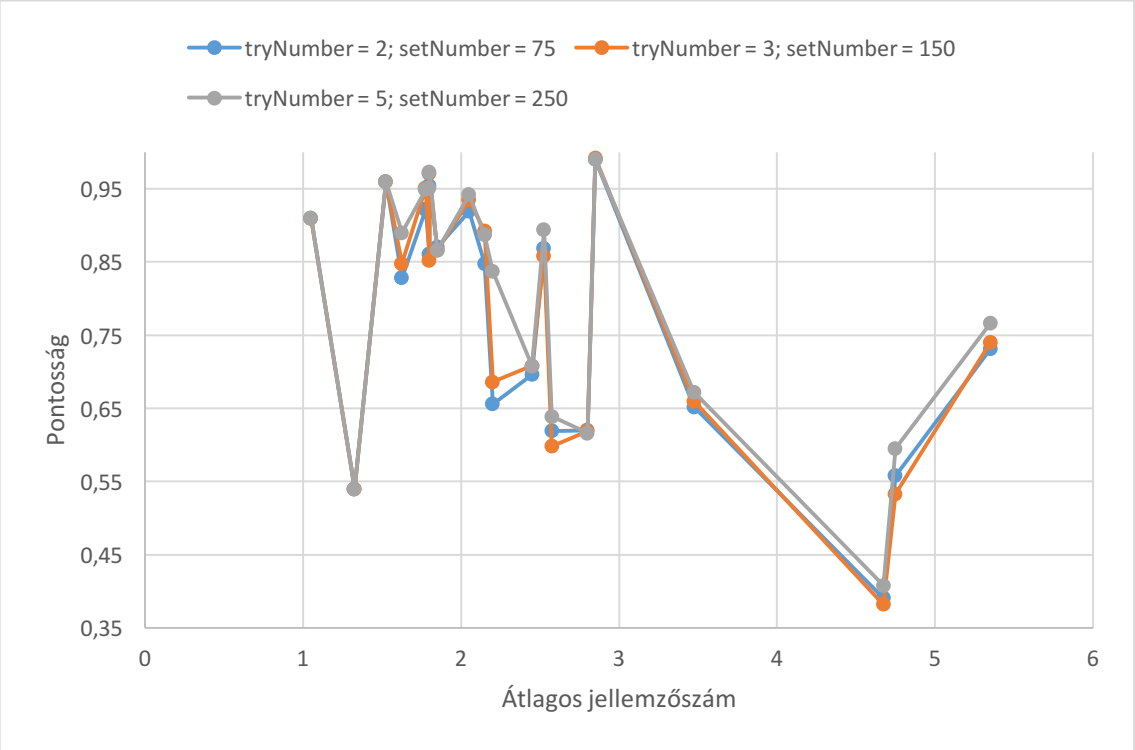
23. ábra: A karom-mentes párosítás futási ideje az átlagos jellemzőszám függvényében alapvonalak vizsgálata esetén

Tanulságos lehet megnézni, hogy Minty teljes algoritmusának futtatása mellett hogyan alakulnak a futási idők (*24. ábra*): ebben az esetben ugyanis nagyobb jellemzőszámnál akár 5 percig is tarthat a kiértékelés, ami már nyilván nem kényelmesen kivárható idő.



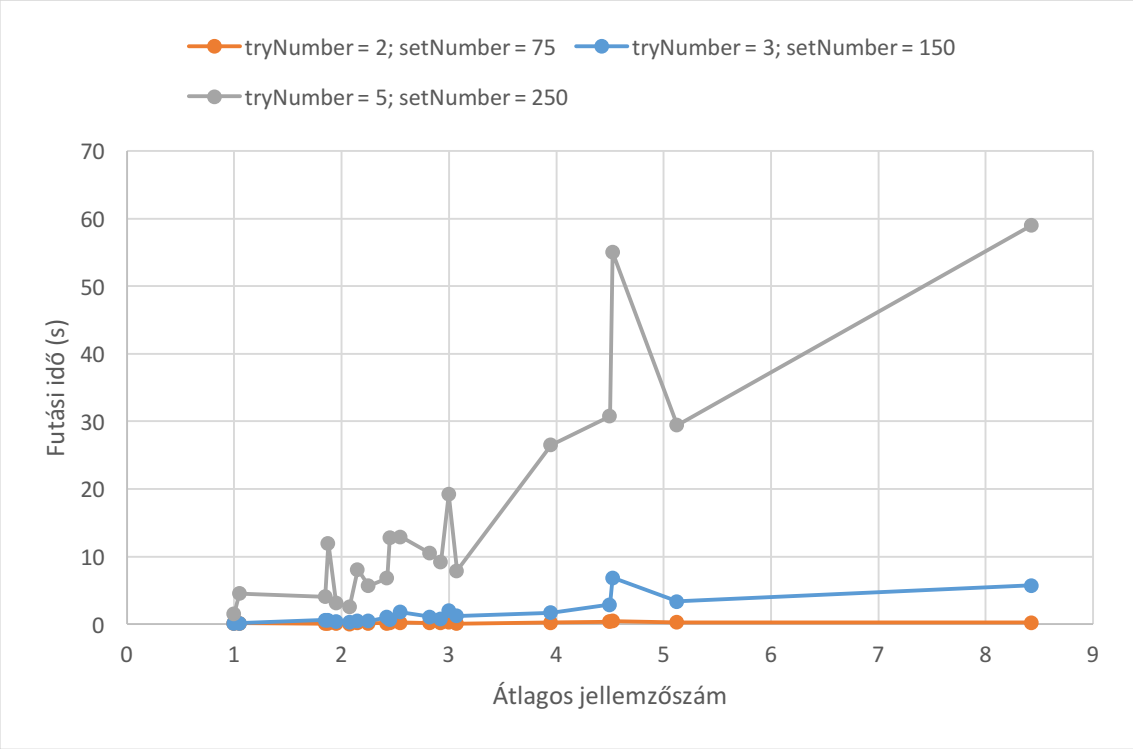
24. ábra: A karom-mentes párosítás futási ideje az átlagos jellemzőszám függvényében alapvonalak vizsgálata esetén

Pontosság szempontjából már olvashattuk az algoritmusok átlagos eredményét, megéri megnézni azonban, hogy ez hogy alakul jellemzőszám szerint (*25. ábra*).



25. ábra: A karom-mentes párosítás pontossága az átlagos jellemzőszám függvényében hurkok vizsgálata esetén

Ezen eredményeken érdemes lehet megfigyelni, hogy az algoritmus különböző paraméterek mellett is hasonló eredményeket ad ugyanazokra az aláírásokra (néha szinte meg se lehet különböztetni az eredményeket), de még ha kicsivel is, a legtöbb halmazt megvizsgáló algoritmus bizonyul a legpontosabbnak. Ugyanezeknek a paraméterezéseknek az időigényét megvizsgálva azonban úgy vélhetjük, nem feltétlen éri meg a legnagyobb halmazszámot választani a javulás érdekében, mivel az exponenciális jellege miatt jóval több időd igényel a számításokhoz (*26. ábra*).



26. ábra: A karom-mentes párosítás futási ideje az átlagos jellemzőszám függvényében alapvonalak vizsgálata esetén

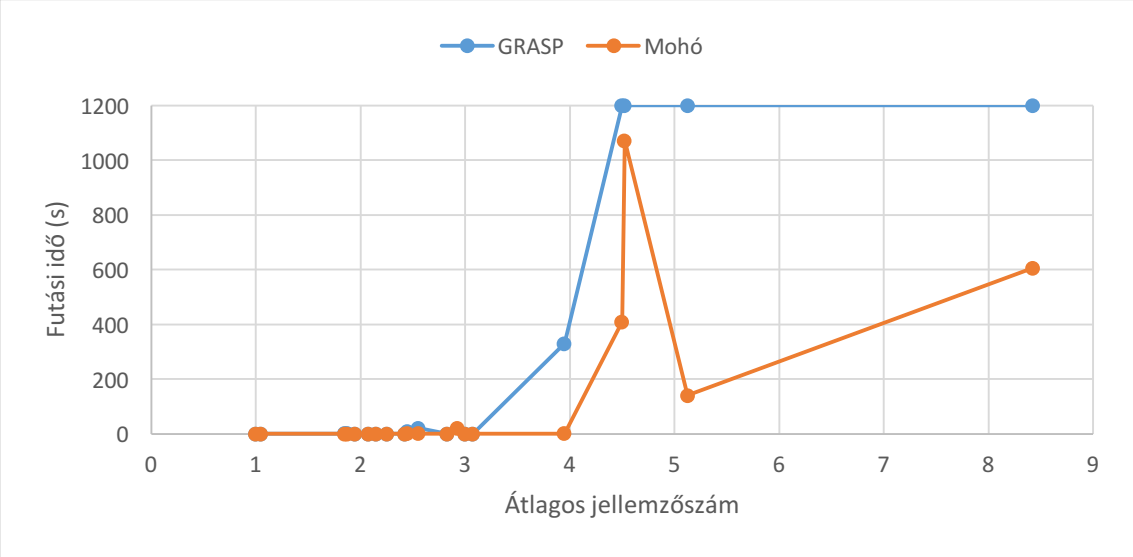
4 A megoldási módszerek értékelése

4.1 Az algoritmusok összehasonlítása

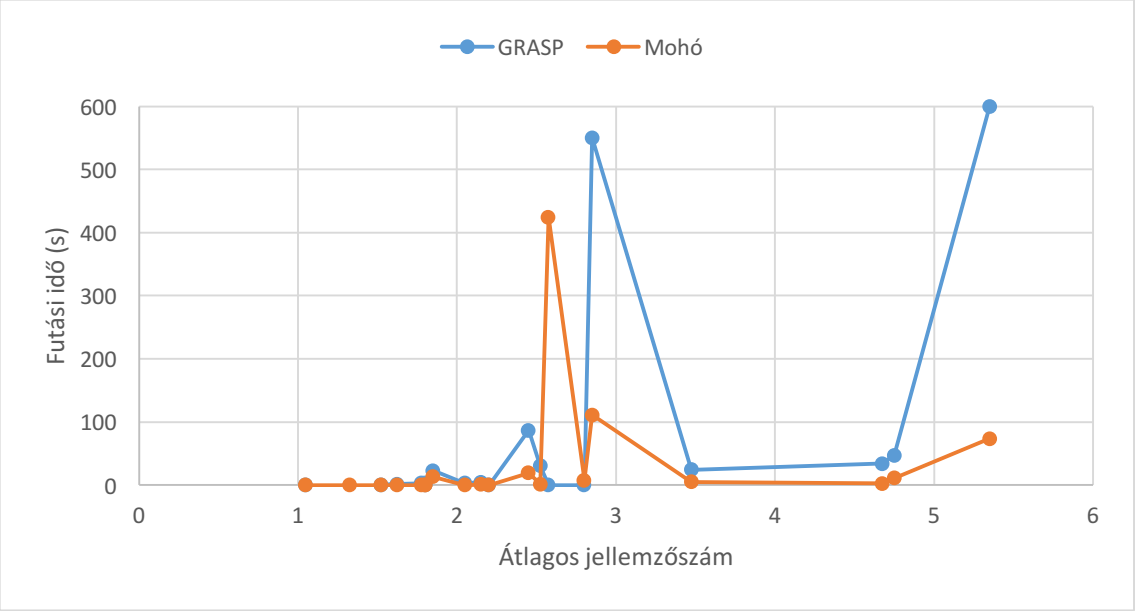
A harmadik fejezetben bemutatásra kerültek az egyes algoritmusok és azok méréséi eredményei, most pedig az egymáshoz képest elért futásidők és pontosságok ismertetése következik.

Korábban írtuk, hogy a véletlenszerű mohó keresés és a GRASP a többi algoritmussal ellentétben csak egy 20 aláírásból álló mintán lett futtatva, így nem lenne korrekt, ha a 40 darab szignatúrán párosító vetélytársaikkal hasonlítanánk össze. Először e kettő algoritmust vetjük össze.

A futásidők esetén azt várhatjuk, hogy a véletlenszerű mohó keresés lesz a gyorsabb, hiszen hiányoznak a további, lokális javítás lépései (*27. ábra, 28. ábra*). Ez így is történt, különösen a nagy jellemzőszámú (bemeneti élhalmazzal rendelkező) aláírásoknál, ahol a GRASP lefutását már ki sem tudtuk várni, a mohó keresés 10 perc alatt, vagy kicsivel afelett futott.

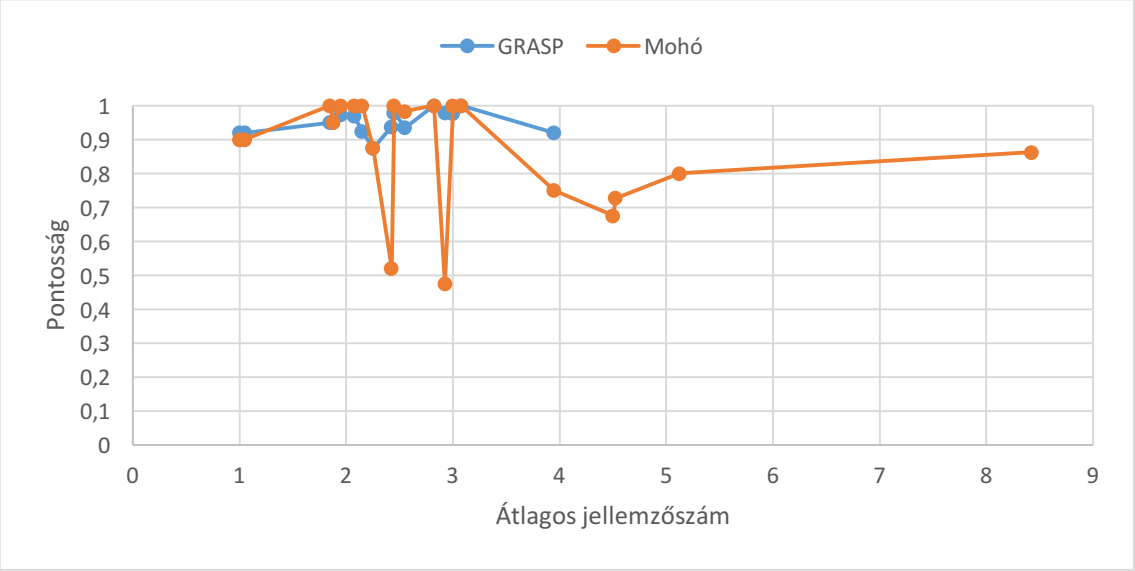


27. ábra: GRASP és mohó keresés futási idők alapvonalak esetén



28. ábra: GRASP és mohó keresés futási idők hurkok esetén

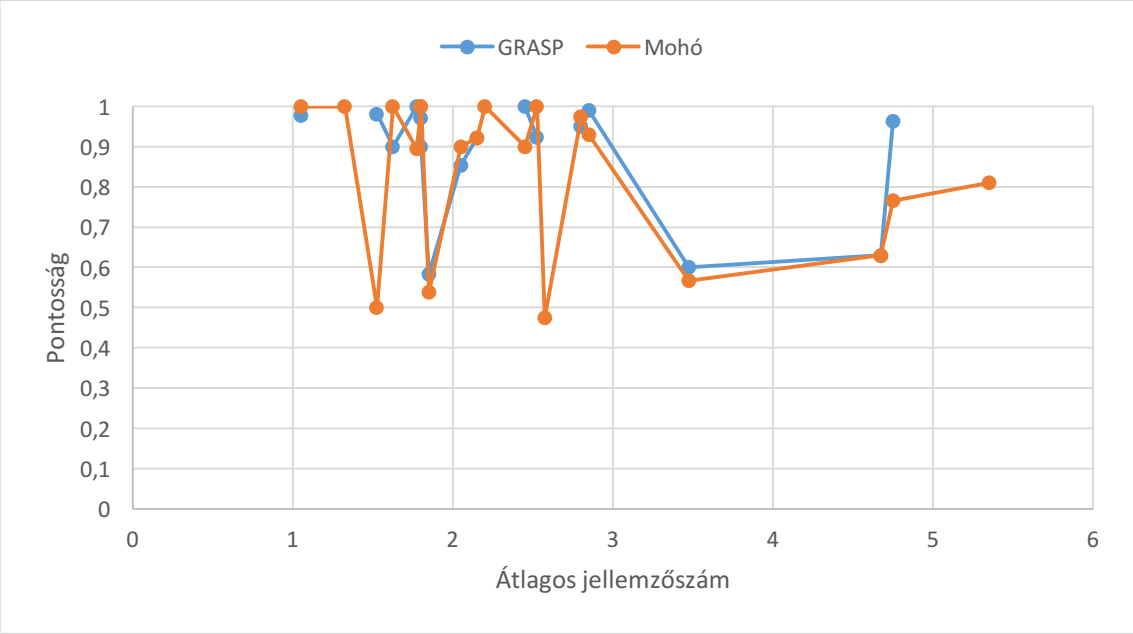
Kis jellemző számú aláírások esetén nem tapasztaltunk jelentős különbséget, mindkettő közel azonos idők alatt végezte el a párosítást. Az olyan esetekben, ahol a GRASP nem futott le időn belül, ott a grafikon automatikusan a legmagasabb értéket kapta meg. Látható sajnos a GRASP exponenciális lassulása a bonyolultabb aláírásoknál, és a lokális keresésből adódó lomhasága.



29. ábra: GRASP és mohó keresés pontossága alapvonalakon mérve

Ami a GRASP előnye a mohóval szemben az a pontosság (*29. ábra, 30. ábra*). Mind az alapvonalak, mind a hurkok esetén elmondható, hogy a mohóság miatti

ingadozásba stabilitást vitt a kimenetbe, viszont a rosszabb minőségű bemeneteknél a lokális keresés sem tudott nagyon sokat javítani. Több helyen, ahol a mohó 100%-ot ért el, ott az eltérő heurisztikából adódó feltételrendszer miatt a GRASP „csak” 95-99%-os pontosságot produkált.

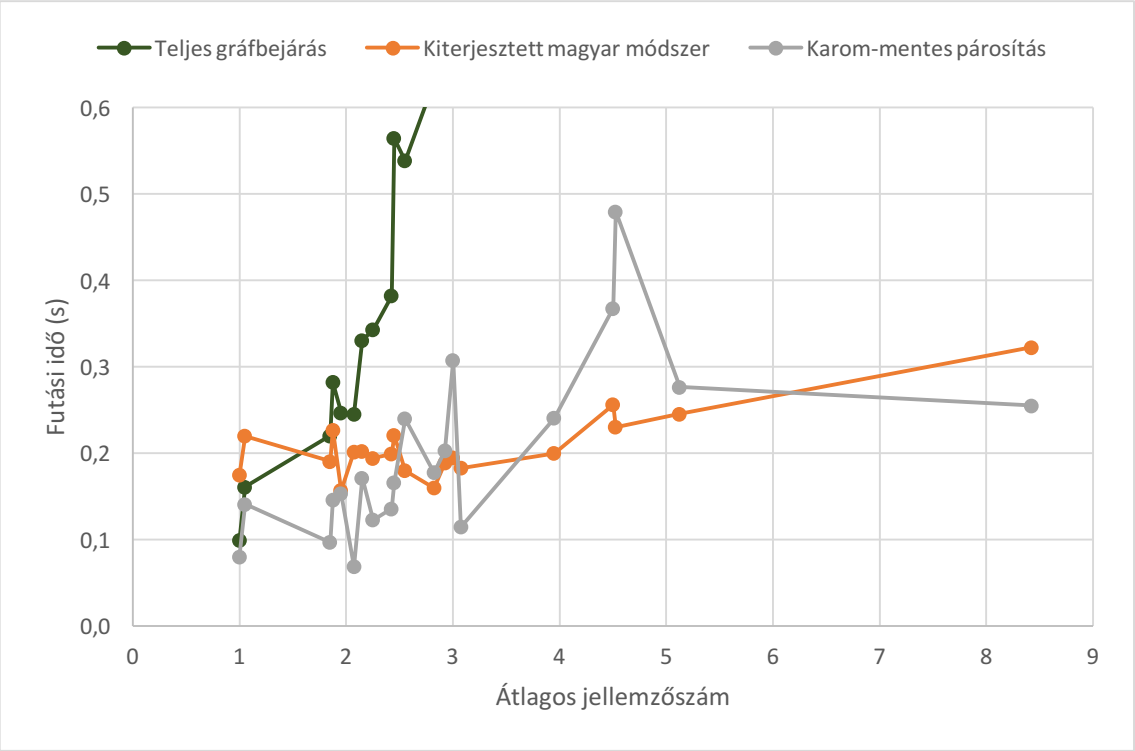


30. ábra: GRASP és mohó keresés pontossága hurkok esetén

Összességében a GRASP-nak sikerült a véletlenszerű mohó keresés átlagos pontosságán javítani (alapvonalak esetén 87,0975%-ról 91%-ra, hurkok esetén 84,04%-ról 90,2%-ra), azonban ennek az ára a futásidőben megmutatkozik.

A másik három algoritmusunkat azonban szerencsére könnyen össze tudjuk hasonlítani mind a 40 aláírásra számított pontosságuk és futási idejük alapján, és következtetéseket vonhatunk le mind a külön-külön mért értékek, mind az aggregált statisztikák alapján.

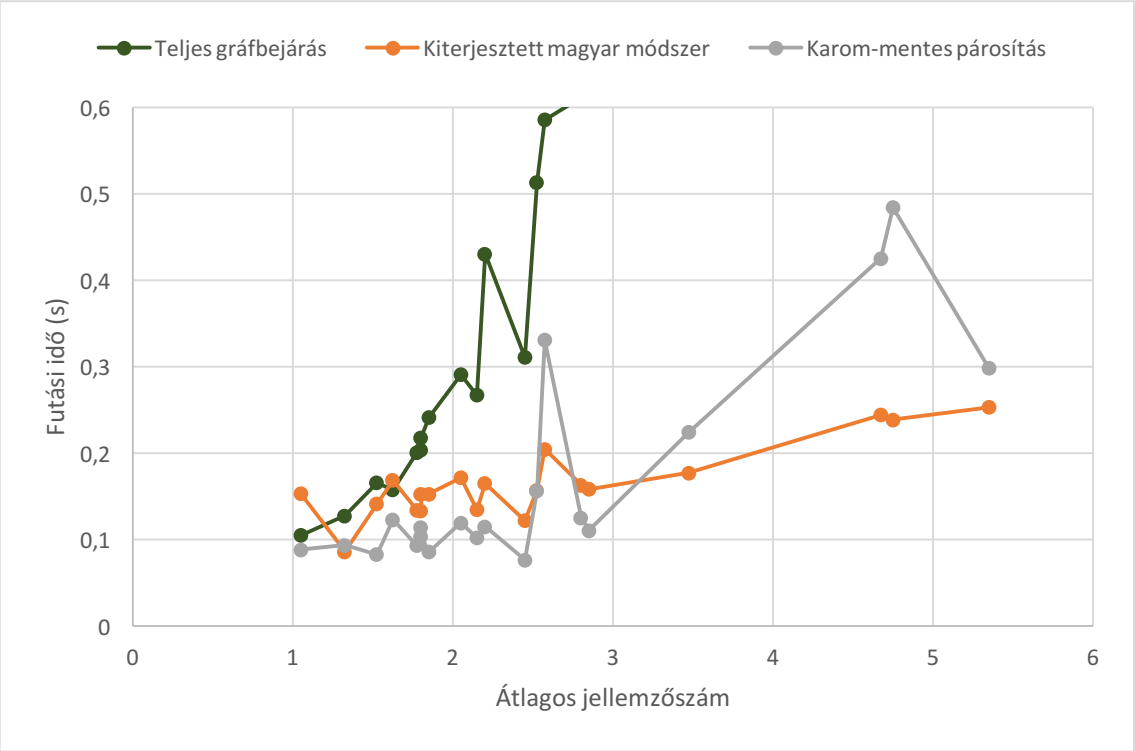
A megmaradt módszerek közül sajnos a teljes gráfbejárás minden tekintetben alulmúlja a többit: futási ideje (*31. ábra, 32. ábra*) jelentősen lassabb nagyon erős exponenciális jellege miatt, és pontossága is elmarad a várthoz képest.



31. ábra: Teljes gráfbejárás, kiterjesztett magyar módszer és karom-mentes párosítás futási ideje alapvonalak vizsgálata esetén

A másik két algoritmus igen váltakozóan teljesít, de összességében a karom-mentes párosítás mondható a legjobbnak: futási idő tekintetében mind az alapvonalak vizsgálata esetén, mind a hurkoknál megfigyelhető, hogy bár a kiterjesztett magyar módszer sokkal stabilabban tudja biztosítani az átlagos 0,184 másodperces futási idejét, a karom-mentes párosítás viszont alacsony jellemzőszám mellett gyorsabban adott eredményt, átlagosan 0,182 másodperc alatt végezve az aláírásokkal. Az eltérés persze egyáltalán nem jelentős, de mégis megfigyelhető (azzal pedig külön számoljunk, hogy utóbbi algoritmus esetén a paraméterek változtatásával továbbra is van lehetőségünk gyorsítani az futáson, ha az alkalmazás úgy kívánja meg), és a valódi alkalmazás során gyakrabban is fordulnak elő a kevés jellemzőből álló aláírások (ahogy azt az SVC20 adatbázis is megjeleníti).

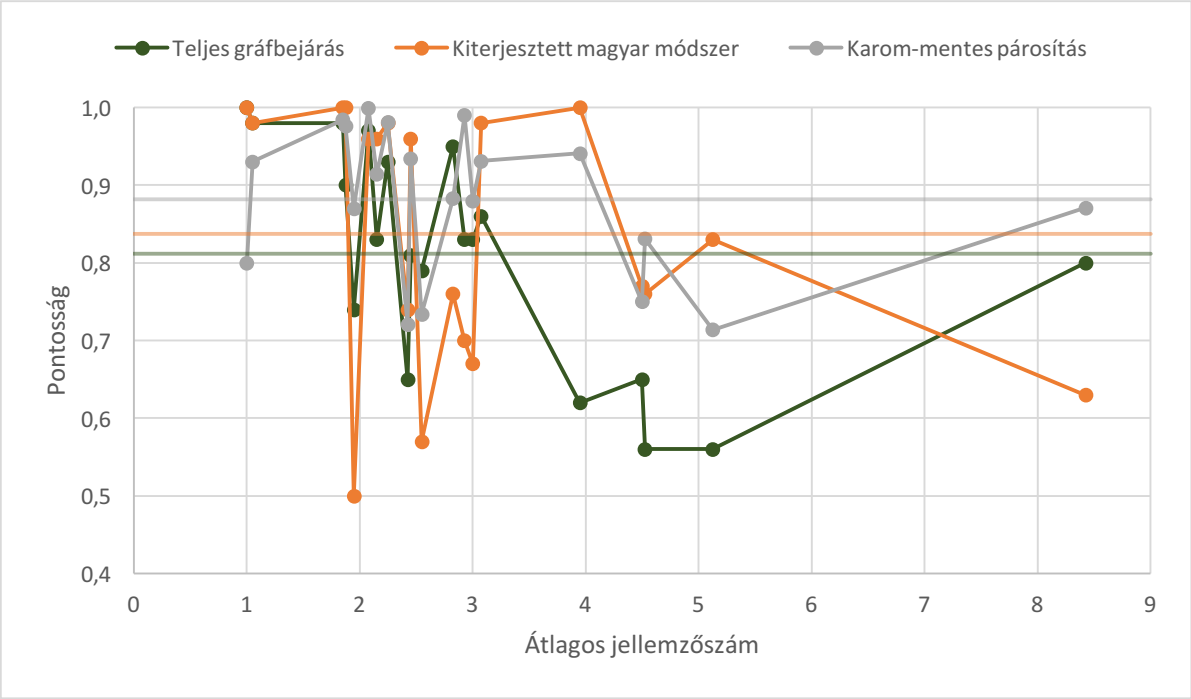
Az időbeli eltérés annak tulajdonítható, hogy a karom-mentes algoritmus külön figyelmet fordít arra, hogy alacsony jellemzőszám mellett kevesebb lehetséges hiperélt vizsgáljon meg, hisz valószínűleg sokkal kisebb halmazból is ki lehet választani egy optimális megoldást, míg a kiterjesztett magyar módszerbe ilyen szűrő nincs beépítve (az alapvetően teljes vizsgálatra épülő algoritmusba pedig nem is lehet).



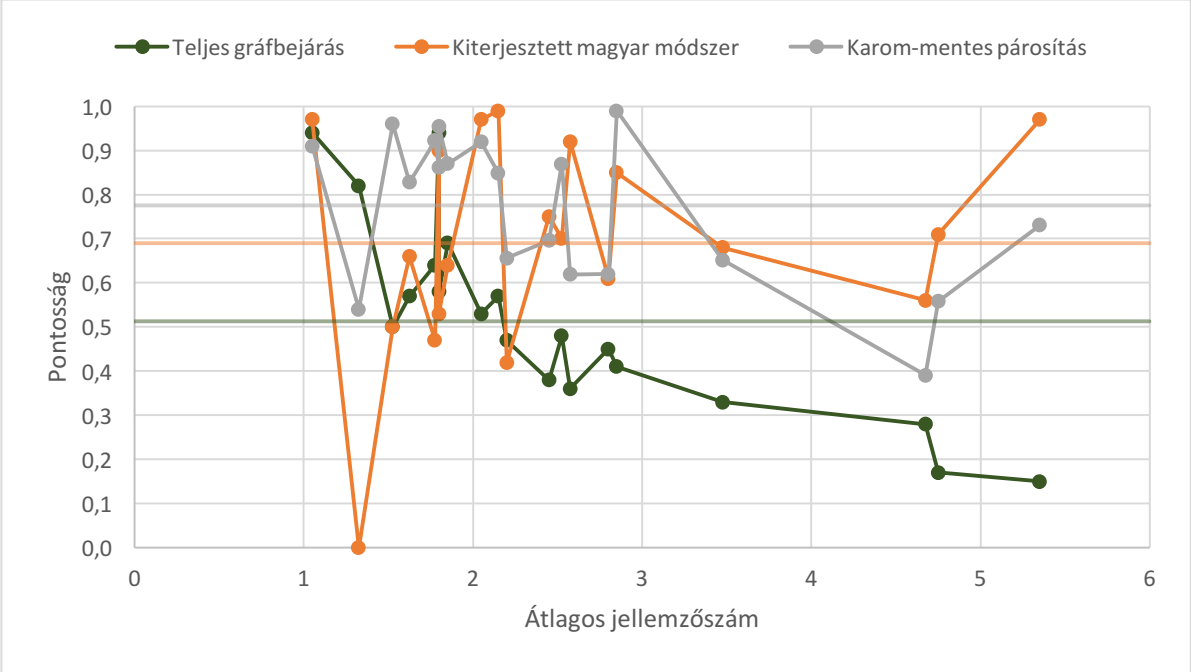
32. ábra: Teljes gráfbejárás, kiterjesztett magyar módszer és karom-mentes párosítás futási ideje hurkok vizsgálata esetén

Ami igazán fontos eredménynek mondható, az azonban nem a gyorsaság, hanem a pontosság növekedése: a kiterjesztett magyar módszer átlagos 76,37%-os eredményével (83,75% alapvonalakra és 69% hurkokra) szemben a karom-mentes párosítás 88,2%-os pontosságot (88,2% alapvonalakra és 77,5% hurkokra) ért el, mindezt az általunk vizsgált leggyorsabb futási idejű paraméterek mellett (*33. ábra, 34. ábra)*.

Pontosság tekintetében az eddigiektől eltérően a karom-mentes párosítás az, amelyik kisebb ingadozásokat mutat, míg a kiterjesztett magyar módszer kisebb jellemzőszámnál szélsőséges értékeket is képes mutatni: van olyan eset, ami mellett egyáltalán nem találja meg a keresendő jellemzőpárt.



33. ábra: Teljes gráfbejárás, kiterjesztett magyar módszer és karom-mentes párosítás pontossága alapvonalak vizsgálata esetén



34. ábra: Teljes gráfbejárás, kiterjesztett magyar módszer és karom-mentes párosítás pontossága hurkok vizsgálata esetén

5 Összefoglalás és kitekintés

A dolgozatunk megírása során megtapasztaltuk, hogy a jellemzőpárosítási probléma megoldása milyen nehézségekkel jár, és jó pár NP-nehéz algoritmus implementálásával és közelítésével próbálkoztunk. A módszerek kipróbálása és összehasonlítása pedig támpontot adhat a jövőbeli kutatások számára, hogy milyen esetleges új szemszögekből érdemes vizsgálni a problémát.

Egyik legfontosabb tapasztalat a teljes gráfbejárás algoritmus gyenge működése: a vizsgált módszerek közül ez volt az, amelyik a legegyszerűbb heurisztikákkal próbálta megközelíteni a problémát, nem pedig komplex optimalizációs feladatként. A lokális döntések és lokális útkeresések választása a bonyolultabb optimalizáció helyett azonban egyértelműen téves útnak bizonyult, és tanulságul szolgálhat későbbi ilyen kezdeményezésekhez.

Fontos észrevétel a karom-mentes algoritmusban Minty munkájának egyszerűsítése, mely a gráf speciális felépítését használja fel arra, hogy a módszer gyorsabban működjön: bár matematikailag az eredeti algoritmus egy lényeges részét vesszük ki, így még az adott iterációban tesztelt halmazokra se lesz garantáltan optimális az eredmény, mégis, a jelentősen lecsökkent futási idő nélkül elképzelhetetlen lenne az algoritmus működése. Ehhez hasonló trükköket pedig könnyen lehet, hogy más megoldásokban is fel lehet használni majd a jövőben, a létrejövő súlyozott gráfok egy áthatóbb vizsgálatának eredményei alapján.

A GRASP és mohó keresés esetén láthattuk, hogy a pontosság növelése a futási idő exponenciális megugrásával járhat, így mindenképp el kell gondolkodnunk, hogy a tapasztalt, átlagban néhány százalékos javítás megéri e plusz időt, hiszen egy, az algoritmus számára ideális bemeneti halmaz esetén is remekül működik a mohó keresés, ha azonban a nagyobb pontosság a cél, akkor a GRASP az egyértelmű választás.

Dolgozatunk írása során a BME VIK AUT tanszékének aláírás hitelesítő projectjét tudtuk két új párosítási algoritmussal bővíteni, melyek közül az egyik jobb eredményeket ért el, mint a korábban rendelkezésre álló megoldások. Ez önmagában is komoly gyakorlati haszonnal fog rendelkezni a project további élete során, míg összehasonlító és összefoglaló munkánk a jövőbeli kutatásokat segítheti.

5.1 Tervek

A terület természetesen még távol áll a lezárástól, és rengeteg fejlődési lehetőség van a mi algoritmusaink számára is.

Első körben mindenképp szeretnénk utánajárni a grafikonokon található adatok eredetének: mi okozza a kiugró értékeket a különböző esetekben, és miért szerepelnek egyes algoritmusok jobban néhány aláírásra, mint mások. Lehet-e a különböző algoritmusok egyes részeit összekombinálni, és egy egységes algoritmusban szerepeltetni?

Azoknál az algoritmusoknál, amik először egy halmazt hoznak létre, majd abból választják ki az optimális megoldást, a lehető legnagyobb hangsúlyt kell fektetni az időre is: hisz például a karom-mentes párosításban minél gyorsabban történik a gráf felépítése, és az abban történő legnagyobb súly független ponthalmaz keresése (például naprakész algoritmusok implementálásával [20] [19]), annál több halmazt tudunk megvizsgálni is, így a pontosság is javul.

A GRASP és mohó keresés kapcsán egyértelműen kiderült, hogy az előbbi algoritmus futási idejében bőven van még mit javítani. A jövőben meg fogjuk vizsgálni a gyorsítási, ezen belül a többszálúsítási lehetőségeket. Tekintve, hogy mindkettő algoritmus egy előre generált bemeneti halmazon dolgozik, így milliós nagyságrendnél már jelentősen lelassulhat ez is, így itt is újabb heurisztikák bevezetésére van szükség.

6 Irodalomjegyzék

[1] F. Bryan és R. Doug, „Documentation of Forensic Handwriting Comparison and Identification Method: A Modular Approach,” *Journal of Forensic Document Examination,* %1. kötet12, pp. 1-68, 1999.

[2] Z. Berceli, „Írásjellemzők automatikus párosítása az aláíráshitelesítésben (Automated feature matching in signature verification),” *Master's thesis,* 2012.

[3] G. Gregory, G. Boris és H. Jing, „Worst Case Analysis of Max-Regret, Greedy and Other Heuristics for Multidimensional Assignment and Traveling Salesman Problems,” *Journal of Heuristics,* %1. kötet14, %1. szám2, pp. 169-181, April 2008.

[4] M. R. Garey és D. S. Johnson, Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., 1990.

[5] T. Fleiner, „A számítástudomány alapjai,” 2014..

[6] L. Lovász és M. D. Plummer, „Matching Theory,” North-Holland, Amsterdam, 1986.

[7] M. Mucha és P. Sankowski, „Maximum Matchings via Gaussian Elimination,” *Symposium on Foundations of Computer Science,* p. 248–255, 2004.

[8] H. W. Kuhn, „The Hungarian Method for the assignment problem,” *Naval Research Logistics,* pp. 83-97, 1955.

[9] M. L. Fredman és R. E. Tarjan, „Fibonacci heaps and their uses in improved network optimization algorithms,” *Journal of the ACM,* p. 596–615, 1987.

[10] Z. Füredi, „Matchings and covers in hypergraphs,” in *Graphs and Combinatorics volume 4*, 1988, p. 115–206.

[11] J. M. Robson, „Algorithms for maximum independent sets,” *Journal of Algorithms,* p. 425–440, 1986.

[12] P. Crescenzi, „A Short Guide To Approximation Preserving Reductions,” *Proceedings of the 12th Annual IEEE Conference on Computational Complexity,* 1997.

[13] D. Lokshtanov, M. Vatshelle és Y. Villanger, „Independent sets in P5-free graphs in polynomial time,” *Symposium on Discrete Algorithms,* 2014.

[14] M. Grötschel, L. Lovász és A. Schrijver, „Coloring Perfect Graphs,” in *Algorithms and Combinatorics*, 1998, p. 296–298.

[15] G. J. Minty, „On maximal independent sets of vertices in claw-free graphs,” *Journal of Combinatorial Theory,* 1980.

[16] A. Frank, „Some polynomial algorithms for certain graphs and hypergraphs,” *Congressus Numerantium,* 1976.

[17] N. Sbihi, „Algorithme de recherche d'un stable de cardinalité maximum dans un graphe sans étoile,” *Discrete Mathematics,* 1980.

[18] J. Edmonds, „Paths, trees, and flowers,” *Canadian Journal of Mathematics,* 1965.

[19] D. Nakamura és A. Tamura, „A revision of Minty's algorithm for finding a maximum weighted stable set of a claw-free graph,” *Journal of the Operations Research Society of Japan,* 2001.

[20] Y. Faenza, G. Oriolo és G. Stauffer, „An algorithmic decomposition of claw-free graphs leading to an O(n3)-algorithm for the weighted stable set problem,” *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms,* 2011.

[21] P. Nobili és A. Sassano, „An O(n2 log(n)) algorithm for the weighted stable set,” 2015.

[22] „SVC 2004: First International Signature Verification Competition,” 2003. [Online]. Available: http://www.cse.ust.hk/svc2004/.

[23] C. Tóth, „Párosítási algoritmusok vizsgálata az aláírás-hitelesítésben,” 2015.

[24] B. Dr. Kővári, „Models and algorithms in off-line, feature-based, handwritten signature verification,” 2013.

[25] Z. Albert, „Aláíráshitelesítés Alapvonalak DTW Alapú Összehasonlításával (Signature verification using DTW based comparison of lower envelopes),” *Scientific Students' Associations Conference,* 2009.

[26] F. Bryan és R. Doug, „Documentation of Forensic Handwriting Comparison and Identification Method: A Modular Approach,” *Journal of Forensic Document Examination,* pp. 1-68, 1999.

7 Függelék

GRASP és mohó keresés esetén keletkező bemeneti halmazok mérete az egyes aláírások esetén:

	Alapvonalak	Hurkok
002	10	94620
004	124	33
006	128	1261
008	261	402
010	2291	2432797
013	10	85333
015	245509	508192
018	1028443	302338
020	57	19
022	182688	779
024	3071127	1376
025	2299	43152
028	29099	3394
032	58	1558
033	57605	9088
034	262	null set
035	185	12485
037	165	29816
038	645	6506
040	264	49519