Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aleš Smodiš

Strategije za uravnoteženo izbiro retrospektivnih podatkov za simulacijo prospektivnih raziskav

diplomsko delo na univerzitetnem študiju

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana, 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Avtomatizirano računalniško uravnoteževanje kohort bolnikov je koristno, ker omogoča natančnejšo in predvsem nepristransko primerjavo izidov zdravljenja. Računalnik ima pred človekom pri tej nalogi dve poglavitni prednosti: (a) zaradi hitrosti je sposoben najti čimbolj uravnotežene kohorte po več značilnostih hkrati in (b) je nepristranski, kar zagotovi korektnost izbire.

Kandidat naj robustno implementira naš obstoječi algoritem za uravnoteženo izbiro ter uparjanje bolnikov iz retrospektivnih podatkov. Obenem naj razišče nekaj že predlaganih idej kot tudi lastne ideje za izboljšavo kvalitete izbire bolnikov – tako v smislu podobnosti izbranih bolnikov kot v smislu velikosti množic izbranih bolnikov. Kandidat naj primerja izvirni algoritem s svojimi izboljšavami in poda mnenje o najbolj primernem algoritmu za realno uporabo v medicinskih in drugih raziskavah. Testiranje naj kandidat opravi na realnih medicinskih podatkih s področja onkologije.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Aleš Smodiš sem avtor diplomskega dela z naslovom:

Strategije za uravnoteženo izbiro retrospektivnih podatkov za simulacijo prospektivnih raziskav

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom akad. prof. dr. Ivana Bratka,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 22. decembra 2015 Podpis avtorja:

Zahvaljujem se mentorju akad. prof. dr. Ivanu Bratku za pomoč pri izdelavi diplomskega dela, viš. pred. dr. Aleksandru Sadikovu za izvorni algoritem in potrpežljivo razlaganje pojmov ter razblinjanje skrbi, ter Timoteju Lazarju za idejno pomoč pri sestavljanju nove mere. Prav tako se zahvaljujem Živi za potrpežljivost ter Ediju in Marku za izkazano razumevanje pri mojem vztrajanju v odsotnosti.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
uPA	urokinase plasminogen activator	urokinazni aktivator plazminogena
PAI-1	plasminogen activator inhibitor-1	zaviralec aktivatorja plazminogena 1
CMF	cyclophosphamide, methotrexate, fluorouracil	ciklofosfamid, metotreksat, fluorouracil

Povzetek

Naslov: Strategije za uravnoteženo izbiro retrospektivnih podatkov za simulacijo prospektivnih raziskav

Porast raziskav v medicini poraja vedno več ugotovitev, ki imajo lahko za posledico nova ali izboljšana obstoječa zdravljenja. Obenem pa potrebe po novih raziskavah pripeljejo do težav pri zagotavljanju zadostnega števila bolnikov za prospektivne raziskave vseh obetajočih zdravljenj. Po drugi strani lahko z restrospektivno raziskavo na obstoječih podatkih bolnikov do določene mere simuliramo prospektivno raziskavo. Glavna težava pri tem pristopu je, da imajo obstoječi podatki običajno neuravnotežene porazdelitve karakteristik po množicah bolnikov, na katerih izvajamo retrospektivno raziskavo, kar oteži vrednotenje učinkov zdravljenja. Predstavljen je algoritem za uravnoteževanje množic bolnikov z danimi karakteristikami, ki z uparjanjem in z izločanjem izbranih bolnikov ustvari uravnotežene podmnožice bolnikov. Algoritem uporablja Pearsonov test χ^2 za merjenje kvalitete medsebojne uravnoteženosti množic in vsoto uteženih razlik vrednosti karakteristik za določanje parov elementov med dvema množicama. Uvedeni sta dve novi strategiji uparjanja elementov: s požrešno metodo preko matrike podobnosti parov, ter z algoritmom minimin na drevesu stanj do predpisane globine za izbiro naslednjih dveh elementov za uparjanje. Uvedena je mera kvalitete uparjenosti med dvema množicama. Rezultati kažejo, da požrešna metoda daje boljše rezultate od izvirnega algoritma, medtem ko se algoritem minimin izkaže za časovno zahtevnega zaradi kombinatorične zahtevnosti in pri globinah, ki so glede tega še praktične za izvajanje algoritma, daje kvečjemu

primerljive rezultate izvirnemu algoritmu, vendar slabše od požrešne metode. Metode so bile eksperimentalno primerjane na realnih podatkih iz medicinskih raziskav zdravljenja raka.

Ključne besede: retrospektivne raziskave, simulacija prospektivnih raziskav, uparjanje, uravnoteževanje množic, hevristično preiskovanje, hevristična ocena kvalitete uravnoteženosti, Pearsonov test χ^2.

Abstract

Title: Strategies for Balanced Selection from Retrospective Data for Simulation of Prospective Studies

The increase of medical research generates more and more findings which can result in new or enhanced existing treatments. This increase of medical research leads to problems at ensuring a sufficent number of patients for prospective studies of all of the promising treatments. On the other hand a prospective study can be simulated to a certain degree with a retrospective study using existing data. The main problem with this approach is, that the existing data usually have unbalanced distributions of characteristics over the sets of patients, which makes it difficult to evaluate effects of treatment. An algorithm is described for balancing sets of patients with given characteristics, which creates balanced subsets of patients using pairing and elimination of selected patients. The algorithm uses Pearson's χ^2 test for measuring the balance quality between two sets, and the sum of weighed differences between the characteristics for defining element pairs between sets. Two new element pairing strategies are introduced: a greedy method using an element similarity matrix, and the minimin algorithm using a state tree with limited depth for choosing the next elements to pair. A measure for the quality of a match between two sets is introduced. Results show that the greedy method gives better results from the original algorithm, whereas the minimin algorithm turns out to be time demanding because of the combinatorial complexity. At depths at which the algorithm is still practical to use, it gives results at best comparable to the original algorithm, but worse

than the greedy method. The methods were experimentally compared on real data from medical studies in cancer treatment.

Keywords: retrospective studies, simulated prospective studies, pairing, data set balancing, heuristic search, heuristic evaluation of balance quality, Pearson's χ^2 test.

Poglavje 1

Uvod

Z vse hitrejšim razvojem v medicini prihaja do vedno več odkritij glede zdravljenj bolezni. Zaradi tega vse pogosteje prihaja do situacij, ko ni mogoče zagotoviti skupin bolnikov za preizkuse vseh novih potencialno uspešnih zdravljenj, saj preprosto ni na voljo takega števila bolnikov. Nova zdravljenja pa morajo skozi vrsto preizkusov, kjer se praviloma pri končnem preizkusu vrši prospektivna randomizirana klinična študija, ki potrebuje več sto ali celo več tisoč bolnikov.

Ker je bolnikov premalo, da bi lahko izvedli preizkuse vseh potencialno uporabnih zdravljenj, je potrebno za klinične študije izbrati samo najbolj obetavna zdravljenja. To lahko naredimo z retrospektivnimi študijami na obstoječih podatkih bolnikov, nastalih skozi prejšnja zdravljenja. Z retrospektivno študijo testiramo hipotezo na obstoječih podatkih. Rezultate take študije ne moremo enačiti z rezultati, ki bi jih dobili s prospektivno študijo, saj so retrospektivne študije dovzetne za razne pristranskosti. Pri prospektivnih študijah se večini pristranskosti izognemo z randomiziranimi nadzorovanimi študijami, kjer bolnike randomiziramo in izvajamo preizkuse na slepo. Randomiziranje zagotavlja, da so tako znani kot neznani dejavniki enakomerno porazdeljeni med skupinami, z izvajanjem na slepo pa se ščitimo pred placebo-efektom in pred pristranskostjo izvajalca [1].

Testiranje hipoteze pri retrospektivni študiji poteka na sledeči način. Naj-

prej se določi ena ali več karakteristik na obstoječih podatkih bolnikov, po katerih razbijemo množico na več podmnožic. Tipične karakteristike bolnikov so na primer: prejeta zdravila, spol, starost, odzivi na zdravljenja, biološki markerji. Nato pa gledamo določeno karakteristiko podmnožic bolnikov, ki nas zanima za testiranje hipoteze, tipično je to preživetje oz. ozdravitev bolnikov. V najbolj preprosti obliki retrospektivne študije nas zanima, če je možnost preživetja bolnikov večja vkolikor jemljejo določeno zdravilo, kot pa da ga ne jemljejo. To pomeni, da najprej razdelimo bolnike na dve podmnožici, podmnožico bolnikov, ki so dobili zdravilo, ter podmnožico bolnikov, ki niso dobili zdravila. Z danim zdravilom smo izboljšali možnost preživetja bolnikov, vkolikor podmnožica bolnikov, ki je dobila zdravilo, kaže statistično značilno večje preživetje. Tako potrjene hipoteze so samo predlog, še vedno jih je potrebno potrditi s prospektivnimi študijami. Rezultati retrospektivnih študij so zato uporabni za manjšanje števila študij, ki naj se izvedejo na prospektivni način, ali njihovo prioritiziranje.

Statistični testi so najbolj zanesljivi na uravnoteženih podatkih [1]. To pomeni, da bolj ko so si bolniki med podmnožicami podobni, manj nas skrbi vpliv razlik med bolniki na rezultat študije in bolj zaupamo rezultatu študije. Vendar se pri retrospektivnih študijah običajno ukvarjamo s podmnožicami s precej neenakomerno porazdelitvijo karakteristik bolnikov. Zato je potrebno podmnožice bolnikov najprej uravnotežiti. Ena od metod za uravnoteževanje podatkov je multivariatno uparjanje [2]. Metoda deluje tako, da med bolniki dveh množic izbira pare po karakteristikah podobnih bolnikov, po enega iz vsake množice, ter iz njih sestavi podmnožici originalnih dveh množic. Nastali podmnožici sta manjši, vendar uravnoteženi po karakteristikah bolnikov.

Na zelo neuravnoteženih podatkih multivariatno uparjanje dosega slabe rezultate, zato smo za izhodišče uporabili algoritem A. Sadikova [3], ki poleg opisanega uparjanja uporablja še izločanje bolnikov s požrešno metodo in s posebej za to zasnovano hevristično oceno. Ideja je, da po opisani uparitvi bolnikov iz dveh množic, novonastalim podmnožicam izmerimo njuno medsebojno uravnoteženost in pričnemo izločati bolnike, ki uravnoteženost najbolj

kvarijo. To počnemo, dokler podmnožici ne postaneta uravnoteženi.

Podobnost dveh bolnikov je pojem, ki ga je mogoče določiti na različne načine, odvisno od tega kako strogi želimo biti. To neposredno vpliva tudi na kvaliteto rezultata, saj z manj strogim določilom podobnosti v splošnem dosežemo manj uravnotežen rezultat. Običajno imamo določen nek kriterij za to, kdaj sta dve množici (še) uravnoteženi in želimo pregledati rezultate uravnoteževanja pri različno določenih podobnostih, da dobimo občutek, kako vplivajo na uravnoteženost. En tak primer je, če želimo dobiti čim večje množice bolnikov, ki so še uravnotežene. Iz same definicije podobnosti namreč ne moremo neposredno sklepati na kvaliteto rezultata, potrebno je izvesti uravnoteževanje in izmeriti uravnoteženost. To pomeni precej računskega dela nad kopico podatkov, zato je v veliko pomoč ustrezen program za uravnoteževanje.

Poglavje 2

Metodologija

Za izhodišče smo uporabili algoritem A. Sadikova za uravnoteževanje [3] ter meri za kvaliteto uravnoteženosti dveh množic in za kvaliteto uparjenosti oz. podobnosti dveh elementov, ki ju uporablja. V nadaljevanju bomo ta algoritem imenovali ,,izvirni algoritem“. Ustvarili smo dodatne strategije za uravnoteževanje in dodatno mero, ki meri kvaliteto uparjenosti dveh množic.

2.1 Mere

Podobnost dveh množic skušamo zagotoviti z uravnoteženo porazdelitvijo vrednosti določenih karakteristik med množicama elementov. Tem karakteristikam pravimo uravnoteževalne karakteristike. Izbere jih strokovnjak in z njimi določi mero podobnosti med dvema elementoma. Za mero uravnoteženosti med dvema množicama elementov uporabimo Pearsonov statistični test χ^2 po frekvencah vrednosti med množicama za vsako uravnoteževalno karakteristiko posebej.

2.1.1 Podobnost dveh elementov

Za dva elementa (bolnika) pravimo, da sta si podobna, ko je mera podobnosti med njima znotraj določenega praga. Vrednost mere podaja kvaliteto uparitve obeh elementov. Bolj sta si elementa podobna, boljša je kvaliteta uparitve

in manjša je ta vrednost. Vrednost 0 pomeni, da sta elementa enaka. Tako mero kot prag določi strokovnjak, oblika mere pa je tipično uravnotežena vsota razlik med vrednostmi posamičnih karakteristik:

$$s(a,b) = \sum_i k_i \left| a_i - b_i \right| \tag{2.1}$$

kjer je s mera podobnosti, i je indeks uravnotežitvene karakteristike, k_i je koeficient karakteristike i, a_i in b_i pa sta vrednosti karakteristike i elementov a in b. Vkolikor vrednosti karakteristik niso podane numerično, jih moramo najprej pretvoriti v numerično obliko, kar lahko najpreprosteje storimo z oblikovanjem zaporednih številk vrednosti. Elementa a in b sta si podobna, ko njuna podobnost ne preseže praga:

$$s(a,b) \leq M \tag{2.2}$$

kjer je M prag, do kjer še trdimo, da sta si elementa podobna.

Lahko se tudi predpišejo pragovi za vsako karakteristiko posebej, kjer razlika med vrednostima dane karakteristike ne sme preseči njenega praga. Tako podane omejitve prevedemo v obliko v enačbi 2.1.

2.1.2 Uravnoteženost dveh množic

Za računanje mere uravnoteženosti med dvema množicama uporabimo Pearsonov statistični test χ^2 [4]. Računa se posebej za vsako uravnotežitveno karakteristiko iz frekvenc njenih vrednosti med dvema množicama:

$$\chi^2 = \sum_i \sum_j \frac{\left(F^o_{i,j} - F^e_{i,j}\right)^2}{F^e_{i,j}}; \qquad F^e_{i,j} = \frac{F_i F_j}{N} \tag{2.3}$$

kjer je i indeks množice, j je indeks vrednosti za dano karakteristiko, $F^o_{i,j}$ je frekvenca oz. število elementov iz množice i z j-to vrednostjo, F_i je vsota frekvenc iz množice i, F_j je vsota frekvenc za j-to vrednost, N pa je število vseh elementov.

Za vsako karakteristiko se predpiše mejna p-vrednost, to je p-vrednost, ki je še dopustna, da se množici smatrata za uravnoteženi. Na podlagi dane

mejne p-vrednosti in prostostne stopnje karakteristike se določi mejna vrednost χ^2. Če so dejanske vrednosti χ^2 vseh karakteristik pod njihovimi mejnimi vrednostmi, se smatra da sta si množici podobni in s tem uravnoteženi.

Računanje te mere smo dopolnili z Yatesovo korekcijo [5]. Standardno se uporablja pri matrikah velikost 2×2, kjer vsaj ena izmed frekvenc pade pod 5. V našem primeru to pomeni, da uporabimo Yatesovo korekcijo v primeru, ko imamo karakteristiko z dvema možnima vrednostima (natančneje, s prostostno stopnjo 1), kjer frekvenca vsaj ene vrednosti v eni izmed množic pade pod 5.

2.1.3 Kvaliteta uparjenosti dveh množic

Poleg merjenja uravnoteženosti dveh množic smo v rezultatih želeli meriti tudi kvaliteto uparjenosti elementov. Ker algoritmi, ki so opisani v nadaljevanju, v splošnem uravnotežijo množice tako, da niso enakih velikosti, se nismo mogli poslužiti povprečja podobnosti elementov med množicama. Sestavili smo novo mero kvalitete uparjenosti dveh množic u.

Najprej definiramo minimalno podobnost danega elementa a iz množice $\mathbf{A}$ z množico $\mathbf{B}$ kot najmanjšo podobnost elementa a s katerimkoli elementom iz množice $\mathbf{B}$ z uporabo funkcije s (enačba 2.1):

$$s_m(a, \mathbf{B}) = \min_{b \in \mathbf{B}} (s(a, b)) \tag{2.4}$$

Kvaliteto uparjenosti množic $\mathbf{A}$ in $\mathbf{B}$ definiramo kot povprečno najmanjšo podobnost elementov obeh množic:

$$u(\mathbf{A}, \mathbf{B}) = \frac{\sum_{a \in \mathbf{A}} s_m(a, \mathbf{B}) + \sum_{b \in \mathbf{B}} s_m(b, \mathbf{A})}{|\mathbf{A}| + |\mathbf{B}|} \tag{2.5}$$

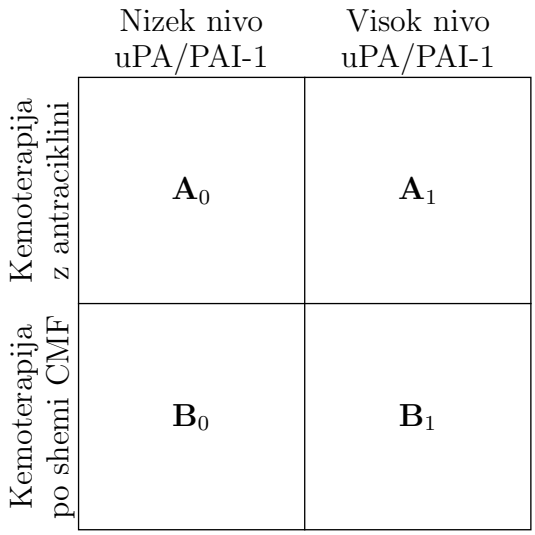
2.2 Izvirni algoritem

Izvirni algoritem [3] je bil sestavljen za konkretni problem uravnoteževanja štirih skupin bolnikov z rakom dojke glede na to, s katerim od dveh načinov so bili zdravljeni (s kemoterapijo z antraciklini, ali s kemoterapijo po shemi CMF

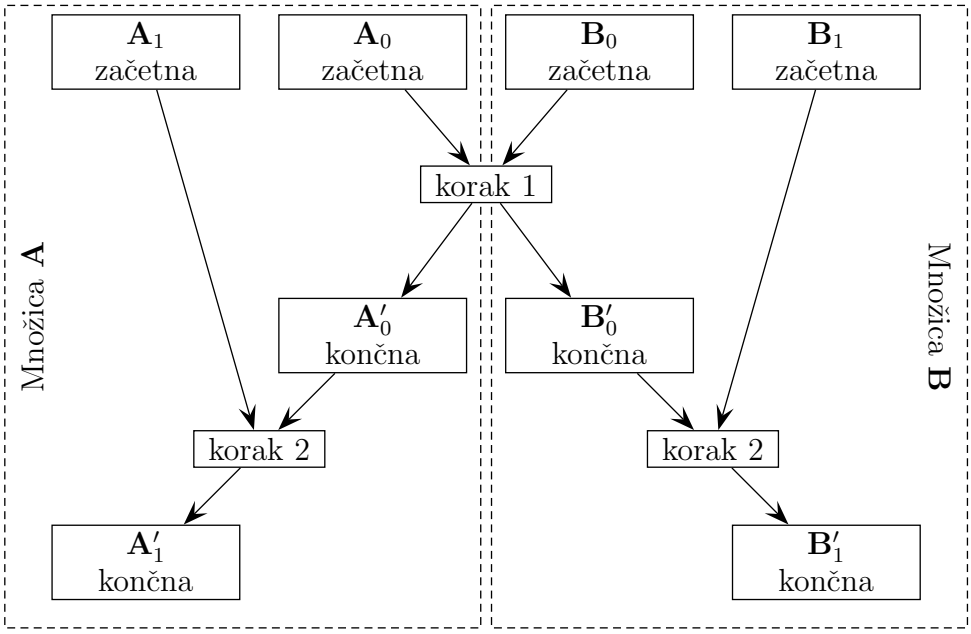
- ciklofosfamid, metotreksat in 5-fluorouracil) in glede na eno od dveh vrednosti kombinirane karakteristike bolnikov z oznako uPA/PAI-1 [6]. Vrednost slednje karakteristike je bila določena na podlagi vrednosti karakteristik urokinaznega aktivatorja plazminogena (uPA) in njegovega glavnega zaviralca (PAI-1), možni vrednosti pri obeh pa sta: nizek nivo in visok nivo. Vrednost uPA/PAI-1 zavzame nizek nivo, če oba, uPA in PAI-1, vsebujeta nizek nivo, sicer pa zavzame visok nivo. Način zdravljenja in vrednost uPA/PAI-1 razbijeta množico bolnikov na štiri podmnožice: po načinu zdravljenja in po vrednosti karakteristike uPA/PAI-1. Cilj je bilo preveriti hipotezo, da imajo bolniki z visokim nivojem uPA/PAI-1 večjo stopnjo preživetja pri zdravljenju s kemoterapijo z antraciklini, kot pa pri zdravljenju s kemoterapijo po shemi CMF.

Množico oz. kohorto bolnikov, ki so bili zdravljeni s kemoterapijo z antraciklini, označimo z $\mathbf{A}$, množico oz. kohorto bolnikov, ki so bili zdravljeni s kemoterapijo po shemi CMF, pa z $\mathbf{B}$. Obe množici nadalje razdelimo glede na vrednost karakteristike uPA/PAI-1. Množica z indeksom 0 vsebuje bolnike z nizkim nivojem uPA/PAI-1, množica z indeksom 1 pa vsebuje bolnike z visokim nivojem uPA/PAI-1. Tako dobimo množice $\mathbf{A}_0$, $\mathbf{A}_1$, $\mathbf{B}_0$ in $\mathbf{B}_1$ (slika 2.1).

Značilnost tega problema je, da je število bolnikov z nizkim nivojem uPA/PAI-1 (množici $\mathbf{A}_0$ in $\mathbf{B}_0$) približno trikrat manjše od števila bolnikov z visokim nivojem uPA/PAI-1 (množici $\mathbf{A}_1$ in $\mathbf{B}_1$), zato algoritem v prvem koraku prične z medsebojnim uravnoteževanjem obeh manjših podmnožic. Rezultat sta dve uravnoteženi množici, $\mathbf{A}'_0$ in $\mathbf{B}'_0$. Ti množici predstavljata jedri, s katerima algoritem v drugem koraku uravnoteži množici $\mathbf{A}_1$ in $\mathbf{B}_1$. Pri tem skuša algoritem ohraniti razmerje 1:3 med množicama $\mathbf{A}_0$ in $\mathbf{A}_1$ ter med $\mathbf{B}_0$ in $\mathbf{B}_1$. Bistvena razlika med korakoma je v tem, da algoritem v prvem koraku oblikuje obe začetni množici z indeksom 0, v drugem koraku pa dvakrat oblikuje po eno množico z indeksom 1, kjer že oblikovane množice z indeksom 0 ne spreminja. Potek opisanega računanja je shematsko prikazan na sliki 2.2.



Slika 2.1: Delitev množice bolnikov na 4 podmnožice



Slika 2.2: Shematski potek uravnoteževanja v izvirnem algoritmu

Uravnoteževanje množic v prvem koraku poteka v dveh fazah. Najprej se med obema začetnima množicama poiščejo pari enakih elementov, torej kjer funkcija podobnosti dveh elementov vrne vrednost 0. Rezultat sta uravnoteženi množici, kjer ima vsak element ene množice svoj par v drugi

Algoritem 2.1: Prva faza prvega koraka.

function poisci_enake($\mathbf{A}_0$, $\mathbf{B}_0$):
    $\mathbf{A}'_0 \leftarrow \varnothing$
    $\mathbf{B}'_0 \leftarrow \varnothing$
    foreach $a \in \mathbf{A}_0$:
        foreach $b \in \mathbf{B}_0 \setminus \mathbf{B}'_0$:
            if $s(a, b) = 0$:
                $\mathbf{A}'_0 \leftarrow \mathbf{A}'_0 \cup \{a\}$
                $\mathbf{B}'_0 \leftarrow \mathbf{B}'_0 \cup \{b\}$
    return $\mathbf{A}'_0$, $\mathbf{B}'_0$

Algoritem 2.2: Druga faza prvega koraka.

procedure dodaj_podobne($\mathbf{A}_0$, $\mathbf{B}_0$, $\mathbf{A}'_0$, $\mathbf{B}'_0$):
    foreach $a \in \mathbf{A}_0 \setminus \mathbf{A}'_0$:
        foreach $b \in \mathbf{B}_0 \setminus \mathbf{B}'_0$:
            if $(s(a, b) \leq M)$ **and**
                (mnozici_sta_podobni($\mathbf{A}'_0 \cup \{a\}$, $\mathbf{B}'_0 \cup \{b\}$):
                $\mathbf{A}'_0 \leftarrow \mathbf{A}'_0 \cup \{a\}$
                $\mathbf{B}'_0 \leftarrow \mathbf{B}'_0 \cup \{b\}$

množici, ki je po meri podobnosti enak. Ta faza je orisana v algoritmu 2.1. V drugi fazi pa algoritem med začetnima množicama poišče še pare podobnih elementov, ter jih doda končnima množicama, vkolikor tako dodajanje ne povzroči, da si množici nista več podobni. Druga faza je orisana v algoritmu 2.2. Fazi se razlikujeta v tem, da algoritem v prvi fazi nabira samo pare enakih elementov, v drugi fazi pa nabira pare podobnih elementov, ki ne kvarijo uravnoteženosti obeh množic. Potreba po tako različnih fazah izvira iz dejstva, da uravnoteženost obeh množic lahko merimo šele, ko sta dovolj veliki. Pri statističnem testu χ^2 se standarno zahteva, da je vsaka vrednost zastopana vsaj petkrat (oz. da je njena frekvenca vsaj 5). Zaradi tega čisto

Algoritem 2.3: Prva faza drugega koraka.

```
function poisci_N_najprej_enakih_potem_podobnih(A_0, A_1, N):
    A'_1 ← ∅
    foreach a_0 ∈ A_0:
        n ← 0
        foreach a_1 ∈ A_1 \ A'_1:
            if (s(a_0, a_1) = 0) and (n < N):
                A'_1 ← A'_1 ∪ {a_1}
                n ← n + 1
        foreach a_1 ∈ A_1 \ A'_1:
            if (s(a_0, a_1) ≤ M) and (n < N):
                A'_1 ← A'_1 ∪ {a_1}
                n ← n + 1
    return A'_1
```

na začetku še ne moremo meriti uravnoteženosti množic. Da premosti ta problem, algoritem na začetku išče zgolj pare enakih elementov, saj le-ti ne morejo pokvariti uravnoteženosti med množicama. Tako oblikovani množici predstavljata medsebojno uravnoteženi množici $\mathbf{A}'_0$ in $\mathbf{B}'_0$. Ker je velikost množic $\mathbf{A}_0$ in $\mathbf{B}_0$ približno trikrat manjša od velikosti množic $\mathbf{A}_1$ in $\mathbf{B}_1$, nadalje algoritem postopa tako, da množic $\mathbf{A}'_0$ in $\mathbf{B}'_0$ ne spreminja več, saj so elementi iz teh množic bolj redki in s tem bolj kritični.

V drugem koraku algoritem posebej uravnoteži množici $\mathbf{A}_1$ in $\mathbf{B}_1$, dobimo uravnoteženi množici $\mathbf{A}'_1$ in $\mathbf{B}'_1$. Ker se korak izvede dvakrat, enako enkrat za $\mathbf{A}_1$ in enkrat za $\mathbf{B}_1$, se v nadaljevanju opisa omejimo samo na $\mathbf{A}_1$. Tudi ta korak je sestavljen iz dveh faz. V prvi fazi poizkuša algoritem vsakemu elementu iz množice $\mathbf{A}_0$ poiskati do tri enake ali podobne elemente v množici $\mathbf{A}_1$, z namenom ohranitve razmerja med množicama. Pri tem algoritem ne pazi na uravnoteženost obeh množic, ampak zgolj skuša upariti čim več elementov. Opisan postopek je prikazan v algoritmu 2.3.

V drugi fazi algoritem poskrbi za to, da množici $\mathbf{A}'_0$ in $\mathbf{A}'_1$ postaneta urav-

noteženi. Poišče element množice $\mathbf{A}'_1$, ki najbolj prispeva k neuravnoteženosti množice $\mathbf{A}'_1$ glede na množico $\mathbf{A}'_0$ in ga izloči iz množice, ter to ponavlja dokler množici ne postaneta uravnoteženi, ali dokler ne najde več takega elementa. Za merjenje uravnoteženosti množic algoritem uporablja prej opisan Pearsonov statistični test χ^2 ter predpisane mejne p-vrednosti posameznih uravnoteževalnih karakteristik. Iz mejne p-vrednosti in iz prostostne stopnje posamezne karakteristike izračuna mejno vrednost χ^2_m. Za oceno tega, koliko posamezni element iz množice $\mathbf{A}'_1$ prispeva k neuravnoteženosti množice $\mathbf{A}'_1$, algoritem izračuna hevristično oceno e na podlagi tega, kolikšna je razlika med χ^2_m in trenutno vrednostjo χ^2 posamezne karakteristike, ter kakšna bi bila ta razlika, če bi posamezni element odstranil iz množice [3]:

$$e = \begin{cases} -1, & p_a \geq 0 \wedge pc_a < 0 \\ \sum\limits_{p_a<0} 1000\, p_a \Delta_a + \sum\limits_{p_a \geq 0} -(M - p_a)\Delta_a, & \text{sicer} \end{cases} \tag{2.6}$$

kjer je p_a razlika med χ^2_m in dejansko vrednostjo χ^2 karakteristike a, pc_a je taka razlika brez trenutno obravnavanega elementa, $\Delta_a = p_a - pc_a$ in $M = \max_a(p_a) + 0{,}1$. Hevristična ocena je sestavljena tako, da dani element izvzame iz nadaljnje obravnave (vrednost $e = -1$), vkolikor bi njegova izključitev privedla katero že uravnoteženo karakteristiko v neuravnoteženo stanje (pogoj $p_a \geq 0 \wedge pc_a < 0$), sicer pa zanj sešteje nagrade in kazni glede na to, če se uravnoteženost posamezne karakteristike izboljša ali poslabša. Člen $\sum_{p_a<0} 1000\, p_a \Delta_a$ predstavlja nagrado za neuravnotežene karakteristike, katerih neuravnoteženost bi se zmanjšala. Člen $\sum_{p_a \geq 0} -(M - p_a)\Delta_a$ pa predstavlja ali nagrado ali kazen za že uravnotežene karakteristike, katerih uravnoteženost bi se z izključitvijo danega elementa ali izboljšala ali poslabšala. Konstanta 0,1 pri izračunu M poskrbi za minimalno kazen ali nagrado. S konstanto 1000 pa želimo poudariti, da ima prednost uravnoteževanje neuravnoteženih karakteristik.

2.3 Dodatne strategije

Sestavili smo dodatne strategije za uravnoteževanje množic elementov, ter s tem skušali dobiti boljše rezultate od izvirnega algoritma. Pri tem smo uporabili algoritem minimin [7] in požrešni algoritem. Z bolj splošnimi in boljšimi hevrističnimi algoritmi, kot so A* in njegove verzije, se nismo ukvarjali. Da bi lahko uporabili algoritem A*, bi morali definirati funkciji razdalje med dvema stanjema in hevristične cenilke razdalje do ciljnega stanja. Definicija takih funkcij ni trivialna, saj bi morali upoštevati vse kvalitete, ki jih hočemo optimizirati (maksimalna velikost, uparjenost in uravnoteženost množic). Le-te so si v nasprotju, saj intuitivno lahko v splošnem povečamo velikost množic samo na račun znižanja kvalitet uparjenosti in uravnoteženosti. V želji da čim prej damo algoritme v uporabo na realnih podatkih, smo zato izhajali iz izvirnega algoritma in sestavili nekaj dodatnih preprostih algoritmov. Na podlagi rezultatov v praksi se bo v prihodnosti pokazalo, če in kakšne izboljšave v algoritmih so potrebne.

2.3.1 Požrešna metoda

Pri iskanju podobnih elementov med dvema množicama izvirni algoritem po vrsti pregleduje elemente in nabere pare elementov, kjer sta si elementa enaka oz. podobna. Namesto tega postopka smo uporabili požrešno metodo tako, da v vsaki iteraciji med vsemi potencialnimi pari poišče in izbere par najbolj podobnih elementov, ter ga izloči iz nadaljnje obravnave. Algoritem ponavlja iteracije, dokler obstaja še vsaj en neizbrani par elementov, ki sta si podobna. Postopek izbire najboljšega para si pomaga z matriko cen, kjer stolpci matrike predstavljajo elemente ene množice, vrstice pa elemente druge množice. Posamezna cena oz. vrednost celice v matriki je vrednost funkcije podobnosti med elementoma s (enačba 2.1). Postopek izbere najboljši par elementov tako, da pregleda vse celice in izbere tako, ki ima najmanjšo vrednost. Če je takih celic več, izbere prvo tako. Če je vrednost celice nad predpisanim pragom za podobnost, algoritem konča z delom, saj to pomeni,

da ni več neizbranih parov podobnih elementov. Po izboru celice je potrebno odgovarjajoča elementa označiti kot zasedena, da ne moreta biti kandidata pri kasnejših izborih. Postopek to naredi z označitvijo vseh celic v istem stolpcu in vseh celic v isti vrstici, da so zasedene. Kadar algoritem uparja k elementov ene množice k vsakemu elementu druge množice, se postopek označevanja spremeni. Tedaj, če privzamemo da uparjamo k stolpcev k vsaki vrstici matrike, označimo vse celice v posamezni vrstici kot zasedene šele, ko smo k tej vrstici uparili k-ti stolpec po vrsti.

2.3.2 Algoritem minimin z rezanjem alfa

Izvirna strategije nabiranja je naivne narave, saj po vrsti preiskuje pare elementov, ter izbere tiste, ki ustrezajo kriteriju. Pri taki strategiji ne moremo biti prepričani v optimalnost izbire, kjer hočemo maksimizirati velikost rezultirajočih množic. Preprosta rešitev tega problema bi bila, da bi v danih omejitvah z ozirom na podobnost in uravnoteženost, zgenerirali vse mogoče različne rešitve in bi na koncu privzeli rešitev, ki vsebuje največ elementov. Vsako rešitev si lahko predstavljamo kot eno pot po drevesu, kjer posamezno vozlišče predstavlja neko vmesno stanje dveh množic, ki ju uravnotežujemo, posamezna povezava pa predstavlja potezo, to je par podobnih elementov, ki ju izberemo, da pridemo do novega stanja. Koren drevesa predstavlja začetno stanje brez elementov, listi drevesa pa predstavljajo vse možne rešitve, to so končna stanja, od kjer naprej ni mogoče več narediti poteze, torej ni moč več najti parov podobnih elementov.

Kombinatorična zahtevnost takega reševanja je prevelika za praktično uporabo, namesto tega smo uporabili algoritem minimin z rezanjem alfa (*alpha pruning*), ki ga je Korf uporabil v fazi iskanja poteze za algoritem RTA* [7]. Samega algoritma RTA*, ki se uporablja v fazi izvedbe poteze, nismo uporabili. Algoritem RTA* se uporablja pri problemih, ki rabijo odločanje v realnem času, kar ni narava našega problema, poleg tega smo postavljeni pred enako nalogo določitve funkcij razdalje med dvema stanjema in hevristične cenilke razdalje do ciljnega stanja, kot pri algoritmu A*. Algoritem

minimin deluje tako, da preiskuje opisano drevo od trenutnega stanja do določene globine. Cilj, ki ga zasledujemo, je pridobiti čim večji uravnoteženi množici. Intuitivno gre to na račun zmanjšane kvalitete uparjenosti in uravnoteženosti množic, vendar sta ti dve kvaliteti v našem primeru sekundarna cilja in zadošča, da uparjenost in uravnoteženost dosegata vsaka svojo dano minimalno kvaliteto. Zato uparjenosti in uravnoteženosti z namenom poenostavitve ne upoštevamo v hevristični oceni. Za uparjenost skrbimo s tem, da izbiramo samo pare podobnih elementov, za uravnoteženost pa poskrbimo na koncu z izločanjem elementov, kakor je to narejeno v izvirnem algoritmu.

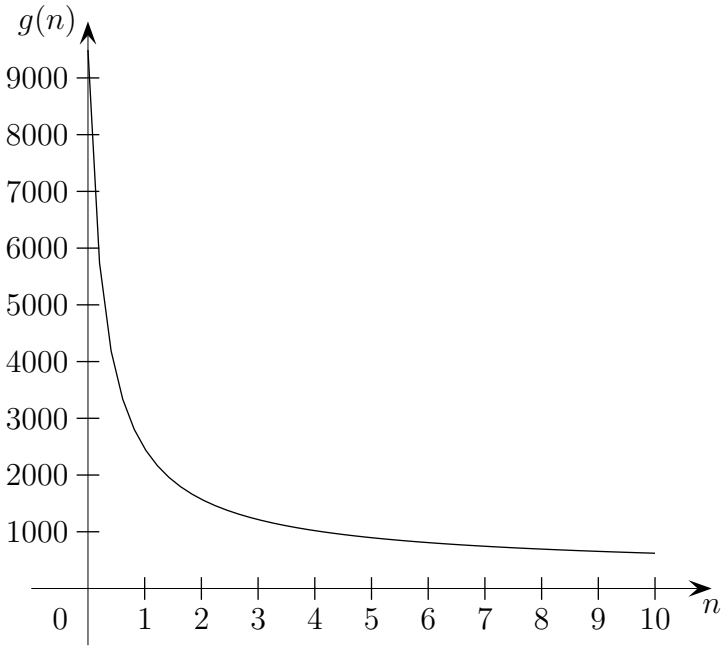
Vsako stanje ocenimo s hevristično oceno njegove prostostne stopnje, to je stopnja njegove prostosti pri izbiri naslednje poteze. Več kot je možnih potez iz stanja, večja je njegova prostostna stopnja. Predpostavljamo, da če vsakič izberemo potezo, ki vodi v stanje z največjo prostostno stopnjo, bomo s tem maksimizirali velikost množic v rezultatu. V ta namen smo definirali hevristično oceno, ki kaznuje stanja z manjšo prostostno stopnjo. V nadaljevanju bomo to oceno uporabili za iskanje poteze, ki vodi v stanje z najmanjšo kaznijo oz. ceno stanja in s tem z največjo prostostno stopnjo.

Hevristično oceno smo definirali s sledečim postopkom. V vsakem stanju imamo na voljo prej opisano matriko cen, ki predstavlja mere podobnosti v posameznem paru med elementoma iz obeh množic. Celice matrike, ki niso označene kot zasedene in ki vsebujejo ceno, ki ne presega praga za podobnost, predstavljajo možne naslednje poteze. Uvedemo funkcijo $n(e)$, ki za dano vrstico ali stolpec, predstavljeno z danim elementom e ene ali druge množice, prešteje število nezasedenih celic s ceno pod pragom. Drugače povedano nam $n(e)$ pove, s koliko elementi nasprotne množice lahko e tvori par, kar je ekvivalentno številu potez, ki jih omogoča element e. Rezultat funkcije $n(e)$ smo hoteli utežiti s kaznovalno funkcijo na tak način, da se rezultat $n(e) = 0$ zelo kaznuje, rezultat $n(e) = 1$ se kaznuje precej manj, pri $n(e) = 2$ se kaznuje še nekoliko manj, za nadaljnje rezultate pa se kaznuje po položno padajoči približno linearni funkciji. S tem smo hoteli doseči močno izogibanje potezam, ki povzročijo izločitev elementa e iz nadaljnje obravnave in manj

močno izogibanje potezam, kjer z elementom e še možno tvoriti nadaljnje poteze. Opisano po obliki ustreza obrnjeni logaritemski funkciji, ki smo jo zato vzeli za osnovo pri definiciji kaznovalne funkcije g:

$$g(n(e)) = -\frac{1000}{\ln\left(9/(10 + 3{,}5\, n(e))\right)} \ , \tag{2.7}$$

kjer $n(e)$ predstavlja število možnih potez za element e. Konstante v funkciji smo empirično določili tako, da je funkcija $g(n)$ definirana za pozitivne vrednosti n, ter da ustreza našim zahtevam glede željenih uteži. Graf funkcije $g(n)$ za $n = 0..10$ je predstavljen na sliki 2.3. Iz njega je razvidno, da je funk-



Slika 2.3: Utežitvena funkcija $g(n)$ za element s hevristično oceno prostostne stopnje n.

cija $g(n)$ sprva zelo strma, pri večjih n pa postaja vedno bolj položna. To je v skladu z našo zahtevo, da se majhno število možnih potez za dani element zelo kaznuje (najbolj ekstremno, ko ni več možnih potez), pri višjem številu možnih potez pa razlika med sosednjima številoma ni več tako pomembna, zato so tudi razlike v kaznih majhne.

Hevristično oceno stanja $h(S)$ dobimo s seštevkom vseh uteženih števil naslednjih možnih potez:

$$h(S) = \sum_{e \in \mathbf{A} \cup \mathbf{B}} g(n(e)) \ , \tag{2.8}$$

kjer je S stanje, ponazorjeno z matriko cen, katerega hevristično oceno računamo, $\mathbf{A}$ in $\mathbf{B}$ sta množici elementov, med katerima uparjamo, n je prej opisana funkcija, ki za dani element prešteje s koliko elementi nasprotne množice lahko tvori par, g pa je funkcija iz enačbe 2.7.

	a	b	c	d	e
v	3	5	2	7	1
w	1	8	1	2	4
x	2	3	5	0	4
y	6	7	8	4	0
z	0	6	2	5	3

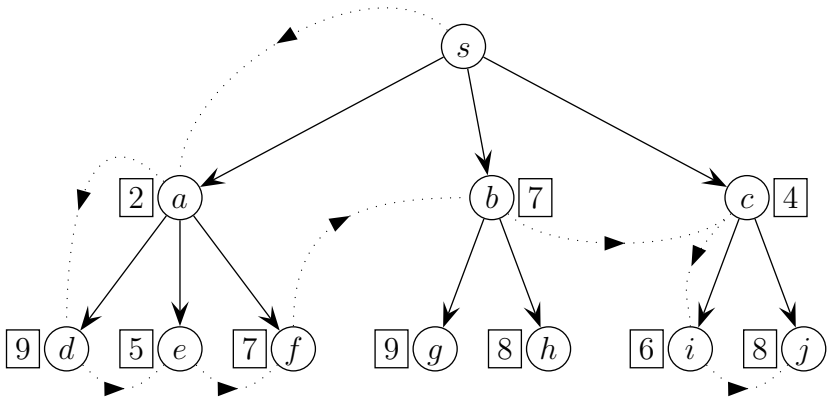
Slika 2.4: Primer uparjanja elementov dveh množic z matriko cen.

Za ilustracijo vzemimo primer matrike na sliki 2.4, ki vsebuje cene oz. mere podobnosti za uparjanje elementov množic $\{a, b, c, d, e\}$ in $\{v, w, x, y, z\}$. Prag, kjer se šteje, da sta si dva elementa še podobna, je določen pri 3. Vse celice s cenami, ki presegajo prag, so prikazane šrafirano; parov s takimi cenami ne moremo uporabiti, saj si elementa v paru nista podobna. Vzemimo, da hočemo upariti elementa d in x, ki sta si s ceno 0 enaka. S tem naredimo potezo v naslednje stanje, kjer se ta dva elementa ne smeta več uporabiti, saj sta bila ,,porabljena“. Zato iz nadaljnje obravnave izločimo vse celice matrike, kjer nastopata ta dva elementa. Take celice so na sliki obarvane s sivo barvo. Vendar s tako potezo iz nadaljnje obravnave izločimo tudi element b, saj ga ni moč porabiti v nobenem od možnih parov, ker ni podoben nobenemu od preostalih elementov nasprotne množice. Število možnih nadaljnjih

potez pri elementu b pade na 0, zaradi česar se na tem mestu potezi pripiše največja kazen, kakor je razvidno iz grafa funkcije $g(n)$ na sliki 2.3.

Ilustrirana poteza ima za posledico, da iz nadaljnje obravnave izpadejo trije elementi. Če bi namesto te poteze izbrali katero drugo potezo, ki postransko ne povzroči izpada dodatnih elementov, bi iz nadaljnje obravnave izpadla samo izbrana elementa in bi bila hevristična ocena novega stanja manjša. Za izbiro poteze, ki privede do stanja z najmanjšo hevristično oceno v omejeni okolici, smo uporabili algoritem minimin z rezanjem alfa [7]. Algoritem minimin do omejene globine v drevesu stanj poišče stanje z najmanjšo hevristično oceno, v smeri katerega naredimo naslednjo potezo. Vsako vmesno stanje na poti do rešitve (oz. cilja) si lahko predstavljamo kot koren poddrevesa v prej opisanem celotnem drevesu reševanja. Da lahko izberemo naslednjo potezo, najprej z iskanjem v globino do določene globine pregledamo poddrevo, katerega koren je trenutno stanje. Stanja na tej največji globini so listi poddrevesa. Med preiskovanjem ves čas vzdržujemo minimalno ceno stanja v obiskanih listih. Na začetku, ko pričenjamo s preiskovanjem, nastavimo minimalno ceno na ∞. Ko pridemo do prvega lista, se njegova cena privzame za minimalno ceno. Ko pridemo do lista z manjšo ceno od minimalne, se minimalna cena prepiše z njegovo ceno, in tako naprej. Pri preiskovanju poddrevesa se poslužimo dejstva, da je hevristična ocena $h(S)$ monotono naraščajoča z globino drevesa. To izvira iz dejstva, da vsaka poteza zniža prostostno stopnjo vsaj dvema elementoma novega stanja. Zaradi te lastnosti hevristične ocene lahko uporabimo rezanje alfa (*alpha pruning*) in preskočimo preiskovanje vmesnih stanj poddrevesa, ki imajo enako ali večjo ceno od trenutne minimalne cene, saj zagotovo s preiskovanjem potez, ki sledijo takemu stanju, ne bomo dobili stanja z manjšo ceno od trenutne minimalne cene.

Primer takega preiskovanja je ilustriran na sliki 2.5, kjer iščemo do globine 2. Vsakemu stanju, ki sledi iz trenutnega korena s, je pripisana njegova hevristična ocena. Na začetku dodelimo minimalni ceni vrednost ∞ in se iz korenskega stanja premaknemo v stanje a s ceno 2. Ker je cena tega stanja



Slika 2.5: Primer iskanja najboljše poteze.

$2 < \infty$, obiščemo še stanja, ki sledijo iz a. Ta stanja so na največji globini, torej so listi poddrevesa in zato prvo stanje d spremeni minimalno ceno v 9. Naslednje stanje je e, katerega cena je 5, ki je manjša od trenutne minimalne cene, zato nova minimalna cena postane 5. Zadnje stanje pod a je f, ki pa ima večjo ceno od trenutne minimalne cene. Sedaj se pomaknemo nivo višje in nadaljujemo z naslednjim stanjem na tem nivoju, s stanjem b. Ker ima to stanje ceno 7, ki je večja od trenutne minimalne cene 5, preskočimo obisk stanj, ki sledijo iz njega. Nadaljujemo s stanjem c, ki ima manjšo ceno od trenutne minimalne cene. Obiščemo še obe stanji, ki sledita iz stanja c, ki pa ne spremenita minimalne cene. Na koncu preiskovanja ugotovimo, da je minimalna cena bila dosežena v poddrevesu, ki ima koren v stanju a, zato bo naša naslednja poteza, da preidemo v stanje a, ki postane koren novega poddrevesa. Ta postopek ponavljamo, dokler ne pridemo v stanje brez nadaljnjih potez.

2.3.3 Naberi in oklesti

Izvirni algoritem postopa v dveh korakih, kjer se posamezni korak nadalje deli na dve fazi. V prvem koraku algoritem sestavi začetni množici $\mathbf{A}'_0$ in $\mathbf{B}'_0$ tako, da najprej nabere paroma enake, nato pa še podobne elemente, ki ne pokvarijo uravnoteženosti. V drugem koraku pa za vsako začetno množico

($\mathbf{A}'_0$ in $\mathbf{B}'_0$) najprej sestavi nasprotno množico ($\mathbf{A}'_1$ in $\mathbf{B}'_1$), nato pa jo oklesti, dokler ne doseže uravnoteženosti. Z željo po poenostavitvi postopka smo ta algoritem nekoliko predrugačili. Nastal je preprost algoritem, ki najprej nabere vse paroma podobne elemente za vse štiri množice, nato pa klesti vse štiri množice, dokler ne doseže uravnoteženosti. Pri nabiranju parov smo se poslužili enake ideje, kot pri izvirnem algoritmu. Najprej uparimo elemente med dvema manjšima množicama ($\mathbf{A}_0$ in $\mathbf{B}_0$), nato pa k vsaki tako nastali množici poiščemo še pare iz nasprotne množice v razmerju 1:3. Bistvena razlika od izvirnega algoritma tukaj je, da samo uparjamo podobne elemente, brez dodatnih omejitev. Šele na koncu se poslužimo izločanja elementov, ki najbolj kvarijo uravnoteženost. Ker smo med vsemi množicami uparjali brez upoštevanja, kako to vpliva na uravnoteženost, smo sestavili nekoliko drugačen postopek izločanja. Izvirni algoritem uporablja postopek, v katerem za izločanje obravnava elemente samo ene množice. Z ozirom na to, da je k že prej uravnoteženi množici (z indeksom 0) uparil novo množico (z indeksom 1), je smiselno, da obravnava samo novo množico. Novi algoritem pa mora obravnavati obe množici, saj nobena še ni bila uravnotežena. Zaradi tega novi algoritem med izločanjem uporabi obstoječi postopek za izločanje dvakrat: najprej ugotovi najboljši element za izločitev iz ene množice, nato pa še najboljši element za izločitev iz druge množice. Izloči tistega, ki je najboljši od obeh tako dobljenih elementov. Vendar moramo uravnoteževati po štirih parih množic: $\mathbf{A}'_0$ in $\mathbf{B}'_0$, $\mathbf{A}'_0$ in $\mathbf{A}'_1$, $\mathbf{B}'_0$ in $\mathbf{B}'_1$ ter $\mathbf{A}'_1$ in $\mathbf{B}'_1$. Tako pridemo do naslednjega postopka za izločanje: najprej pridobimo 8 najboljših elementov za izločitev (ki najbolj prispevajo k neuravnoteženosti), po dva za vsak par množic, nato pa izločimo tistega, ki najbolj prispeva k neuravnoteženosti. To ponavaljamo, dokler množice niso uravnotežene.

Pri testiranju smo za uparjanje enkrat uporabili požrešno metodo, drugič pa algoritem minimin.

2.3.4 Vzemi in oklesti

Za primer, ko hočemo doseči čim večjo velikost uravnoteženih množic, smo sestavili naivni algoritem. Ta algoritem preskoči uparjanje in vzame začetne množice in iz njih po vrsti izloča elemente, ki najbolj prispevajo k neuravnoteženosti množic, nato pa še po vrsti izloča elemente, katerih minimalna podobnost s_m (enačba 2.4) presega prag za podobnost dveh elementov. Algoritem uporabi enak postopek za izločanje elementov, kot preprosti algoritem iz poglavja 2.3.3.

2.4 Ponovljivost rezultatov in možnosti zlorab

Pri medicinskih raziskavah je pomembno, da raziskovalec nima možnosti, da s kakšnim trikom priredi okoliščine raziskave, z namenom da dobi željene rezultate. Prav tako je pomembno, da so rezultati ponovljivi, da se jih lahko neodvisno preveri. Algoritmi, ki smo jih do sedaj opisali, so občutljivi na vrstni red podanih podatkov, kar lahko uporabnik izkoristi tako, da s preurejanjem vrstnega reda elementov v vhodnih množicah poizkusi priti do rezultatov, ki jih favorizira. Uporabniku skušamo otežiti tak primer zlorabe s tem, da vhodne množice elementov najprej uredimo po vrednostih vseh karakteristik. S tem preprečimo možnost zlorabe s preurejanjem vrstnega reda vhodnih elementov. Vendar se s tem ne ognemo v celoti možnosti zlorabe, saj lahko na primer uporabnik preuredi vrstni red karakteristik v vhodnih podatkih in s tem vpliva na urejenost podatkov. S tem problemom se tukaj nismo podrobno ukvarjali, ker naš namen ni bil iskati možnosti vpliva na rezultate z mešanjem vhodnih podatkov. V prihodnosti pa načrtujemo sestavitev zgoščevalne funkcije, katere rezultat uporabimo kot seme za generator naključnih števil. Zgoščevalna funkcija bo neodvisna tako od vrstnega reda podanih elementov množic, kot od števila in vrstnega reda podanih karakteristik pri vsakem elementu. Taka funkcija bo vedno izračunala enako

število pri enakih podatkih in enakih uravnoteževalnih karakteristikah, ne glede na kakršenkoli vrstni red. Generator naključnih števil pa bo zaradi vedno enakega semena poskrbel za vedno enako perturbacijo elementov vhodnih množic.

Poglavje 3

Rezultati in diskusija

To poglavje opiše rezultate in jih hkrati tudi prediskutira. Ta dva dela načrtno nista razdeljena na dve poglavji zaradi lažjega toka besedila.

Testirali smo različne strategije uravnoteževanja štirih podmnožic bolnikov na realnih podatkih bolnikov z rakom dojke [6]. Kohorta bolnikov **A** je bila zdravljena s kemoterapijo z antraciklini, kohorta **B** pa s kemoterapijo po shemi CMF. Znotraj vsake kohorte se bolniki nadalje delijo glede na nizko (indeks 0) ali visoko (indeks 1) stanje karakteristike uPA/PAI-1. Na ta način dobimo štiri množice za uravnoteževanje: $\mathbf{A}_0$ in $\mathbf{B}_0$ z bolniki iz obeh kohort z nizkim uPA/PAI-1, ter $\mathbf{A}_1$ in $\mathbf{B}_1$ z bolniki iz obeh kohort z visokim uPA/PAI-1.

Merili smo kvaliteto uravnoteževanja izračunano z uporabo Pearsonovega testa χ^2 (enačba 2.3), kvaliteto uparjenosti u (enačba 2.5), ter število bolnikov c, katerih minimalna podobnost s_m (enačba 2.4) presega prag za mero podobnosti. Množice smo uravnoteževali po sedmih karakteristikah, ki jih je izbral in definiral medicinski ekspert. Po vrsti so to: status menopavze, tip tumorja, status hormonskih receptorjev, tip zdravljenja, velikost tumorja, kardiovaskularna invazija in prizadetost bezgavk. Prizadetost bezgavk je bila trinarna, ostale karakteristike pa binarne.

Kvaliteto uravnoteževanja smo merili za vsako karakteristiko posebej. V rezultatih je prikazana kot tabela med posameznima množicama s tremi

stolpci: mejne p-vrednosti p_m, dejanske p-vrednosti p_d in razlike med njimi Δ. Pozitivna razlika pomeni, da so dejanske p-vrednosti nad mejnimi in kot take sprejemljive, negativna razlika pa pomeni, da dejanske p-vrednosti ne dosegajo mejnih.

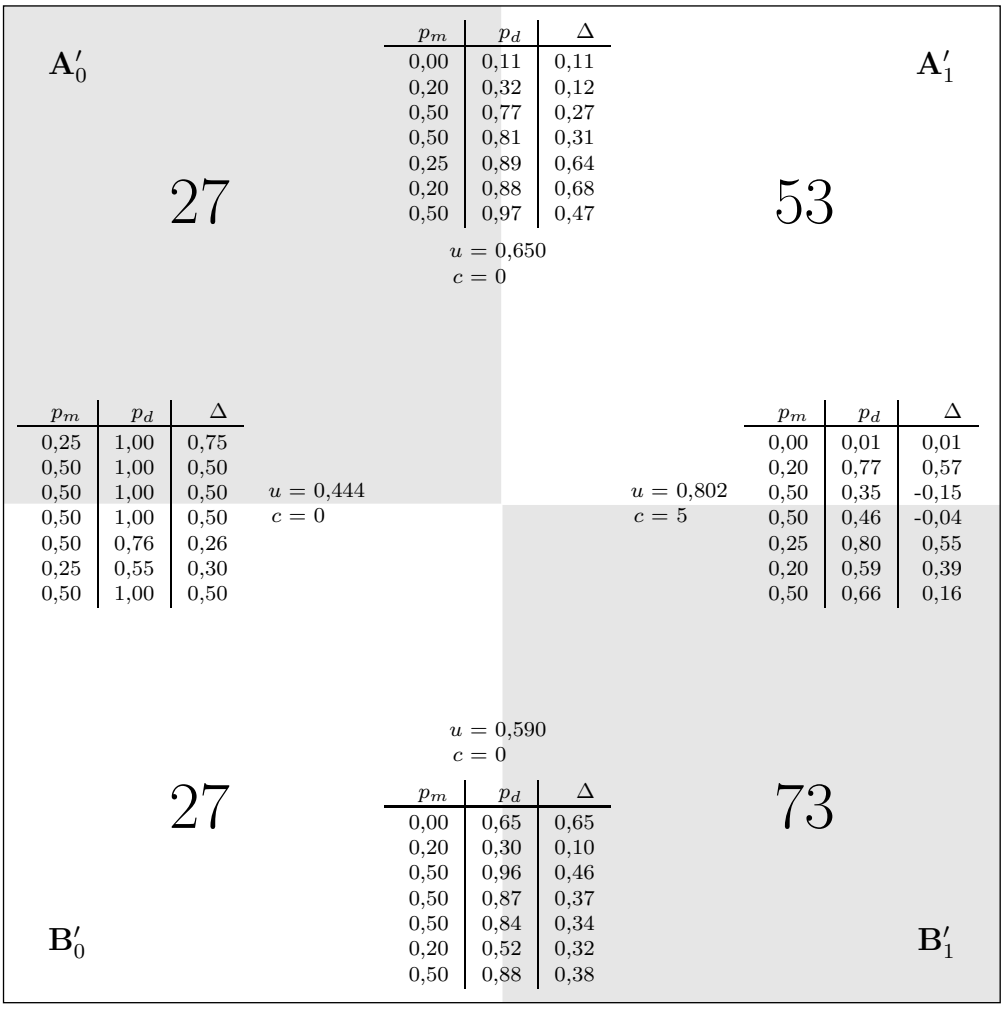
Pri definiciji podobnosti dveh elementov s (enačba 2.1) je medicinski ekspert določil prag z vrednostjo 2, utež pri prizadetosti bezgavk je nastavil na 10, za ostale karakteristike pa je nastavil utež na 1. Glede na prag to pomeni, da se je pri podobnih bolnikih morala prizadetost bezgavk ujemati, drugje pa ni smelo biti odstopanja v več kot dveh karakteristikah.

Na voljo so bili podatki o 655 bolnikih, od tega jih je bilo v množici $\mathbf{A}_0$ 78, v množici $\mathbf{A}_1$ 208, v množici $\mathbf{B}_0$ 80, ter v množici $\mathbf{B}_1$ 289. Cilj je bilo doseči rezultat s čim večjimi uravnoteženimi množicami, v razmerju približno 1:3 med množicama v posameznih kohortah. Pri tem je bil posebej poudarek na doseganju čim večjega števila elementov v manjših množicah $\mathbf{A}'_0$ in $\mathbf{B}'_0$. Ti dve množici sta kritični zaradi veliko manjšega števila kandidatov za uparjanje, poleg tega pa v algoritmih nastopata kot svojevrstno seme za sestavitev množic $\mathbf{A}'_1$ in $\mathbf{B}'_1$.

Vsak rezultat je predstavljen s kvadratno shemo, ki je razdeljena na štiri kvadrante, kjer vsak predstavlja eno množico rezultata. Kvadranti so v zunanjem kotu označeni, katero izmed množic predstavljajo: $\mathbf{A}'_0$, $\mathbf{A}'_1$, $\mathbf{B}'_0$ ali $\mathbf{B}'_1$. Velikost posamezne množice je v vsakem kvadrantu navedena v večji velikosti pisave. Na vsaki meji med sosednjima kvadrantoma se nahajajo tabela, ki ponazarja kvaliteto uravnoteževanja, ter kvaliteta uparjenosti u in število bolnikov c, katerih minimalna podobnost s_m presega prag za mero podobnosti.

3.1 Izvirni algoritem

Rezultate izvirnega algoritma na sliki 3.1 navajamo kot izhodišče za primerjavo z ostalimi strategijami uravnoteževanja. Metrike kažejo, da je kvaliteta uravnoteženosti dobra, le pri primerjavi množic $\mathbf{A}'_1$ in $\mathbf{B}'_1$ vidimo slabši re-



p_m	p_d	Δ
0,00	0,11	0,11
0,20	0,32	0,12
0,50	0,77	0,27
0,50	0,81	0,31
0,25	0,89	0,64
0,20	0,88	0,68
0,50	0,97	0,47

p_m	p_d	Δ
0,25	1,00	0,75
0,50	1,00	0,50
0,50	1,00	0,50
0,50	1,00	0,50
0,50	0,76	0,26
0,25	0,55	0,30
0,50	1,00	0,50

p_m	p_d	Δ
0,00	0,01	0,01
0,20	0,77	0,57
0,50	0,35	-0,15
0,50	0,46	-0,04
0,25	0,80	0,55
0,20	0,59	0,39
0,50	0,66	0,16

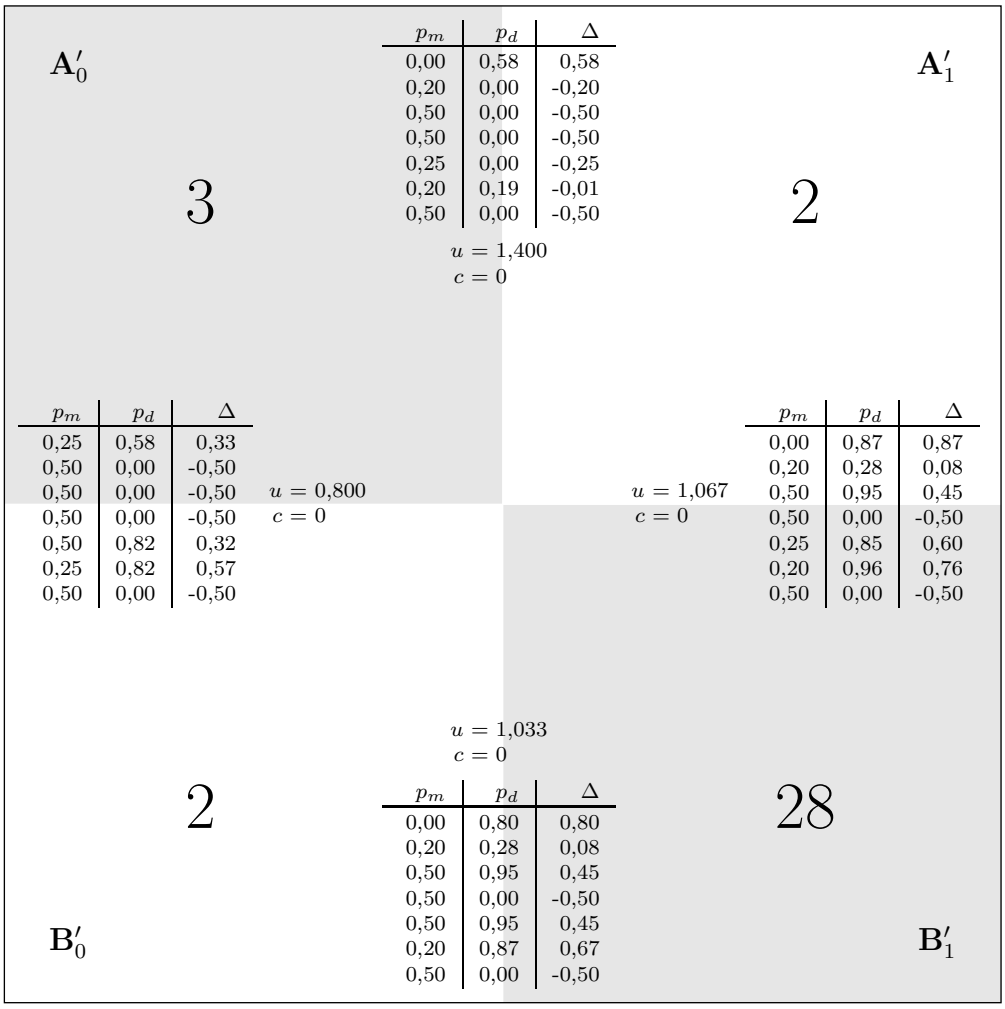
p_m	p_d	Δ
0,00	0,65	0,65
0,20	0,30	0,10
0,50	0,96	0,46
0,50	0,87	0,37
0,50	0,84	0,34
0,20	0,52	0,32
0,50	0,88	0,38

Slika 3.1: Rezultat izvirnega algoritma.

zultat. Vzrok temu je, da algoritem ne poskrbi za uravnoteževanje med tema množicama. Da bi dodatno uravnotežili še ti dve množici, bi morali izločiti elemente, ki kvarijo uravnoteženost. To bi povzročilo zmanjšanje velikosti ene ali obeh množic

3.2 Naivni algoritem

Množice smo skušali uravnotežiti z naivno metodologijo (poglavje 2.3.4), kjer iz množic po vrsti izločamo elemente, ki najbolj kvarijo podobnost. Izločanje



p_m	p_d	Δ
0,00	0,58	0,58
0,20	0,00	-0,20
0,50	0,00	-0,50
0,50	0,00	-0,50
0,25	0,00	-0,25
0,20	0,19	-0,01
0,50	0,00	-0,50

p_m	p_d	Δ
0,25	0,58	0,33
0,50	0,00	-0,50
0,50	0,00	-0,50
0,50	0,00	-0,50
0,50	0,82	0,32
0,25	0,82	0,57
0,50	0,00	-0,50

p_m	p_d	Δ
0,00	0,87	0,87
0,20	0,28	0,08
0,50	0,95	0,45
0,50	0,00	-0,50
0,25	0,85	0,60
0,20	0,96	0,76
0,50	0,00	-0,50

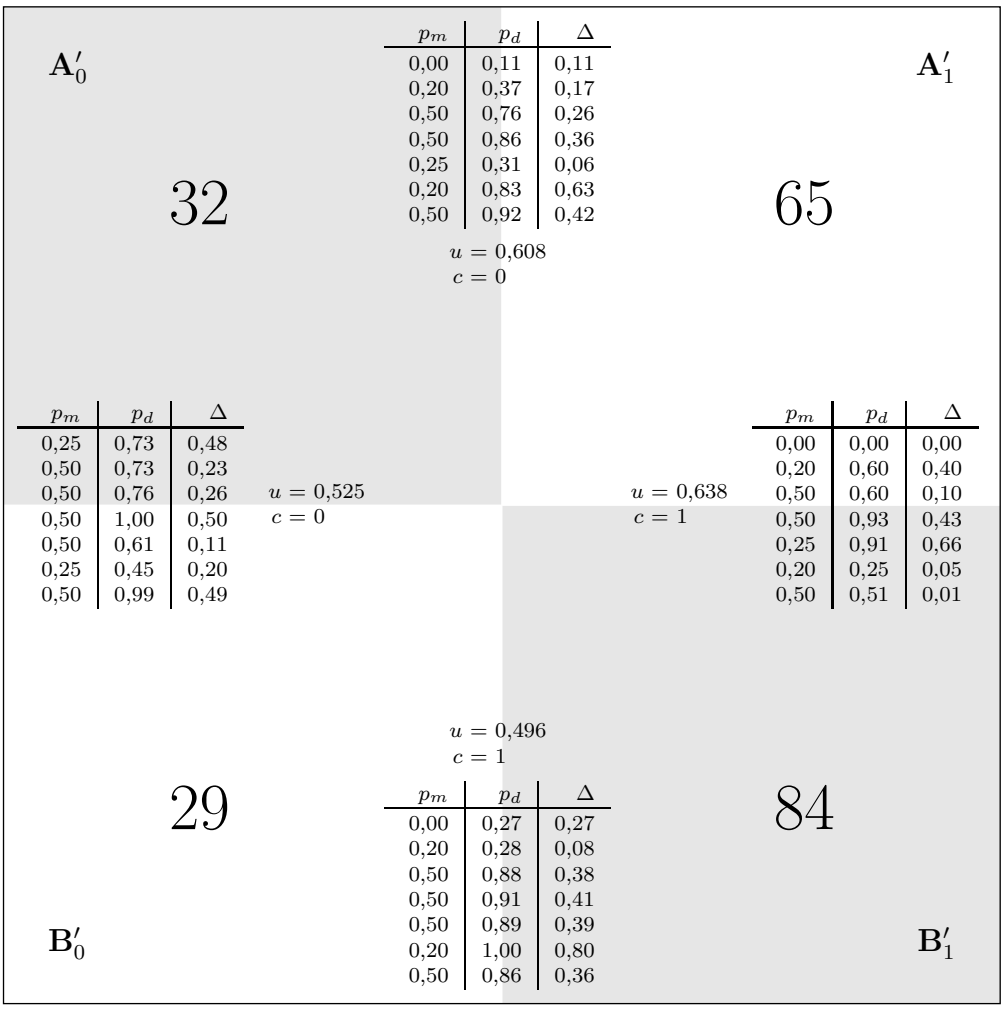
p_m	p_d	Δ
0,00	0,80	0,80
0,20	0,28	0,08
0,50	0,95	0,45
0,50	0,00	-0,50
0,50	0,95	0,45
0,20	0,87	0,67
0,50	0,00	-0,50

Slika 3.2: Rezultat naivnega algoritma, ki zgolj izloči elemente, ki najbolj kvarijo uravnoteženost in uparjenost množic.

je potekalo v dveh fazah: najprej smo požrešno izločali elemente, ki najbolj kvarijo uravnoteženost, v drugi fazi pa še take, ki najbolj kvarijo uparjenost.

Rezultati te metodologije na sliki 3.2 so se izkazali za zelo slabe. Nastale množice so zelo majhne, kvaliteta pa je pravtako slabša od izvirnega algoritma. Domnevamo, da bi boljši rezultat lahko dobili, če bi pri izločanju nekako upoštevali obe metriki, uravnoteženost in uparjenost, skupaj.

3.3 Preprosti algoritem s požrešno metodo



Slika 3.3: Rezultat preprostega algoritma, ki najprej s požrešno strategijo nabere pare podobnih elementov, nato pa izloči elemente, ki najbolj kvarijo uravnoteženost množic.

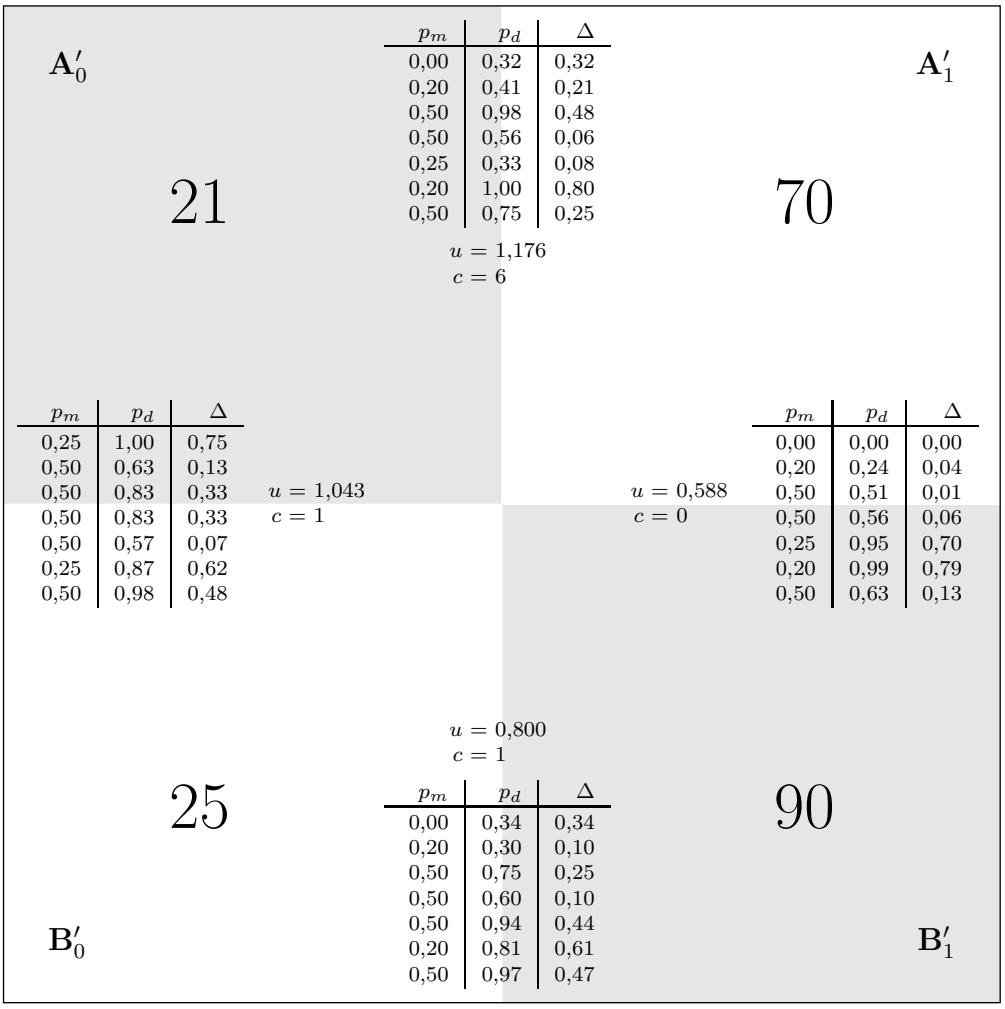
Poizkusili smo poenostaviti izvirni algoritem tako, da smo najprej požrešno nabrali vse paroma podobne elemente, nato pa po smo po vrsti izločali elemente, ki najbolj kvarijo uravnoteženost med množicami (poglavje 2.3.3). Ubrali smo podobno strategijo, kot pri izvirnem algoritmu. Najprej smo uparili elemente med množicama $\mathbf{A}_0$ in $\mathbf{B}_0$, nato pa smo k $\mathbf{A}'_0$ uparili $\mathbf{A}_1$, k

$\mathbf{B}'_0$ pa $\mathbf{B}_1$. Izvirni algoritem je tukaj malo bolj kompliciran, saj pri uparjanju $\mathbf{A}_0$ z $\mathbf{B}_0$ najprej išče enake elemente, nato pa še podobne ter pri tem ohranja množici uravnoteženi. Na koncu smo iz množic izločili elemente, ki kvarijo uravnoteženost. Elemente smo izločali tako, da smo za vsak od štirih parov množic poiskali tak element, z izločitvijo katerega bi se uravnoteženost med množicama najbolj izboljšala, nato pa smo še izmed teh elementov izbrali takega, izločitev katerega najbolj prispeva k izboljšanju uravnoteženosti in ga izločili.

Rezultati tega algoritma so prikazani na sliki 3.3. Glede na rezultat izvirnega algoritma na sliki 3.1 so vidno boljši. Vsaka od štirih množic je večja, tudi metrike so v splošnem malo boljše. Med množicama $\mathbf{A}'_0$ in $\mathbf{B}'_0$ vidimo slabše vrednosti metrik, k čemur prispeva povečanje velikosti teh množic. To povečanje se je zgodilo zato, ker se pri uparjanju začetnih dveh množic nismo ozirali na uravnoteženost, kakor to dela izvirni algoritem. Ker sta s tem začetni množici $\mathbf{A}'_0$ in $\mathbf{B}'_0$ postali večji, je algoritem lahko k elementom le-teh poiskal več parov iz $\mathbf{A}_1$ in $\mathbf{B}_1$. Vrednosti uravnoteženosti skupin med vsemi štirimi pari množic kažejo pozitiven rezultat, kar je še eno izboljšanje napram izvirnemu algoritmu. Vzrok temu je, da tukaj pri izločanju elementov, ki kvarijo uravnoteženost, upoštevamo vse pare množic hkrati. Izvirni algoritem je izločal samo iz množic $\mathbf{A}'_1$ glede na $\mathbf{A}'_0$ in $\mathbf{B}'_1$ glede na $\mathbf{B}'_0$, torej ni globalno upošteval vpliva izločitve posameznega elementa, kot to počne algoritem za izločanje tukaj.

3.4 Preprosti algoritem in algoritem minimin

Preprostemu algoritmu smo spremenili strategijo nabiranja parov podobnih elementov. Namesto strategije s požrešnim iskanjem parov z najboljšo podobnostjo smo uporabili algoritem minimin, opisan v poglavju 2.3.2, ki preiskuje drevo stanj z uporabo hevristične ocene v enačbi 2.8. Algoritem smo testirali na globinah preiskovanja 1 in 2. Čas izvajanja tako spremenjenega algoritma se je precej podaljšal. S prej opisanimi algoritmi smo dobili rezultat sko-



p_m	p_d	Δ
0,00	0,32	0,32
0,20	0,41	0,21
0,50	0,98	0,48
0,50	0,56	0,06
0,25	0,33	0,08
0,20	1,00	0,80
0,50	0,75	0,25

p_m	p_d	Δ
0,25	1,00	0,75
0,50	0,63	0,13
0,50	0,83	0,33
0,50	0,83	0,33
0,50	0,57	0,07
0,25	0,87	0,62
0,50	0,98	0,48

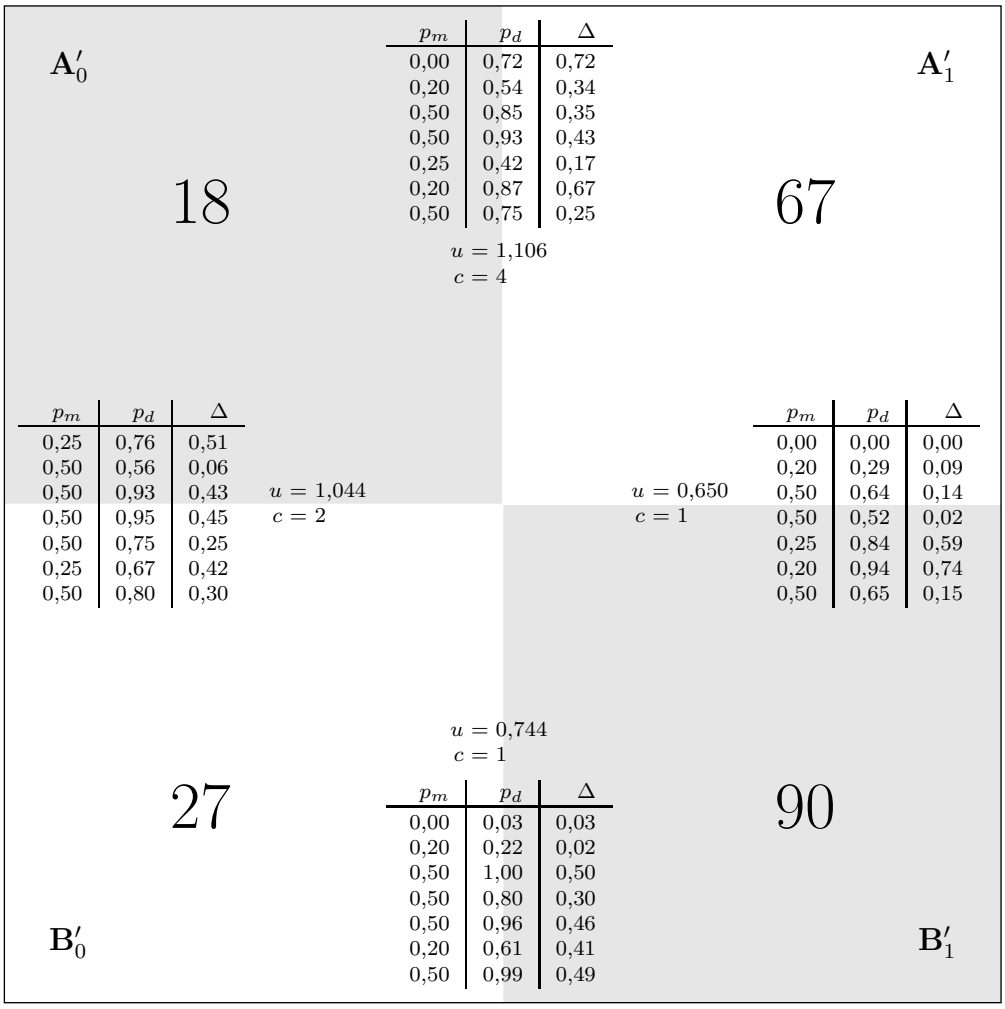
p_m	p_d	Δ
0,00	0,00	0,00
0,20	0,24	0,04
0,50	0,51	0,01
0,50	0,56	0,06
0,25	0,95	0,70
0,20	0,99	0,79
0,50	0,63	0,13

p_m	p_d	Δ
0,00	0,34	0,34
0,20	0,30	0,10
0,50	0,75	0,25
0,50	0,60	0,10
0,50	0,94	0,44
0,20	0,81	0,61
0,50	0,97	0,47

Slika 3.4: Rezultat preprostega algoritma, ki najprej nabere pare podobnih elementov z uporabo algoritma minimin na drevesu stanj do globine 1, nato pa izloči elemente, ki najbolj kvarijo podobnost med množicami.

raj hipno, sedaj pa že pri globini 1 algoritem rabi za izvajanje približno 67 sekund, pri globini 2 pa slabih 6 ur.

Rezultat izvajanja algoritma pri globini 1 (slika 3.4) kaže slabšo kvaliteto uparjenosti od izvirnega in od preprostega algoritma s požrešnim iskanjem. Kvaliteta uravnoteženosti množic je malo boljša od obeh omenjenih algoritmov, medtem ko je velikost celotne množice (206 elementov) malo slabša od preprostega algoritma s požrešnim iskanjem (210 elementov), vendar boljša



Slika 3.5: Rezultat preprostega algoritma, ki najprej nabere pare podobnih elementov z uporabo algoritma minimin na drevesu stanj do globine 2, nato pa izloči elemente, ki najbolj kvarijo podobnost med množicami.

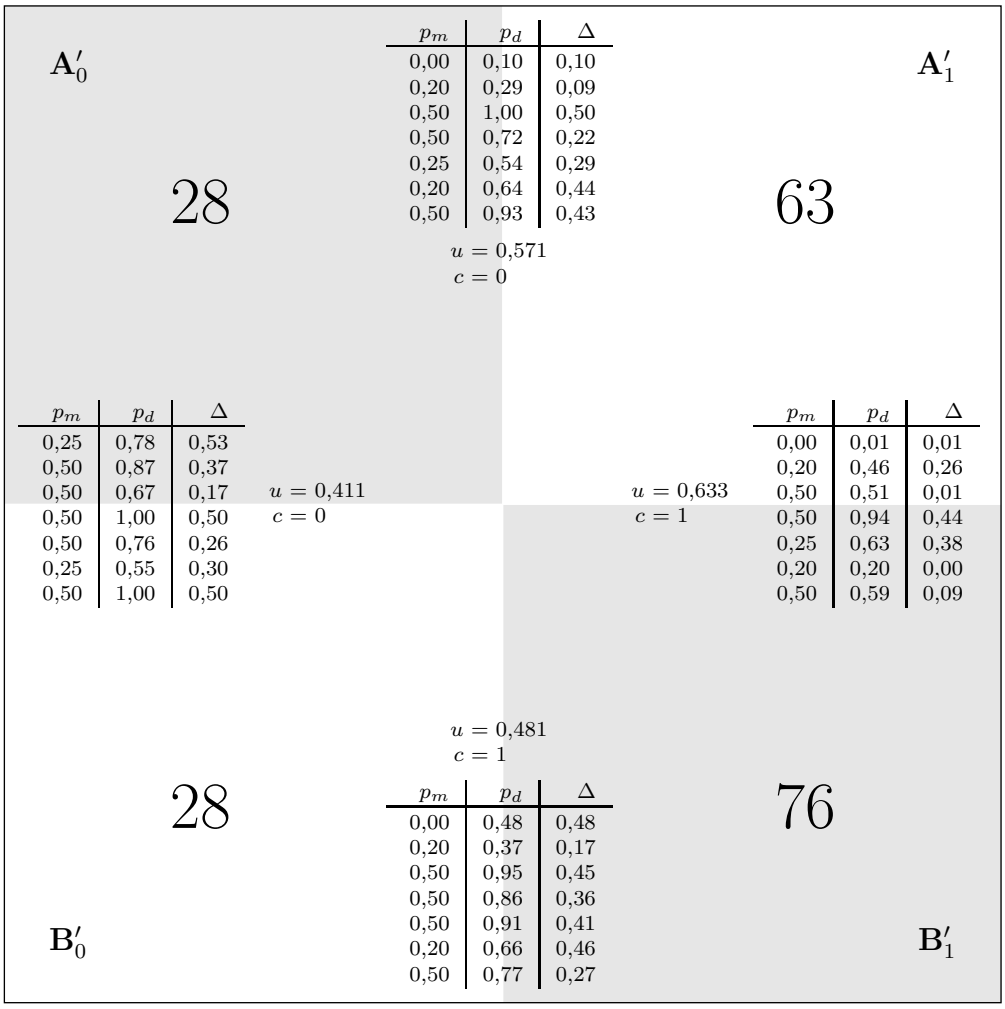
od izvirnega algoritma (180 elementov). Izstopajo manjši velikosti množic $\mathbf{A}'_0$ in $\mathbf{B}'_0$, ki sta še posebej kritični zaradi manj razpoložljivih elementov, ki jih lahko uvrstimo sem, ter precej slabša uparjenost množic. Da je uparjenost zelo slaba, nakazuje tudi dejstvo, da je v celotni množici kar 8 elementov, katerih minimalna podobnost presega prag podobnosti. To izhaja iz tega, da hevristična ocena, ki jo uporablja algoritem minimin, meri prostostno stopnjo stanj. S ciljem doseči čim večjo velikost celotne množice, hevristična

ocena dano stanje kaznuje glede na to koliko nadaljnjih stanj je možno izpeljati iz njega: manj kot je možnih stanj, večja je kazen. Kvaliteta uparjenosti se pri oceni ne upošteva, zato postane slabša. Pri globini 2 (slika 3.5) ne opazimo izboljšave, velikost končne množice se celo zmanjša. Na rezultatih testov enakega algoritma na vhodnih množicah, ki jih nismo uredili (sliki A.3 in A.4) pa nasprotno opazimo izboljšanje rezultatov s povečanjem globine iskanja. Do take razlike v rezultatih pride, ker hevristična ocena, ki je uporabljena pri algoritmu minimin, ne upošteva kvalitet uravnoteženosti in uparjenosti, kar privede do precejšnje občutljivosti na zaporedje elementov na vhodu napram požrešni metodi. Testov na večjih globinah iz praktičnih razlogov nismo opravili, saj bi kombinatorična zahtevnost preiskovanja tako globokih dreves bila prevelika.

3.5 Izvirni algoritem s požrešno strategijo

Izvirnemu algoritmu smo spremenili strategijo nabiranja parov podobnih elementov (poglavje 2.3.1) tako, da namesto prejšnjega nabiranja parov v istem zaporedju kakor se elementi pojavijo na vhodu, uparja elemente dveh množic s požrešno strategijo najbolj podobni najprej. Na koncu smo dodali še uravnoteževanje množic $\mathbf{A}'_1$ in $\mathbf{B}'_1$ z obojestranskim izbiranjem najboljših elementov za izločanje, opisano v poglavju 2.3.3.

Iz rezultata (slika 3.6) vidimo izboljšanje kvalitete uparjenosti glede na izvirni algoritem ter tudi večjo velikost množice (195 elementov) glede na velikost množice, ki je rezultat izvirnega algoritma (180 elementov). Kvaliteta uravnoteženosti med množicami je slabša, razen med množicama $\mathbf{A}'_1$ in $\mathbf{B}'_1$, kjer so sedaj vse p-vrednosti nad mejnimi. Slabšo uravnoteženost pripisujemo dejstvu, da je množica sedaj večja, kar pomeni, da je algoritem dodal več parov s slabo podobnostjo. Algoritem namreč še vedno deluje tako, da najprej upari enake elemente, šele nato gre iskat take pare podobnih elementov, ki ohranjajo uravnoteženost. Preprosti algoritem nima zadnje omejitve, ampak že od začetka nabira pare podobnih elementov brez ozira na njihov

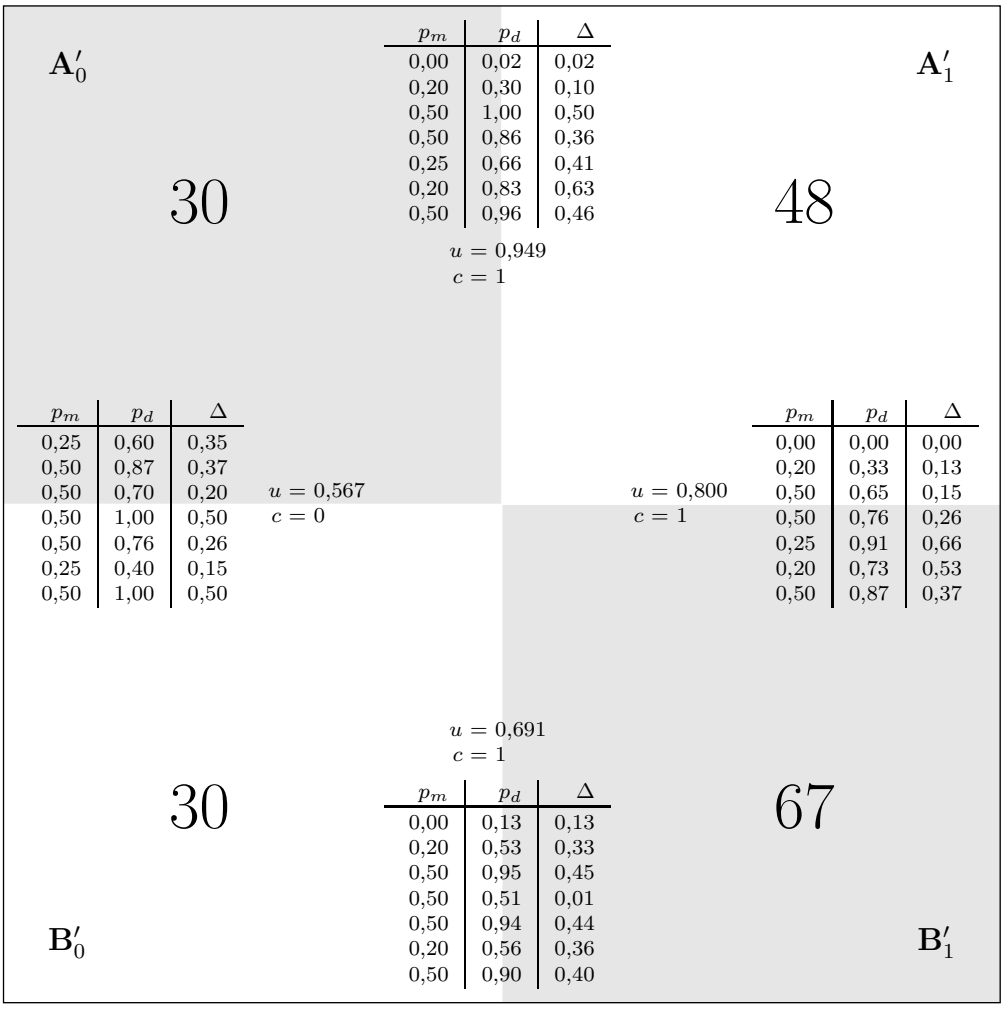


Slika 3.6: Rezultat izvirnega algoritma, ki izbira pare podobnih elementov s požrešno strategijo, ter na koncu dodatno uravnoteži še množici $\mathbf{A}_1$ in $\mathbf{B}_1$.

vpliv na uravnoteženost množic, zato nabere nekoliko več elementov, kar se kaže tudi v večji velikosti množice, ki jo naredi.

3.6 Izvirni algoritem in algoritem minimin

Podobno kot pri preprostem algoritmu, smo sedaj izvirnemu algoritmu spremenili strategijo nabiranja parov, da za izbiro naslednje poteze uporabi algoritem minimin (poglavje 2.3.2), ki preiskuje drevo stanj. Testirali smo



p_m	p_d	Δ
0,00	0,02	0,02
0,20	0,30	0,10
0,50	1,00	0,50
0,50	0,86	0,36
0,25	0,66	0,41
0,20	0,83	0,63
0,50	0,96	0,46

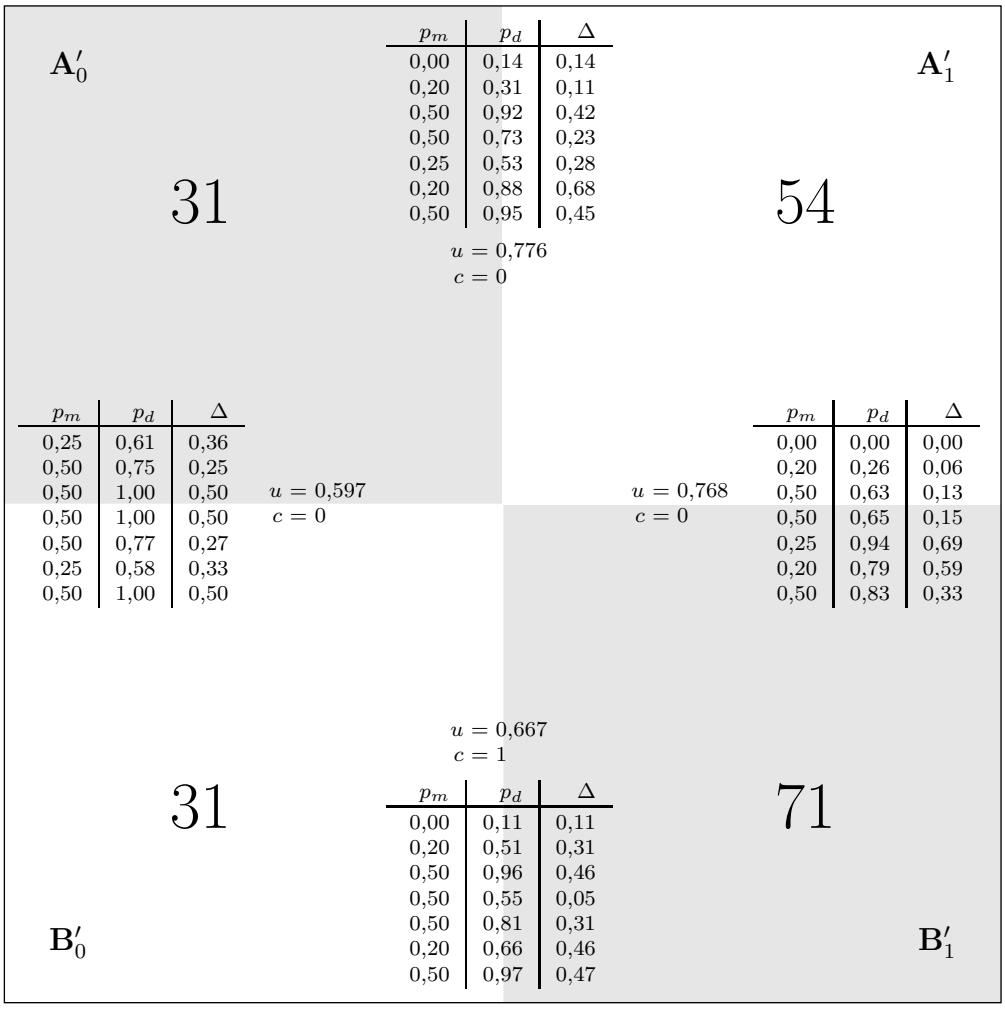
p_m	p_d	Δ
0,25	0,60	0,35
0,50	0,87	0,37
0,50	0,70	0,20
0,50	1,00	0,50
0,50	0,76	0,26
0,25	0,40	0,15
0,50	1,00	0,50

p_m	p_d	Δ
0,00	0,00	0,00
0,20	0,33	0,13
0,50	0,65	0,15
0,50	0,76	0,26
0,25	0,91	0,66
0,20	0,73	0,53
0,50	0,87	0,37

p_m	p_d	Δ
0,00	0,13	0,13
0,20	0,53	0,33
0,50	0,95	0,45
0,50	0,51	0,01
0,50	0,94	0,44
0,20	0,56	0,36
0,50	0,90	0,40

Slika 3.7: Rezultat izvirnega algoritma, ki izbira pare podobnih elementov z uporabo algoritma minimin na drevesu stanj do globine 1, ter na koncu dodatno uravnoteži še množici $\mathbf{A}'_1$ in $\mathbf{B}'_1$.

pri globinah preiskovanja 1 in 2. Pri globini 1 dobimo v rezultatu (slika 3.7) manjšo množico (175 elementov) kot pa pri istem algoritmu s požrešno strategijo (195 elementov). Tudi kvaliteti uparjenosti in uravnoteženosti sta slabši. Opazimo pa, da sta množici $\mathbf{A}'_0$ in $\mathbf{B}'_0$ večji za dva elementa. To gre na račun tega, da sta to prvi dve množici, ki ju algoritem uravnotežuje, začne pa tako, da najprej išče samo pare enakih elementov. V nadaljevanju poišče še take pare podobnih elementov, ki ne pokvarijo uravnoteženosti obeh



Slika 3.8: Rezultat izvirnega algoritma, ki izbira pare podobnih elementov z uporabo algoritma minimin na drevesu stanj do globine 2, ter na koncu dodatno uravnoteži še množici $\mathbf{A}'_1$ in $\mathbf{B}'_1$.

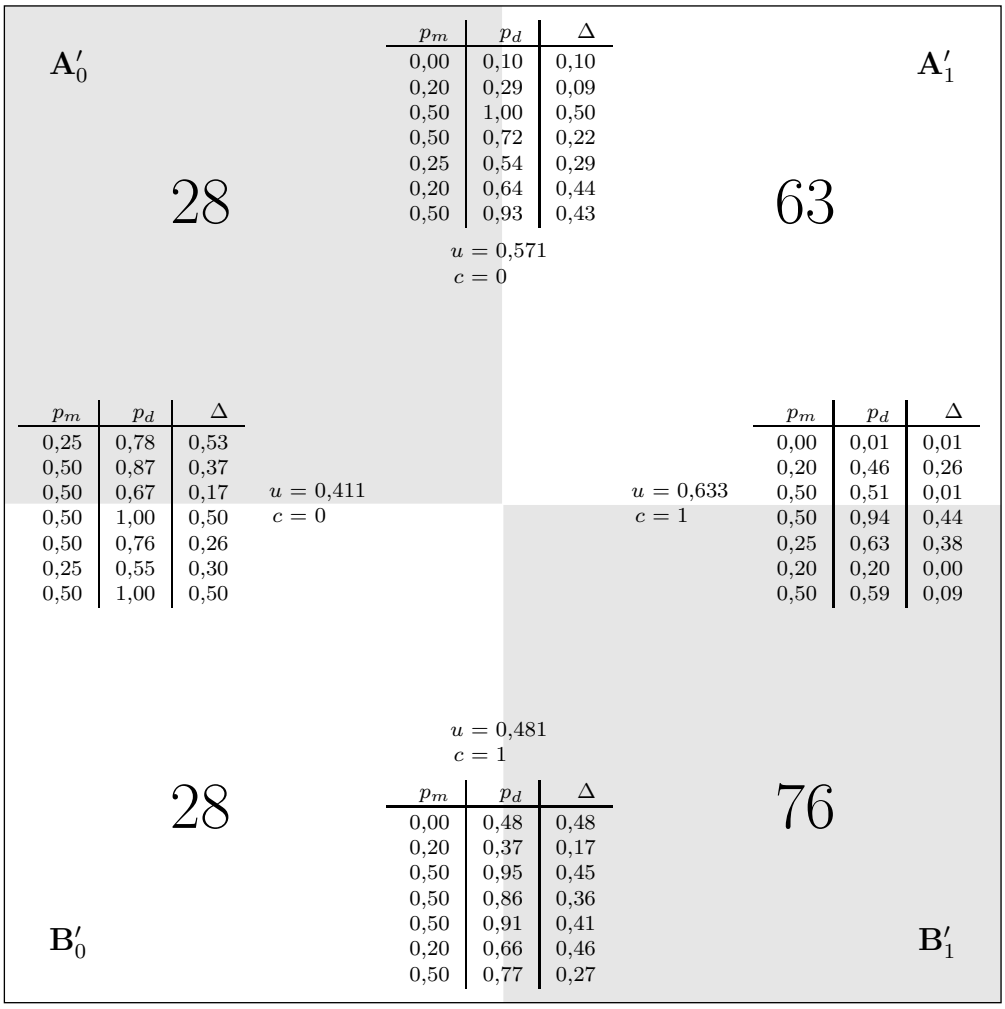
množic. Kaže, da najprej nabiranje enakih, nato pa še podobnih elementov z omejitvijo glede uravnoteženosti, tukaj povzroči, da se algoritem minimin pri uravnoteževanju obeh začetnih množic ne izkaže za tako slabega, kot pri preprostem algoritmu. Med množicama $\mathbf{A}_0$ in $\mathbf{B}_0$ je 17 parov enakih elementov, zato algoritmu ni težko pri dani omejitvi poiskati še nekaj parov podobnih elementov, da pride do boljšega rezultata. Pri preprostem algoritmu pa ni začetnega iskanja enakih in naknadnega dodajanja parov podobnih ele-

mentov, ampak zgolj išče pare brez omejitve glede uravnoteženosti, kar da preprostemu algoritmu več prostora, da ,,pokvari“ rezultat. V nadaljevanju algoritem sicer nabere približno toliko elementov za množici $\mathbf{A}'_1$ in $\mathbf{B}'_1$, kot algoritem s požrešno strategijo, vendar zaradi drugačne strategije nabiranja elementov na koncu izloči več elementov. Požrešni algoritem z nabiranjem parov, ki imajo podobnost pod pragom, posredno pazi na kvaliteto uparjenosti, medtem ko algoritem minimin pazi samo na optimizacijo prostostnih stopenj potez, ki jih ubira. Posledično algoritem nabere več ,,kvarljivih“ parov, ki jih mora na koncu izločiti. Pri globini 2 algoritem sestavi večjo množico (187 elementov) z izboljšanima kvalitetama uparjenosti in uravnoteženosti, vendar je ta velikost še vedno manjša od algoritma s požrešno strategijo iskanja parov. Velikost množice pri tej globini je večja tudi od velikosti množice, ki jo naredi izvirni algoritem (180 elementov). Kvaliteti uparjenosti in uravnoteženosti sta pravtako izboljšani.

3.7 Požrešna strategija in splošno izločanje

Preprosti algoritem (poglavje 2.3.3) na koncu izloča elemente na vseh množicah obenem, da jih uravnoteži. Izvirni algoritem izloča elemente samo pri uravnoteževanju množic $\mathbf{A}'_1$ z $\mathbf{A}'_0$ ter $\mathbf{B}'_1$ z $\mathbf{B}'_0$. Hkrati z zamenjavami strategij nabiranja parov smo izvirnemu algoritmu dodali še medsebojno uravnotežitev množic $\mathbf{A}'_1$ in $\mathbf{B}'_1$. V omenjenih primerih nismo medsebojno uravnoteževali še množic $\mathbf{A}'_0$ in $\mathbf{B}'_0$, ker sta ti dve množici zaradi svoje majhnosti kritični. Poleg tega sam algoritem, ki sestavi ti dve množici, že zagotavlja, da se z vsakim dodatnim parom elementov uravnoteženost ohrani. Sedaj nas je pa zanimalo, če lahko z uporabo izvirnega algoritma z obema strategijama nabiranja parov (s požrešno metodo in z algoritmom minimin) pridobimo večjo velikost končnih množic, če smo pripravljeni žrtvovati kakšen element manjših množic z uporabo splošnega izločanja elementov na koncu, kot ga uporablja preprosti algoritem.

Rezultat izvirnega algoritma s požrešno metodo nabiranja parov podob-



p_m	p_d	Δ
0,00	0,10	0,10
0,20	0,29	0,09
0,50	1,00	0,50
0,50	0,72	0,22
0,25	0,54	0,29
0,20	0,64	0,44
0,50	0,93	0,43

p_m	p_d	Δ
0,25	0,78	0,53
0,50	0,87	0,37
0,50	0,67	0,17
0,50	1,00	0,50
0,50	0,76	0,26
0,25	0,55	0,30
0,50	1,00	0,50

p_m	p_d	Δ
0,00	0,01	0,01
0,20	0,46	0,26
0,50	0,51	0,01
0,50	0,94	0,44
0,25	0,63	0,38
0,20	0,20	0,00
0,50	0,59	0,09

p_m	p_d	Δ
0,00	0,48	0,48
0,20	0,37	0,17
0,50	0,95	0,45
0,50	0,86	0,36
0,50	0,91	0,41
0,20	0,66	0,46
0,50	0,77	0,27

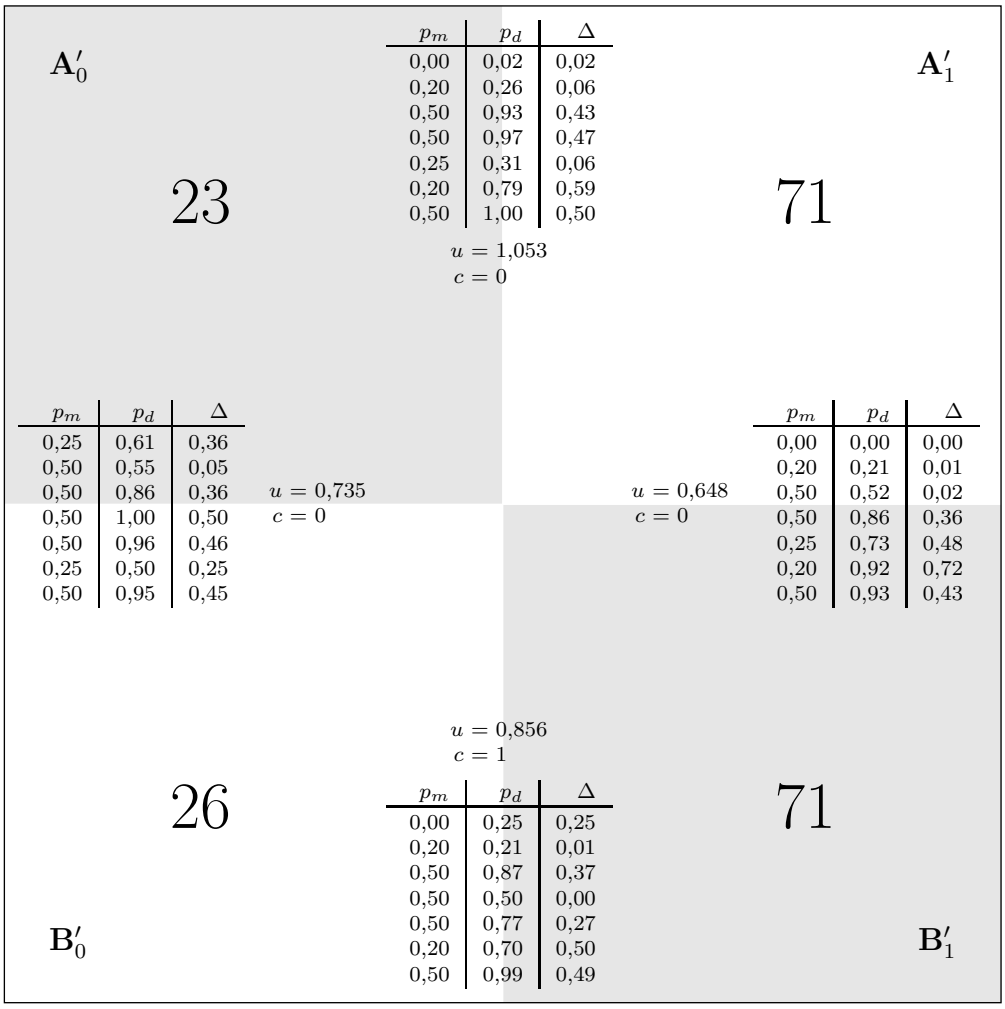
Slika 3.9: Rezultat izvirnega algoritma, ki izbira pare podobnih elementov s požrešno strategijo, ter na koncu množice uravnoteži z izločanjem elementov na vseh množicah hkrati.

nih elementov in s splošnim izločanjem elementov (slika 3.9) pokaže popolnoma enak rezultat, kot smo ga dobili z izvirnim algoritmom, ki je zgolj ločeno uravnotežil $\mathbf{A}'_1$ z $\mathbf{A}'_0$, $\mathbf{B}'_1$ z $\mathbf{B}'_0$, ter obojestransko $\mathbf{A}'_1$ z $\mathbf{B}'_1$. Po eni strani tak rezultat ni posebej presenetljiv, saj omejitve pri nabiranju parov elementov za množici $\mathbf{A}'_0$ in $\mathbf{B}'_0$ zagotavljajo medsebojno uravnoteženost teh dveh množic. Po drugi strani pa to pomeni, da je algoritem slučajno sestavil množici $\mathbf{A}'_1$ in $\mathbf{B}'_1$ tako, da se splošno izločanje ni dotaknilo njunih nasprotnih

množic. Zaporedoma izločeni elementi so na splošno uravnoteženost vplivali tako, da se je naslednji najboljši element za izločitev vedno nahajal v eni od množic $\mathbf{A}'_1$ in $\mathbf{B}'_1$. Da je možen tudi drugačen potek, vidimo iz rezultata na sliki A.8, kjer vhodni podatki predhodno niso bili sortirani. Tam je izločanje žrtvovalo dva elementa iz $\mathbf{A}'_0$ in en element iz $\mathbf{B}'_0$ na račun povečanja množic $\mathbf{A}'_1$ na 66 in $\mathbf{B}'_1$ na 79 elementov. Vendar ocenjujemo, da povečanje celotne množice iz 195 na 198 elementov ni vredno žrtvovanja treh elementov iz manjših dveh množic.

3.8 Algoritem minimin in splošno izločanje

Poleg požrešne metode smo izvirni algoritem s splošnim izločanjem testirali tudi s strategijo nabiranja parov podobnih elementov z algoritmom minimin na drevesu stanj. Izločanje elementov iz manjših dveh množic je tukaj bolj izrazito, saj se uporabljena strategija nabiranja parov podobnih elementov ne ozira na kvaliteto njihove podobnosti. To v splošnem privede do manjše uravnoteženosti prvih dveh množic $\mathbf{A}'_0$ in $\mathbf{B}'_0$ in v nadaljevanju do večje neuravnoteženosti množic $\mathbf{A}'_1$ in $\mathbf{B}'_1$. Algoritem minimin z mejo na globini 1 privede do slabšega rezultata (slika 3.10), kot z mejo na globini 2 (slika 3.11). Pri globini 1 je splošno izločanje zaradi slabše kvalitete uparjanja žrtvovalo 7 elementov iz manjših množic $\mathbf{A}'_0$ in $\mathbf{B}'_0$, da je velikost celotne množice dosegla 191 elementov. Pri globini 2 pa je algoritem žrtvoval 4 elemente na račun tega, da ima celotna množica 204 elemente. Slednja žrtev bi glede na dober končni rezultat morebiti še bila sprejemljiva, vendar nas moti občutljivost algoritma na zaporedje vhodnih podatkov. Enak algoritem z obema globinama smo zagnali tudi na podatkih, ki jih predhodno nismo uredili. Dobili smo precej slabše rezultate (sliki A.9 in A.10), kjer algoritem žrtvuje že nesprejemljivo število elementov iz manjših dveh množic in doseže bistveno slabši končni rezultat.



p_m	p_d	Δ
0,00	0,02	0,02
0,20	0,26	0,06
0,50	0,93	0,43
0,50	0,97	0,47
0,25	0,31	0,06
0,20	0,79	0,59
0,50	1,00	0,50

p_m	p_d	Δ
0,25	0,61	0,36
0,50	0,55	0,05
0,50	0,86	0,36
0,50	1,00	0,50
0,50	0,96	0,46
0,25	0,50	0,25
0,50	0,95	0,45

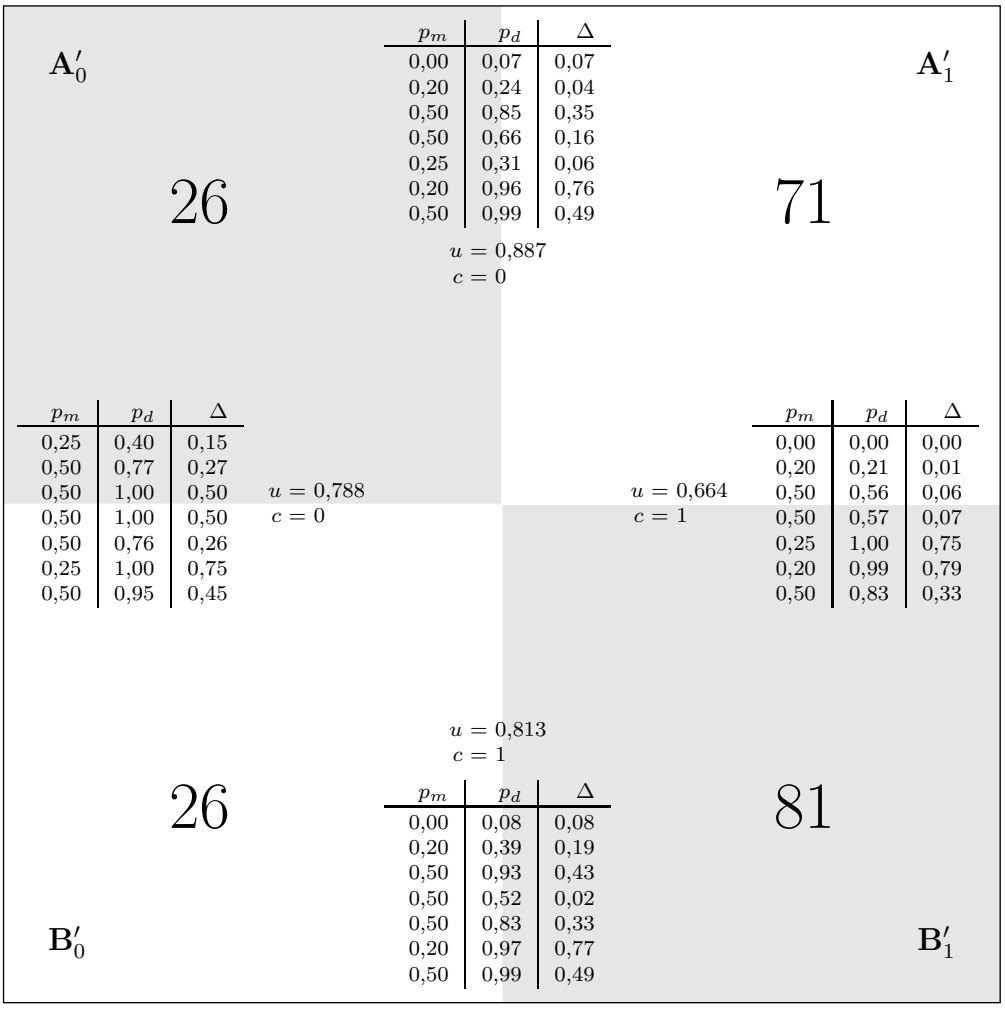
p_m	p_d	Δ
0,00	0,00	0,00
0,20	0,21	0,01
0,50	0,52	0,02
0,50	0,86	0,36
0,25	0,73	0,48
0,20	0,92	0,72
0,50	0,93	0,43

p_m	p_d	Δ
0,00	0,25	0,25
0,20	0,21	0,01
0,50	0,87	0,37
0,50	0,50	0,00
0,50	0,77	0,27
0,20	0,70	0,50
0,50	0,99	0,49

Slika 3.10: Rezultat izvirnega algoritma, ki izbira pare podobnih elementov z algoritmom minimin na drevesu stanj do globine 1, ter na koncu množice uravnoteži z izločanjem elementov na vseh množicah hkrati.

3.9 Vpliv urejenosti vhodnih podatkov

Algoritme z drugačnimi strategijami uravnoteževanja smo pognali tudi na vhodnih podatkih, ki jih predhodno nismo uredili. Rezultate podajamo v dodatku A. Opaziti je, da so se spremenili vsi rezultati, nekateri na slabše, drugi na boljše. Najbolj na slabše so se spremenili rezultati pri preprostem algoritmu, ki za uparjanje uporablja algoritem minimin, ter pri naivnem



Slika 3.11: Rezultat izvirnega algoritma, ki izbira pare podobnih elementov z algoritmom minimin na drevesu stanj do globine 2, ter na koncu množice uravnoteži z izločanjem elementov na vseh množicah hkrati.

algoritmu.

Preprosti algoritem z algoritmom minimin je najbolj občutljiv na zaporedje vhodnih rezultatov, saj uravnoteženost in podobnost ne vplivata na zaporedje nabiranja parov. V našem primeru so slučajno elementi vhodnih podatkov razporejeni v takem vrstnem redu, da algoritem na njih izračuna slabši rezultat, kot pa na vhodnih podatkih, ki jih najprej uredimo po vrednostih karakteristik. Požrešna metoda je, s svojo strategijo izbire najbolj

podobnega elementa v vsaki iteraciji, bolj odporna na različno zaporedje vhodnih podatkov. Vendar so v našem primeru dopustne podobnosti, po katerih dela požrešna metoda selekcijo, samo tri cela števila: 0, 1 in 2. To pomeni, da ima veliko parov enako podobnost, kar tudi pri požrešni metodi povzroči različno zaporedje nabiranja parov pri različnih zaporedjih vhodnih podatkov. Slednje sicer ne vpliva na uparjenost množic, vpliva pa na uravnoteženost, saj dva para z enako podobnostjo nimata nujno enakih razlik oz. enakosti po posameznih karakteristikah.

Podobno velja tudi za algoritem izločanja elementov, ki imajo največji doprinos k neuravnoteženosti množic. Le-ta se odloča na podlagi hevristične ocene (enačba 2.6), ki meri skupen doprinos vseh karakteristik elementa k neuravnoteženosti množic. Ker ni pomembno katere karakteristike to naredijo, se lahko zgodi, da obstajata dva neenaka elementa, ki imata enak maksimalen doprinos k neuravnoteženosti. Odvisno od vrstnega reda elementov v množici je potem, kateri element bo algoritem izbral, da ga izloči.

Poglavje 4

Zaključki

Ukvarjali smo se z uravnoteževanjem štirih množic na realnih podatkih 655 bolnikov z rakom dojke [6], s ciljem pridobiti čim večje uravnotežene množice ob podanih omejitvah. Bolniki so bili razdeljeni najprej glede na način zdravljenja, nato pa še vsakič glede na vrednost binarne karakteristike uPA/PAI-1. Problem je bil precej težek, saj so si bili bolniki po svojih karakteristikah dokaj različni, poleg tega pa je omenjena karakteristika znotraj vsakega zdravljenja povzročila nastanek dveh precej neenako velikih množic. Za eno zdravljenje smo dobili podmnožici velikosti 78 in 208 elementov, za drugo zdravljenje pa podmnožici velikosti 80 in 289 elementov. Zaradi precejšnje majhnosti dveh množic bolnikov je izvirni algoritem [3] bil razvit tako, da najprej uparja elemente obeh majhnih množic. Nato ti dve množici uporabi kot osnovi, h katerima ločeno za vsako majhno množico upari elemente iz pripadajoče večje množice v razmerju 1:3 in jih nato uravnoteži.

Z namero izboljšanja rezultatov izvirnega algoritma smo sestavili nove strategije uparjanja elementov za sestavljanje množic in izločanja elementov za uravnoteževanje množic. Dodatno smo definirali še mero za kvaliteto uparjenosti dveh množic, da lahko med rezultati poleg uravnoteženosti primerjamo še splošno kvaliteto uparjenosti elementov.

Iz rezultatov je takoj razvidno, da je daleč najslabši naivni pristop, ko preprosto vzamemo vse elemente množic in po vrsti izločamo take, katerih

izločitev povzroči največji prispevek v smeri uravnoteženosti. Po drugi strani dobimo vidno boljši rezultat, če izvirnemu algoritmu zgolj spremenimo strategijo uparjanja elementov, da ne uparja elementov po vhodnem vrstnem redu, ampak uporablja požrešno metodo, kjer v vsaki iteraciji vzame najbolj podoben par elementov. Hkrati se je zaradi uporabe požrešne metode algoritem izkazal za manj občutljivega na različno urejenost vhodnega zaporedja elementov. Nasprotno se je strategija uparjanja z uporabo algoritma minimin izkazala za občutljivo na vrstni red elementov na vhodu. Vzrok temu je, da ta algoritem pri preiskovanju uporablja hevristično oceno, ki ne upošteva kvalitet uparjanja in uravnoteženosti, ampak upošteva zgolj število možnih nadaljnjih stanj. To dejstvo botruje tudi temu, da s to metodo pri globinah preiskovanj, ki so zaradi časovne zahtevnosti še praktične, v splošnem pridemo do slabih rezultatov.

Ugotovili smo, da je algoritem minimin zelo občutljiv na vrstni red vhodnih podatkov. Hevristična ocena, ki jo uporablja, ne upošteva kvalitet uravnoteženosti in uparjenosti, zato lahko naknadno uravnoteževanje z izločanjem elementov precej zmanjša množice, ki smo jih pridobili z uparjanjem. Primeren vrstni red vhodnih podatkov lahko privede do rezultatov, ko je rezultat z manjšo globino preiskovanja boljši, kot rezultat preiskovanja pri večji globini. Na tak primer smo naleteli v rezultatih preprostega algoritma, ki je za uparjanje uporabil algoritem minimin. Ko vhodnih podatkov nismo uredili, smo pri globini 2 dobili boljši rezultat od rezultata pri globini 1 (sliki A.3 in A.4), medtem ko smo na urejenih vhodnih podatkih dobili pri globini 2 slabši rezultat, kot pa pri globini 1 (sliki 3.4 in 3.5). Nasprotno požrešna metoda delno uredi pare podobnih elementov po njihovi podobnosti, preden jih izbere, zato ni toliko dovzetna na vrstni red vhodnih podatkov.

Za najboljšega se je izkazal preprosti algoritem, ki na začetku brez omejitev požrešno nabere pare podobnih elementov, nato pa na vseh množicah obenem zaporedoma izloča elemente, katerih izločitev najbolj doprinese k splošni uravnoteženosti množic. Od izvirnega se loči samo v dveh delih: pri sestavljanju prvih (majhnih) dveh množic se ne ozira na njuno medsebojno

uravnoteženost, na koncu pa medsebojno uravnotežuje vse množice obenem. Uporaba požrešne metode ima še to dobro lastnost, da je algoritem manj občutljiv na to, v kakšnem zaporedju dobi vhodne podatke. S tem je, od vseh predstavljenih, ta algoritem najbolj primeren za uporabo v raziskavah.

Zanimiv izziv za nadaljnje delo bi bil sestaviti hevristično oceno stanj, ki poleg števila možnih nadaljnjih stanj upošteva tudi spremembo kvalitete uravnoteženosti in/ali uparjenosti množic, do katere bi privedel prehod v novo stanje. Taka hevristika, ki upošteva vse tri kvalitete (velikost, uparjenost in uravnoteženost množic), bi bila logična izbira za uporabo z algoritmom A*.

Literatura

[1] J. M. Lachin, J. P. Matts in L. Wei, "Randomization in clinical trials: Conclusions and recommendations," *Controlled Clinical Trials*, zv. 9, št. 4, str. 365–374, 1988.

[2] P. R. R. Xing Sam Gu, "Comparison of multivariate matching methods: Structures, distances in algorithms," *Journal of Computational and Graphical Statistics*, zv. 2, št. 4, str. 405–420, 1993.

[3] A. Sadikov, "Distribution balancing of retrospective data," tehn. poročilo, University of Ljubljana, Faculty of Computer and Information Science, Večna pot 113, SI-1000 Ljubljana, Slovenia, 2009.

[4] K. P. F.R.S., "On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *Philosophical Magazine Series 5*, zv. 50, št. 302, str. 157–175, 1900.

[5] F. Yates, "Contingency tables involving small numbers and the χ^2 test," *Supplement to the Journal of the Royal Statistical Society*, zv. 1, št. 2, str. 217–235, 1934.

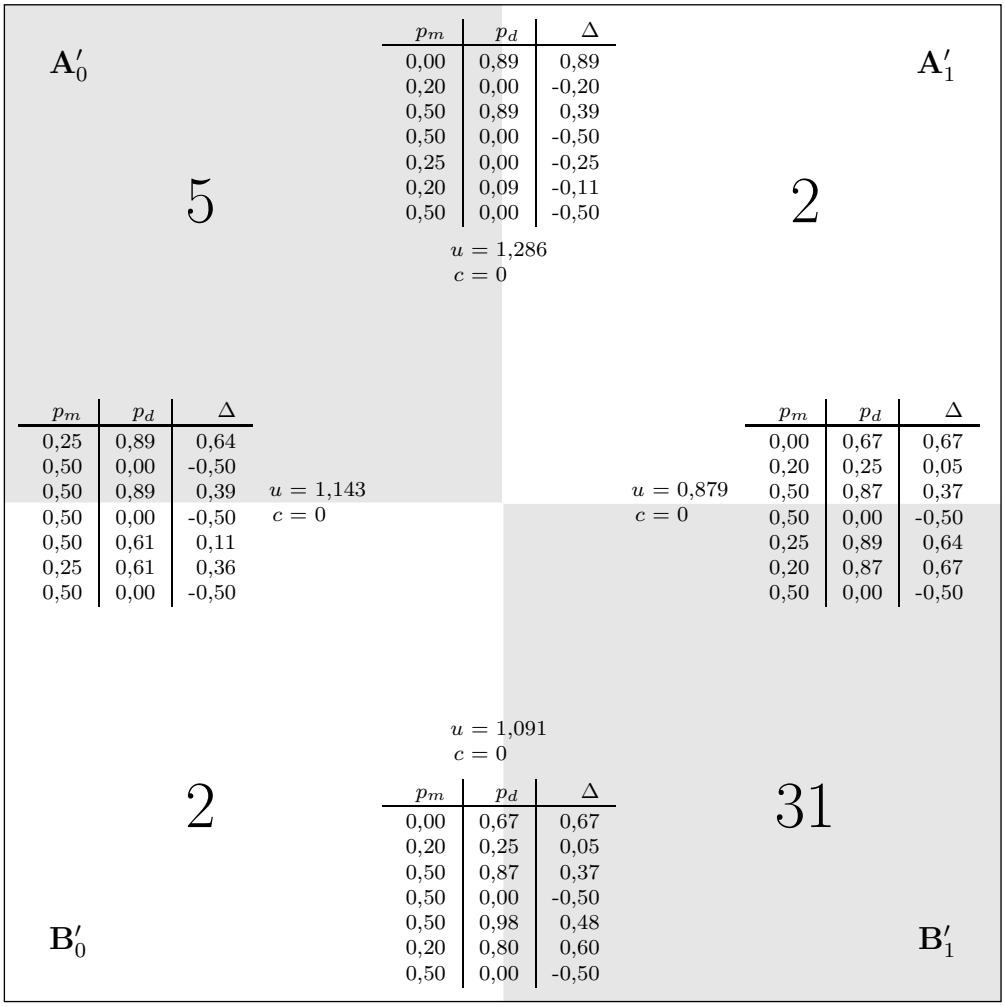
[6] S. Borštnar, A. Sadikov, B. Možina in T. Čufer, "High levels of uPA and PAI-1 predict a good response to anthracyclines," *Breast Cancer Research and Treatment*, zv. 121, št. 3, str. 615–624, 2010.

[7] R. E. Korf, "Real-time heuristic search," *Artificial Intelligence*, zv. 42, št. 2, str. 189 – 211, 1990.

Dodatek A

Rezultati na neurejenih množicah

V nadaljevanju podajamo rezultate algoritmov z dodatnimi strategijami, kjer vhodne množice pred zagonom algoritmov nismo sortirali.



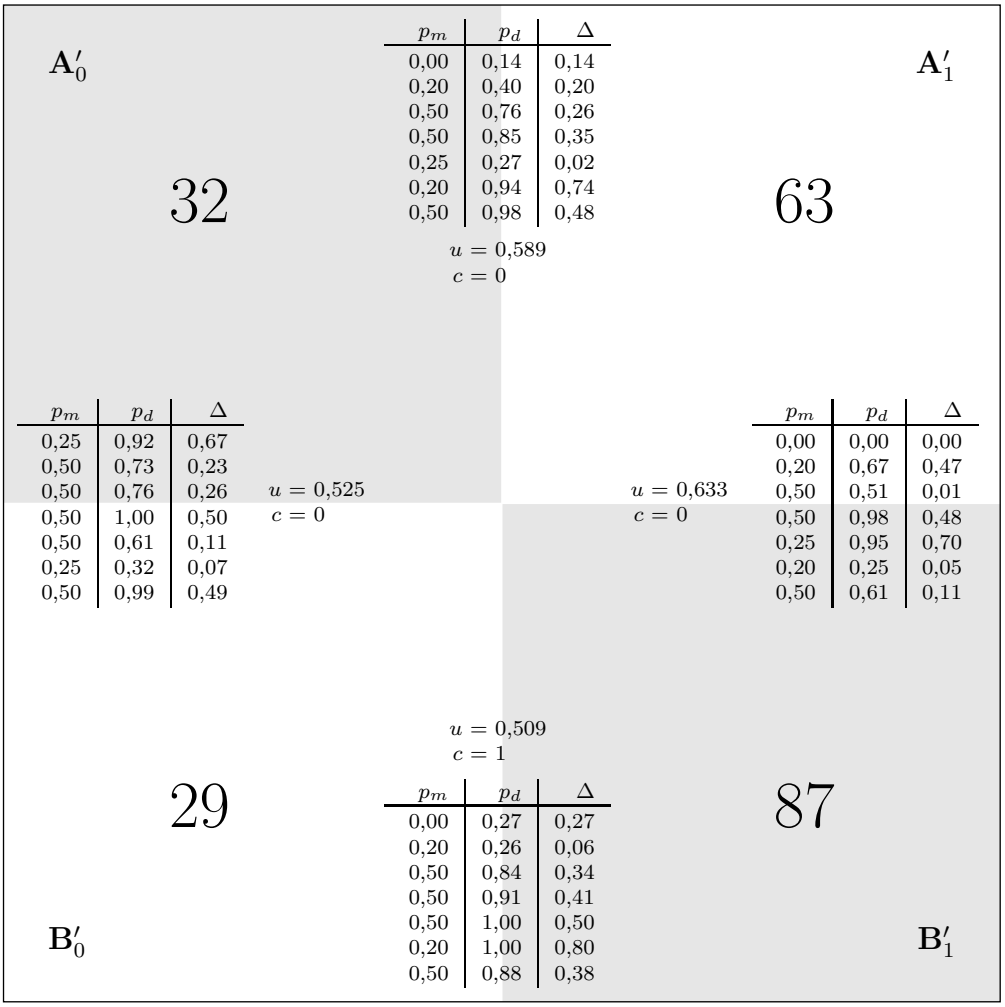
p_m	p_d	Δ
0,00	0,89	0,89
0,20	0,00	-0,20
0,50	0,89	0,39
0,50	0,00	-0,50
0,25	0,00	-0,25
0,20	0,09	-0,11
0,50	0,00	-0,50

p_m	p_d	Δ
0,25	0,89	0,64
0,50	0,00	-0,50
0,50	0,89	0,39
0,50	0,00	-0,50
0,50	0,61	0,11
0,25	0,61	0,36
0,50	0,00	-0,50

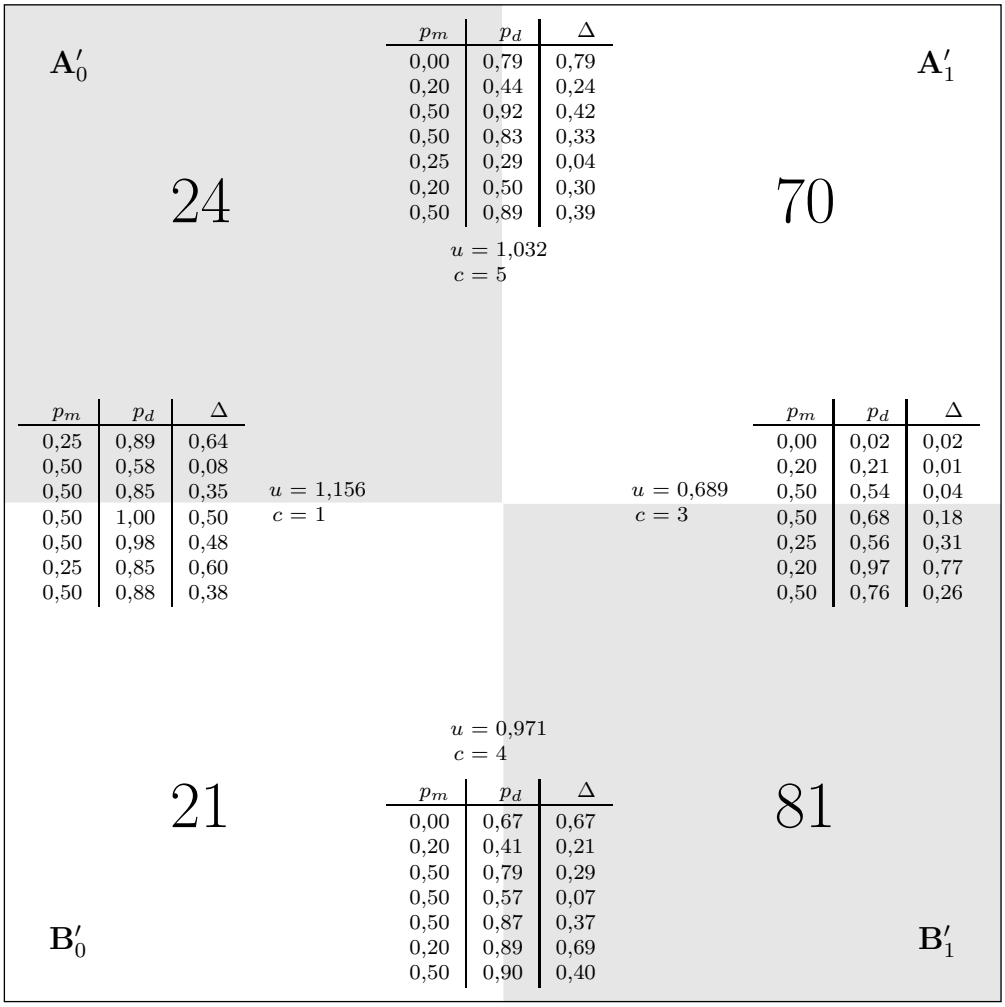
p_m	p_d	Δ
0,00	0,67	0,67
0,20	0,25	0,05
0,50	0,87	0,37
0,50	0,00	-0,50
0,25	0,89	0,64
0,20	0,87	0,67
0,50	0,00	-0,50

p_m	p_d	Δ
0,00	0,67	0,67
0,20	0,25	0,05
0,50	0,87	0,37
0,50	0,00	-0,50
0,50	0,98	0,48
0,20	0,80	0,60
0,50	0,00	-0,50

Slika A.1: Rezultat naivnega algoritma, ki samo izloči elemente, ki najbolj kvarijo uravnoteženost in uparjenost množic.



Slika A.2: Rezultat preprostega algoritma, ki najprej s požrešno strategijo nabere pare podobnih elementov, nato pa izloči elemente, ki najbolj kvarijo uravnoteženost množic.



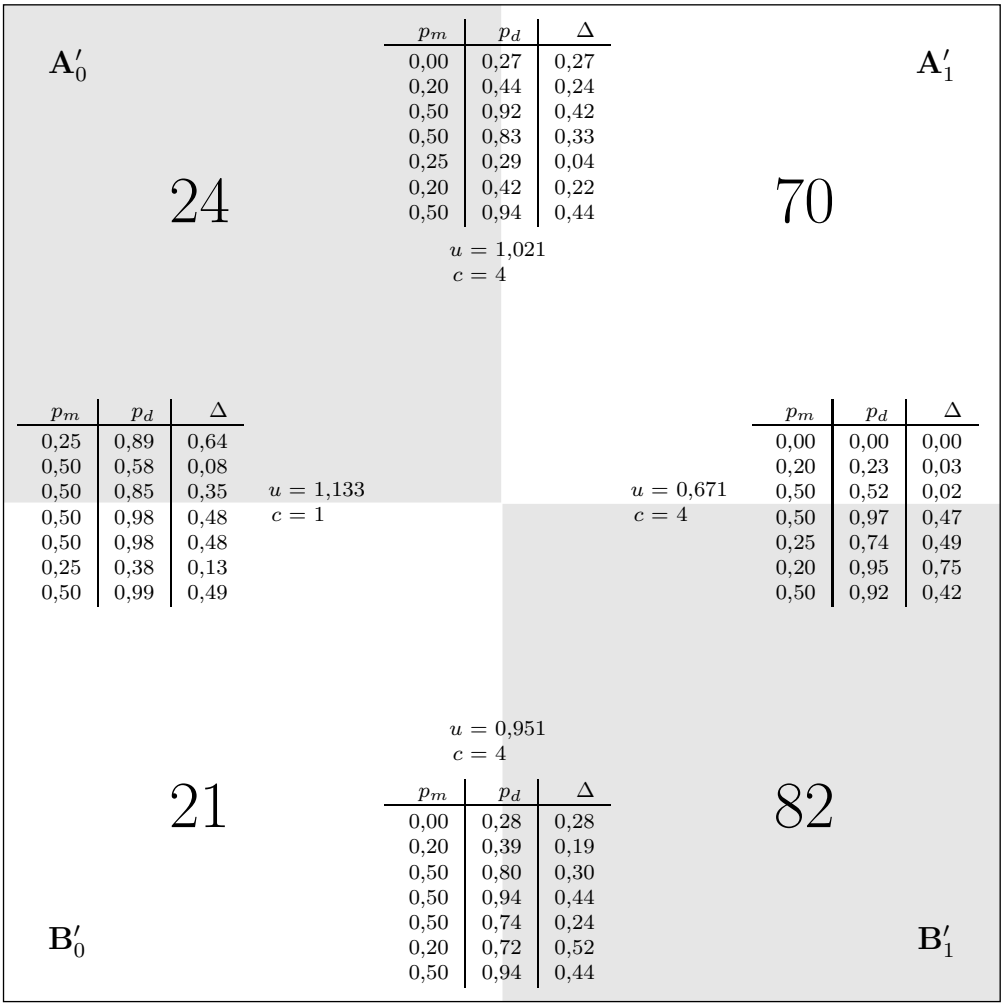
p_m	p_d	Δ
0,00	0,79	0,79
0,20	0,44	0,24
0,50	0,92	0,42
0,50	0,83	0,33
0,25	0,29	0,04
0,20	0,50	0,30
0,50	0,89	0,39

p_m	p_d	Δ
0,25	0,89	0,64
0,50	0,58	0,08
0,50	0,85	0,35
0,50	1,00	0,50
0,50	0,98	0,48
0,25	0,85	0,60
0,50	0,88	0,38

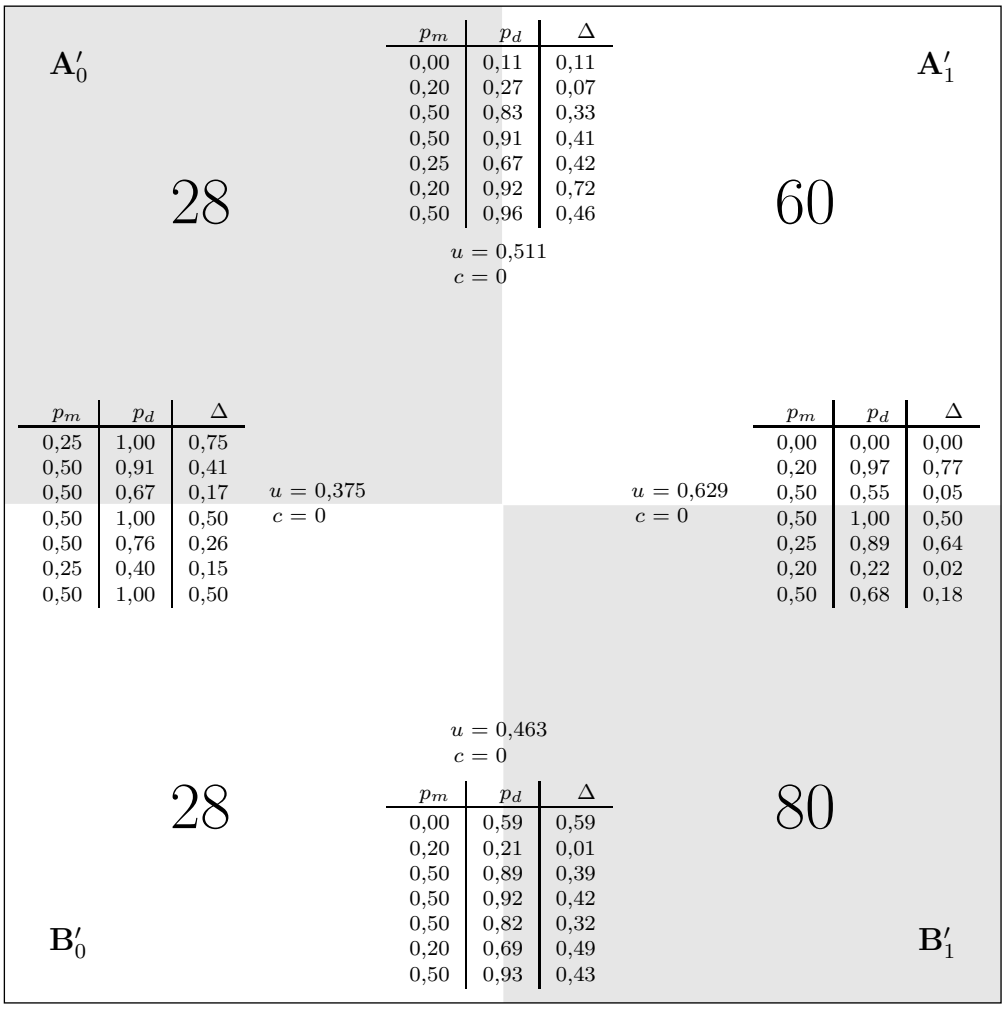
p_m	p_d	Δ
0,00	0,02	0,02
0,20	0,21	0,01
0,50	0,54	0,04
0,50	0,68	0,18
0,25	0,56	0,31
0,20	0,97	0,77
0,50	0,76	0,26

p_m	p_d	Δ
0,00	0,67	0,67
0,20	0,41	0,21
0,50	0,79	0,29
0,50	0,57	0,07
0,50	0,87	0,37
0,20	0,89	0,69
0,50	0,90	0,40

Slika A.3: Rezultat preprostega algoritma, ki najprej nabere pare podobnih elementov z uporabo algoritma minimin na drevesu stanj do globine 1, nato pa izloči elemente, ki najbolj kvarijo podobnost med množicami.



Slika A.4: Rezultat preprostega algoritma, ki najprej nabere pare podobnih elementov z uporabo algoritma minimin na drevesu stanj do globine 2, nato pa izloči elemente, ki najbolj kvarijo podobnost med množicami.



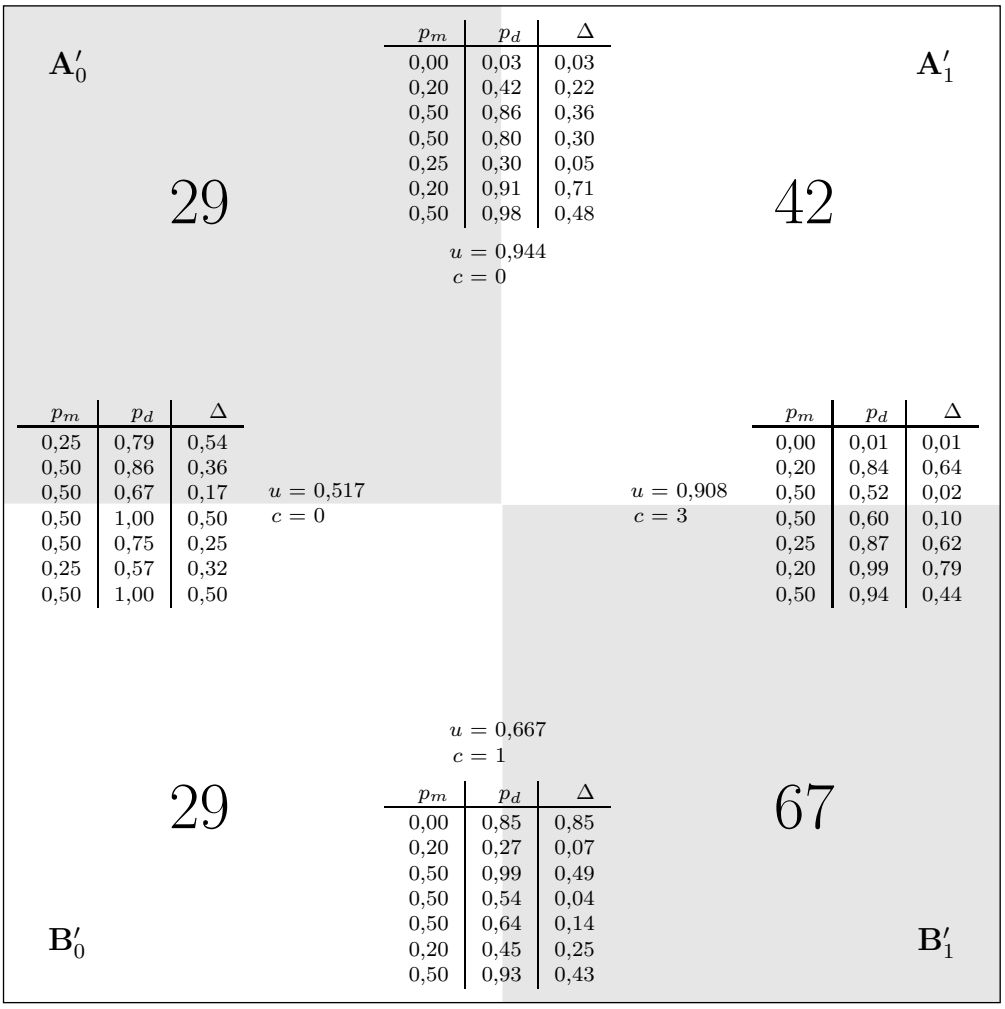
p_m	p_d	Δ
0,00	0,11	0,11
0,20	0,27	0,07
0,50	0,83	0,33
0,50	0,91	0,41
0,25	0,67	0,42
0,20	0,92	0,72
0,50	0,96	0,46

p_m	p_d	Δ
0,25	1,00	0,75
0,50	0,91	0,41
0,50	0,67	0,17
0,50	1,00	0,50
0,50	0,76	0,26
0,25	0,40	0,15
0,50	1,00	0,50

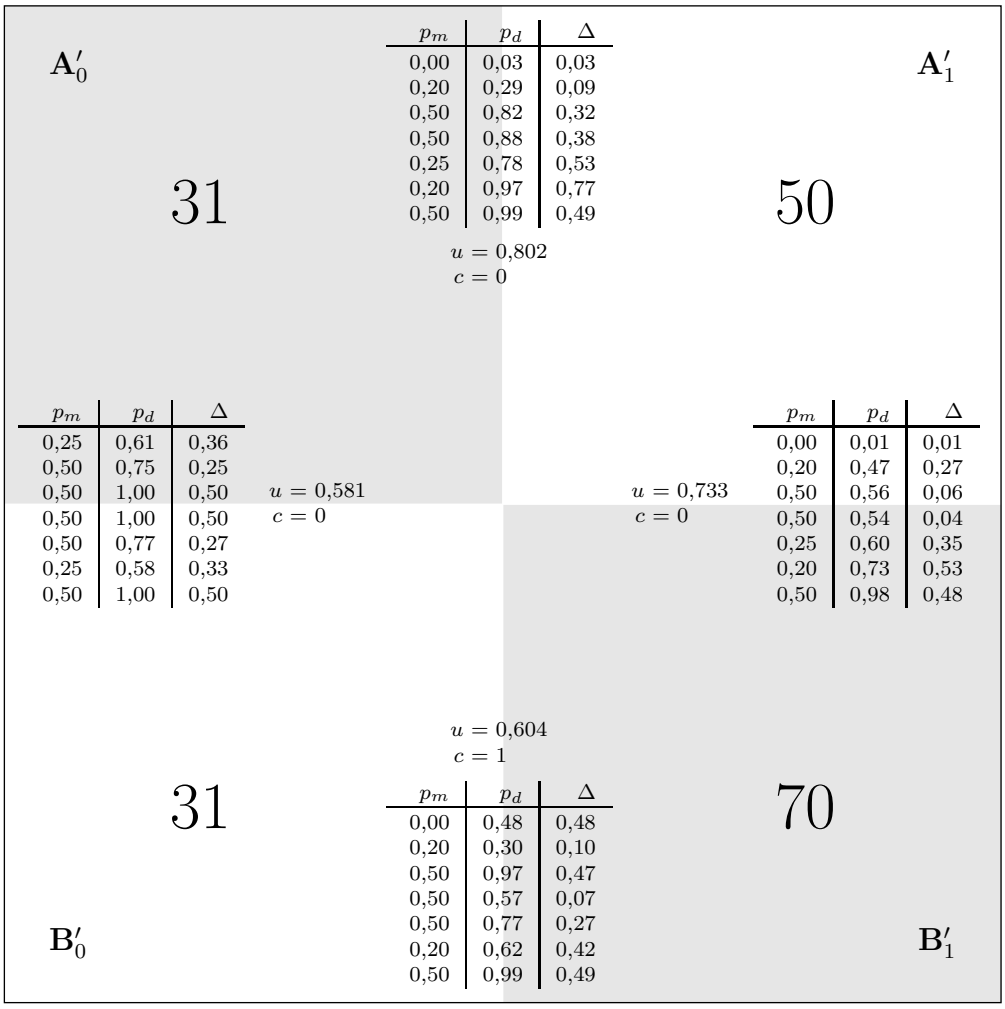
p_m	p_d	Δ
0,00	0,00	0,00
0,20	0,97	0,77
0,50	0,55	0,05
0,50	1,00	0,50
0,25	0,89	0,64
0,20	0,22	0,02
0,50	0,68	0,18

p_m	p_d	Δ
0,00	0,59	0,59
0,20	0,21	0,01
0,50	0,89	0,39
0,50	0,92	0,42
0,50	0,82	0,32
0,20	0,69	0,49
0,50	0,93	0,43

Slika A.5: Rezultat izvirnega algoritma, ki izbira pare podobnih elementov s požrešno strategijo, ter na koncu dodatno uravnoteži še množici $\mathbf{A}'_1$ in $\mathbf{B}'_1$.



Slika A.6: Rezultat izvirnega algoritma, ki izbira pare podobnih elementov z uporabo algoritma minimin na drevesu stanj do globine 1, ter na koncu dodatno uravnoteži še množici $\mathbf{A}'_1$ in $\mathbf{B}'_1$.



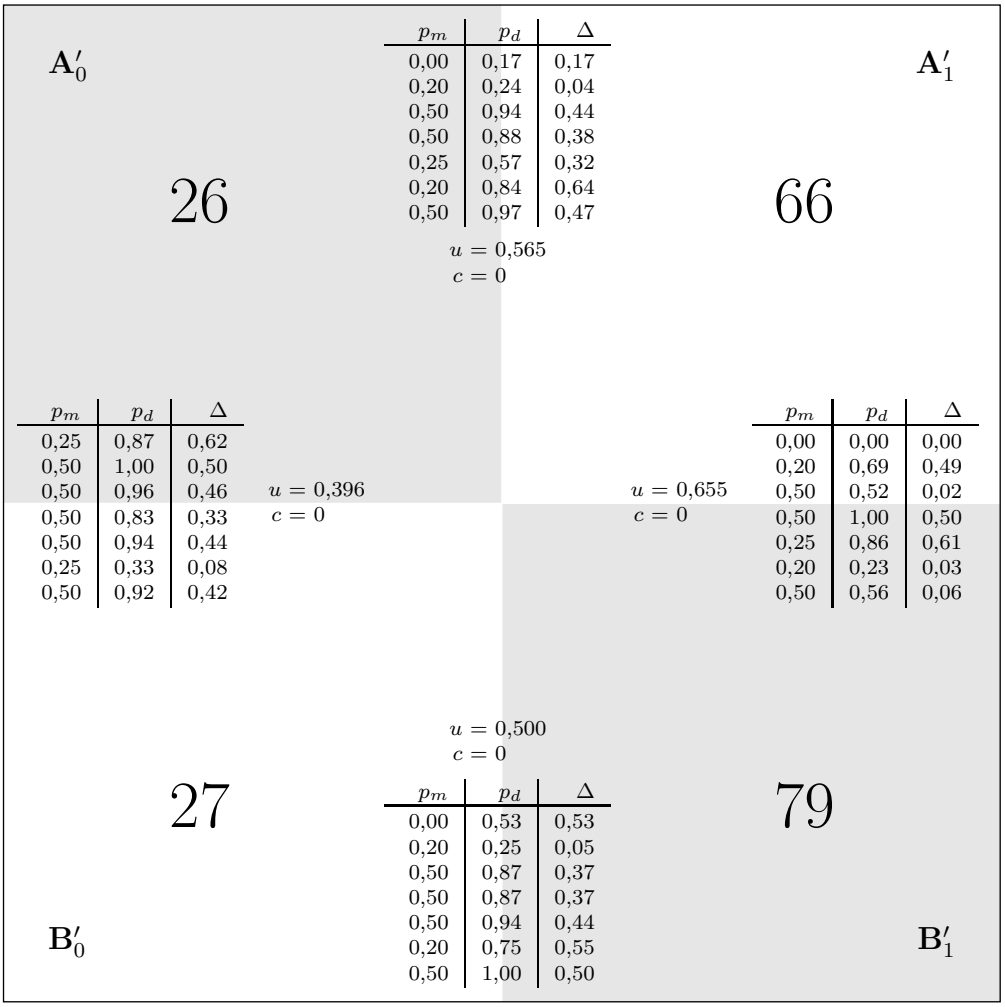
p_m	p_d	Δ
0,00	0,03	0,03
0,20	0,29	0,09
0,50	0,82	0,32
0,50	0,88	0,38
0,25	0,78	0,53
0,20	0,97	0,77
0,50	0,99	0,49

p_m	p_d	Δ
0,25	0,61	0,36
0,50	0,75	0,25
0,50	1,00	0,50
0,50	1,00	0,50
0,50	0,77	0,27
0,25	0,58	0,33
0,50	1,00	0,50

p_m	p_d	Δ
0,00	0,01	0,01
0,20	0,47	0,27
0,50	0,56	0,06
0,50	0,54	0,04
0,25	0,60	0,35
0,20	0,73	0,53
0,50	0,98	0,48

p_m	p_d	Δ
0,00	0,48	0,48
0,20	0,30	0,10
0,50	0,97	0,47
0,50	0,57	0,07
0,50	0,77	0,27
0,20	0,62	0,42
0,50	0,99	0,49

Slika A.7: Rezultat izvirnega algoritma, ki izbira pare podobnih elementov z uporabo algoritma minimin na drevesu stanj do globine 2, ter na koncu dodatno uravnoteži še množici $\mathbf{A}'_1$ in $\mathbf{B}'_1$.



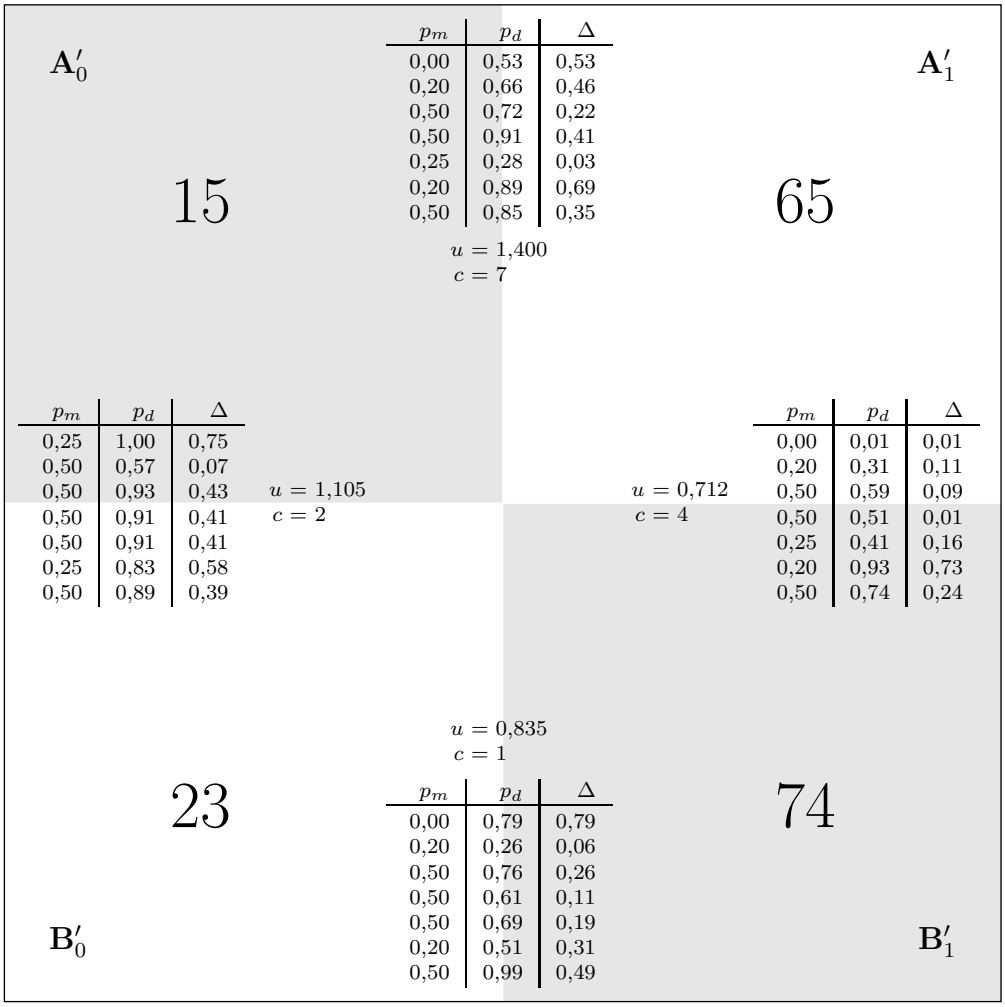
p_m	p_d	Δ
0,00	0,17	0,17
0,20	0,24	0,04
0,50	0,94	0,44
0,50	0,88	0,38
0,25	0,57	0,32
0,20	0,84	0,64
0,50	0,97	0,47

p_m	p_d	Δ
0,25	0,87	0,62
0,50	1,00	0,50
0,50	0,96	0,46
0,50	0,83	0,33
0,50	0,94	0,44
0,25	0,33	0,08
0,50	0,92	0,42

p_m	p_d	Δ
0,00	0,00	0,00
0,20	0,69	0,49
0,50	0,52	0,02
0,50	1,00	0,50
0,25	0,86	0,61
0,20	0,23	0,03
0,50	0,56	0,06

p_m	p_d	Δ
0,00	0,53	0,53
0,20	0,25	0,05
0,50	0,87	0,37
0,50	0,87	0,37
0,50	0,94	0,44
0,20	0,75	0,55
0,50	1,00	0,50

Slika A.8: Rezultat izvirnega algoritma, ki izbira pare podobnih elementov s požrešno strategijo, ter na koncu množice uravnoteži z izločanjem elementov na vseh množicah hkrati.



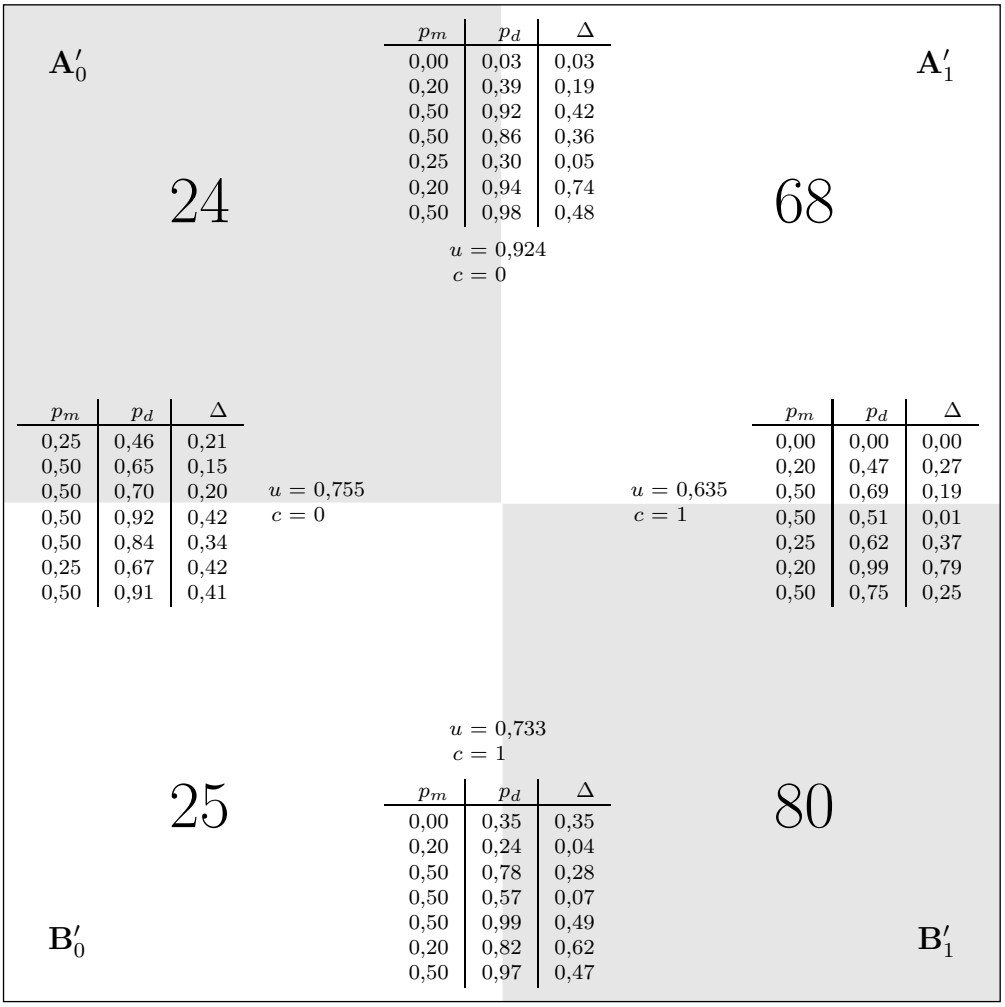
p_m	p_d	Δ
0,00	0,53	0,53
0,20	0,66	0,46
0,50	0,72	0,22
0,50	0,91	0,41
0,25	0,28	0,03
0,20	0,89	0,69
0,50	0,85	0,35

p_m	p_d	Δ
0,25	1,00	0,75
0,50	0,57	0,07
0,50	0,93	0,43
0,50	0,91	0,41
0,50	0,91	0,41
0,25	0,83	0,58
0,50	0,89	0,39

p_m	p_d	Δ
0,00	0,01	0,01
0,20	0,31	0,11
0,50	0,59	0,09
0,50	0,51	0,01
0,25	0,41	0,16
0,20	0,93	0,73
0,50	0,74	0,24

p_m	p_d	Δ
0,00	0,79	0,79
0,20	0,26	0,06
0,50	0,76	0,26
0,50	0,61	0,11
0,50	0,69	0,19
0,20	0,51	0,31
0,50	0,99	0,49

Slika A.9: Rezultat izvirnega algoritma, ki izbira pare podobnih elementov z uporabo algoritma minimin na drevesu stanj do globine 1, ter na koncu množice uravnoteži z izločanjem elementov na vseh množicah hkrati.



Slika A.10: Rezultat izvirnega algoritma, ki izbira pare podobnih elementov z uporabo algoritma minimin na drevesu stanj do globine 2, ter na koncu množice uravnoteži z izločanjem elementov na vseh množicah hkrati.