UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Rok Tomc

Optimizacija strukture svetlobnih prikazovalnikov z uporabo preiskovalnih algoritmov

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Zoran Bosnić

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01637/2010
Datum: 15.02.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ROK TOMC**

Naslov: **OPTIMIZACIJA STRUKTURE SVETLOBNIH PRIKAZOVALNIKOV Z UPORABO PREISKOVALNIH ALGORITMOV**

OPTIMIZATION OF DISPLAY STRUCTURE USING SEARCH ALGORITHMS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Prikazovalniki so strukturirani iz matrike svetilnih elementov (žarnic) in so sposobni prikazovati poljubne napise iz črk in številk. Ker so za izpisovanje poljubnih sporočil svetilni elementi neenakomerno izkoriščeni, je možno stroške proizvodnje prikazovalnikov zmanjšati s pametno izbiro pozicij svetilnih elementov na prikazovalniku dovolj visoke ločljivosti.

Kandidat naj v diplomski nalogi obravnava različne pristope optimizacije opisanega problema. V svojem delu naj predlaga smiselno oceno kakovosti prikaza simbolov na prikazovalniku in naj implementira ter oceni uspešnosti različnih preiskovalnih algoritmov za reševanje opisanega problema.

Mentor:

doc. dr. Zoran Bosnić

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:

prof. dr. Franc Solina

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani Rok Tomc,

z vpisno številko 63000297,

sem avtor diplomskega dela z naslovom:

Optimizacija strukture svetlobnih prikazovalnikov z uporabo preiskovalnih algoritmov.

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Zorana Bosnića
- 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 v zbirki »Dela FRI«.

V Ljubljani, dne 12.05.2010 Podpis avtorja: _____

Zahvala

Zahvaljujem se mentorju, doc. dr. Zoranu Bosniću, za usmerjanje pri izdelavi diplomskega dela ter za nepričakovano požrtvovalnost in pripravljenost na delo ob koncu tedna.

Zahvaljujem se tudi podjetju Evolve d.o.o. in direktorju Luki Brzinu za prevzem finančnih stroškov izdelave diplomske naloge.

Nenazadnje se zahvaljujem tudi svojima staršema, ki sta mi omogočila lagodno življenje v času študija in me ves čas podpirala.

Kazalo

Povzetek

Sodobni svetlobni prikazovalniki običajno sestojijo iz matrike žarnic in so sposobni prikazovati poljubne napise iz črk in številk. Različne napise oblikujejo z izmeničnim prižiganjem in ugašanjem ustreznih žarnic. Ker je pri večini napisov vsaj nekaj žarnic neuporabljenih za prikaz, bi lahko zmanjšali ceno izdelave takih prikazovalnikov tako, da bi zmanjšali število žarnic, ki prikazovalnik sestavljajo.
Pri taki optimizaciji se srečamo z dvema nasprotujočima si ciljema: a) uporabiti čim manjše število žarnic in b) poskrbeti, da bo napis berljiv.
V diplomski nalogi smo implementirali različne pristope preiskovanja rešitev, ki so v prostoru razporeditev žarnic po svetlobnem prikazovalniku iskale najbolj optimalno razporeditev. Optimalnost rešitve smo ocenili s hevristično oceno, ki smo jo razvili in predlagali za namene obravnavanega problema. Predlagana ocena za razporeditev žarnic na svetlobnem prikazovalniku določene velikosti ocenjuje, kako kvalitetno je tak prikazovalnik sposoben prikazati izbrano množico simbolov. Pri tem ocena bolje ocenjuje take razporeditve, ki za enako kvaliteten prikaz potrebujejo manj žarnic na prikazovalniku.
Izmed vseh algoritmov se je za najbolj uspešnega izkazal algoritem iskanja harmonije. Ta je najbolj uspešno preiskoval prostor in našel najbolj kvalitetne razporeditve, pri samem iskanju pa se je pri večkratnih ponovitvah obnašal najbolj konsistentno.

Ključne besede:

optimizacija, prikazovalniki, preiskovalni algoritmi, minimizacija cene

Abstract

Display signs usually consist of lightbulbs organized into a matrix and are capable of dynamically displaying messages. Since not every message uses every available bulb, it is possible to optimize the production costs of such signs by reducing the number of bulbs on the sign.
As with all optimization problems, we are confronted with the two opposing goals: a) we would like to use as few lightbulbs as possible and b) we require the displayed messages to be readable.
In our thesis, we implemented several optimization algorithms for searching through state space of lightbulbs' configurations, trying to find the most optimal position of the lightbulbs. We estimate the optimality of the solution using a heuristical estimate which we developed and proposed especially for our approach. The proposed estimate estimates a quality with which a sign with certain configuration of lightbulbs is capable to represent a set of selected symbols. The estimate takes into consideration number of used bulbs and favors such configurations with less bulbs.
Among the implemented algorithms, the harmony search has shown to be the most successful in this task. Its solutions had best quality values and on consecutive searches showed the most consistent behaviour.

Key words:

optimization, display, search algorithms, cost minimization

1 Uvod

Svetlobne prikazovalnike vsakodnevno srečujemo ob cestah, predvsem pa je njihova uporaba razširjena na avtocestah (slika 1.1). Prometni prikazovalniki nam npr. posredujejo obilico informacij: priporočeno hitrost vožnje po cesti, obvestila o morebitnih prometnih zastojih, temperaturo cestišča in zraka, hitrost vetra... Nekateri svetlobni prikazovalniki celo dinamično prikazujejo omejitev najvišje hitrosti vožnje, ki se spreminja glede na razmere na cesti. Njihova uporaba je prav gotovo vsestranska in, čeprav se med seboj razlikujejo po velikosti in obliki, imajo nekaj vsi skupnega. Sestavljeni so iz matrike žarnic.

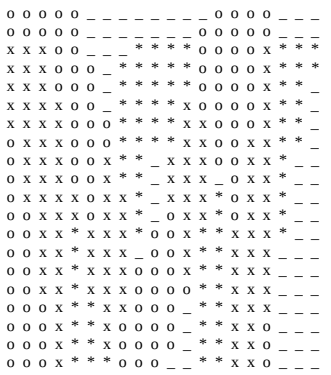
Slika 1.1: *Primer svetlobnega prometnega prikazovalnika*

Svetlobni prikazovalniki prikazujejo želena sporočila tako, da z izmeničnim prižiganjem in ugašanjem žarnic v svoji matriki oblikujejo posamezne simbole (črke in številke), ki skupaj tvorijo sporočilo. Izdelava takih prikazovalnikov zahteva veliko število žarnic, te pa so običajno precej drage. Zato bi radi njihovo število zmanjšali in s tem pocenili izdelavo prikazovalnikov. Pri tem pa ne bi radi okrnili berljivosti napisov, ki jih prikazovalniki oblikujejo. Zastavili smo si torej dva nasprotujoča si cilja, poiskati pa moramo način, kako bi oba čim bolje izpolnili.

Za potrebe diplomske naloge smo se omejili na kvadratne prikazovalnike, velikosti N x N žarnic, ki znajo prikazati katerokoli izmed števil 0 – 9, vseh 25 črk slovenske abecede in črke Q, W, X in Y. Omejili smo se na same velike črke. Tako je v naboru simbolov 39 črk in številk. Omejimo se tudi na istočasni prikaz le enega simbola iz nabora. Priloga A vsebuje vizualno ponazoritev 39 prikazovalnikov velikosti 20 x 20 žarnic – vsak prikazuje po en simbol.

Če želimo število žarnic na prikazovalniku zmanjšati, moramo najprej ugotoviti, katere žarnice so pri prikazu izbranih simbolov manj pomembne. V primeru, da bi ugotovili, da se določena žarnica ne prižge pri nobenem prikazu simbolov, bi jo lahko mirne vesti odstranili s prikazovalnika in s tem zmanjšali stroške izdelave.

Na primer, pri črki I je že na prvi pogled jasno, da je za njen prikaz odveč 4/5 žarnic Pravzaprav pri velikosti prikazovalnika 20 x 20 žarnic potrebuje za popolno predstavitev natanko 80 žarnic od 400. Vendar pa se že pri pri črkah W in M pokaže, da tako razmišljanje ne bo obrodilo sadov. Črki potrebujeta največ žarnic (M 244 in W 205), pri tem pa unija žarnic, potrebnih za njun prikaz, zavzema že skoraj celoten prikazovalnik (slika 1.2).



Slika 1.2: *Unija žarnic, pri prikazu črk M in W. 'o' označuje žarnice, potrebne za prikaz črke M, '*' označuje žarnice, potrebne za prikaz črke W in 'x' žarnice, potrebne za prikaz katerekoli izmed obeh črk.*

Dejansko v primeru, ko prikazovalnik velikosti 20 x 20 žarnic vse simbole, prikazane v prilogi A, prikazuje poravnane s svojim levim robom, ne zmore prikazati vseh simbolov popolno, če ima na prikazovalniku manj kot 359 od 400 žarnic. Ceno proizvodnje bi se torej že na tak način dalo zmanjšati za slabih 10%. Preveriti pa želimo, ali lahko ceno še dodatno znižamo, brez da bi bistveno zmanjšali prepoznavnost prikazanih simbolov.

Pripravljeni smo se zadovoljiti s tem, da vsi simboli ne bodo imeli prisotnih vseh žarnic, ki jih potrebujejo za svoj popoln prikaz. S tem v mislih je potrebno pripraviti hevristično oceno, ki bo znala oceniti, kako kvalitetna je predlagana razporeditev žarnic na prikazovalniku.

Ocena mora zadostovati trem kriterijem:

a) **boljša kakovost posameznega prikaza**: ocena mora podati boljšo oceno takim razporeditvam žarnic, ki vizualno bolj kvalitetno prikazujejo simbole.
b) **enakomerna kakovost vseh prikazov**: ocena mora podati boljšo oceno taki razporeditvi žarnic, ki prikaže vse simbole čimbolj enako kvalitetno.
c) **minimizacija števila žarnic:** ocena mora za razporeditvi žarnic, ki enako kvalitetno prikazujeta simbole, podati boljšo oceno tisti razporeditvi, ki vsebuje manj žarnic.

Da bi našli razporeditev žarnic, ki najbolj ustreza zastavljenim ciljem, se poslužimo preiskovalnih algoritmov kombinatorične optimizacije. Ti za preiskovanje prostora hipotez potrebujejo mero za ocenjevanje kakovosti hipotez. Zato najprej v drugem poglavju razvijemo predlagano hevristično oceno. V tretjem poglavju opišemo osem postopkov optimizacije in vsakega od njih implementiramo v dani problemski domeni. V četrtem poglavju so predstavljeni rezultati empiričnega testiranja algoritmov in njihova medsebojna primerjava, ki se zaključi z zaključnimi ugotovitvami in idejami za nadaljevanje v petem poglavju.

2 Implementacija hevristične ocene

2.1 Funkcija kvalitete predstavitve simbola

Prva naloga ocene je ta, da oceni, kako kvalitetno lahko določena razporeditev žarnic prikaže vsakega od izbranih simbolov. Zato razvijemo funkcijo, ki za vsak simbol iz nabora ugotovi, katere žarnice iz razporeditve morajo biti prižgane za njegovo predstavitev. Hkrati tudi ovrednoti, kako kvalitetna je taka predstavitev simbola.

Intuitivno določimo, da mora funkcija za določen simbol najbolje ovrednotiti take razporeditve žarnic, pri katerih se da prižgati vse žarnice, ki so medsebojno razporejene natanko tako, kot točke simbola. Tedaj prikazovalnik lahko prikaže simbol v polni kvaliteti. Sicer mora vrednost funkcije padati čimbolj sorazmerno z zmožnostjo prižgati žarnice tako, da bo prikaz podoben ciljnemu simbolu. Če se vrednost funkcije ne spremeni, kadar je določena žarnica prižgana ali ugasnjena, smatramo, da na predstavitev simbola nima vpliva. Taka žarnica ostane ob prikazu simbola ugasnjena.

Večina simbolov je po širini manjših, kot pa je velik prikazovalnik. Tako je možno, da se pri kombinaciji večih simbolov lahko žarnice enega simbola bolje pokrijejo z žarnicami drugega simbola, če se ne prikažeta poravnana z levim robom prikazovalnika. Zato funkcija, ob ovrednotenju prikaza posameznega simbola, poskusi premakniti simbol za vse možne kombinacije vzdolž prikazovalnika in izvede ocenjevanje za vsako tako translacijo. Vrnjena vrednost je tako najboljša vrednost izmed vseh teh translacij.

Preprosta različica funkcije bi lahko merila samo delež točk simbola, ki so prekrite z žarnicami. Zamislimo si primer, v katerem bi želeli prikazati črko O s črto debeline dveh točk. Preprosta funkcija bi enako ovrednotila razporeditev žarnic, ki bi zmogla prikazati samo levo polovico črke O in razporeditev, ki bi lahko prikazala samo zunanji rob črke O. Obe razporeditvi žarnic bi pokrili natanko polovico točk črke. Kljub enaki vrednosti funkcije, bi prva razporeditev pravzaprav prikazovala črko C, druga razporeditev žarnic pa tanjšo, a vseeno razpoznavno črko O. Taka preprosta funkcija ne bi zadovoljila naše zahteve, da bolje vrednotimo take razporeditve, ki lahko bolj prepoznavno prikažejo izbrani simbol. Zato v nadaljevanju poskusimo poiskati bolj napredno funkcijo.

2.1.1 Osvetljevanje okolice žarnice

Začetna metoda temelji na predpostavki, da žarnica osvetljuje svojo okolico in, da je človeško oko sposobno svetlobo žarnic povezati v črto oz. linijo, če so si te dovolj blizu. Ob žarnicah, prižganih na ključnih mestih, bi moralo človeško oko svetlobo sámo povezati v simbol.

Matriki osvetljenosti, ki je enake velikosti, kot je matrika prikazovalnika, funkcija za vsako žarnico iz razporeditve prišteje vrednosti iz matrike svetilnosti žarnice, prikazane na sliki 2.1. Pri tem matriko svetilnosti poravna tako, da je žarnica na mestu, kjer ima matrika vrednost 5. Kadar dve žarnici osvetlita isto točko, se svetilnosti posameznih žarnic seštejeta. Algoritem za vsako točko simbola primerja, ali osvetljenost na istem mestu v matriki osvetljenosti presega izbrano spodnjo mejo osvetljenosti. V kolikor jo, se funkcijski rezultat poveča za eno enoto. Namen spodnje meje osvetljenosti je določiti, kdaj smatramo, da je neka točka dovolj

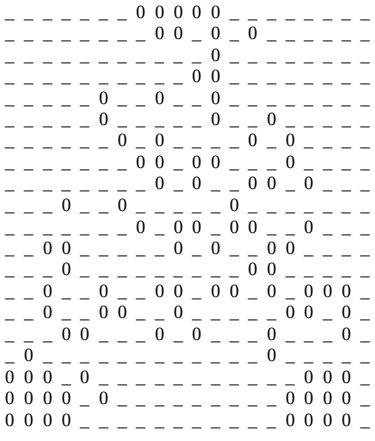
osvetljena, da bi jo moralo oko zaznati kot osvetljeno, čeprav na njenem mestu ni prižgana nobena žarnica.

0	0	0	0	1	0	0	0	0
0	0	0	1	2	1	0	0	0
0	0	1	2	3	2	1	0	0
0	1	2	3	4	3	2	1	0
1	2	3	4	5	4	3	2	1
0	1	2	3	4	3	2	1	0
0	0	1	2	3	2	1	0	0
0	0	0	1	2	1	0	0	0
0	0	0	0	1	0	0	0	0

Slika 2.1: *Matrika svetilnosti žarnice.*

Z uporabo matrike svetilnosti s slike 2.1 je metoda morda učinkovita za prikazovalnike velikosti reda 100 x 100 žarnic, za manjše prikazovalnike, na primer velikosti 20 x 20 žarnic, pa je funkcija preveč radodarna s pripisovanjem vrednosti osvetljenosti. Že ena žarnica lahko osvetli tretjino prikazovalnika in vrednosti v matriki osvetljenosti kaj hitro presežejo izbrano mejo osvetljenosti. Metodo bi zato bilo potrebno spremeniti tako, da bi se matrika svetilnosti žarnice prilagajala velikosti prikazovalnika – njene vrednosti bi se za manjše prikazovalnike zmanjšale, zmanjševati pa bi bilo potrebno tudi velikost okolice, ki jo ena žarnica osvetli.

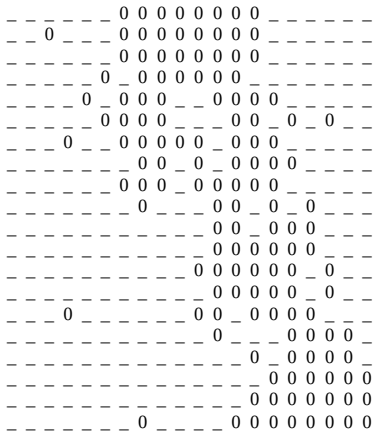
Pokaže pa se tudi druga težava metode. Ko je v okolici določene točke dovolj žarnic, da osvetljenost točke preseže spodnjo mejo osvetljenosti, funkcija enakovredno ovrednoti razporeditve, ne glede na to, kjer se nahajajo preostale žarnice (slika 2.2). Te so lahko v bližini simbola ali pa povsem na robu prikazovalnika, funkcije njihovo mesto ne zanima več. Želimo pa, da bolje ovrednoti take razporeditve, kjer se čim več žarnic nahaja na istih mestih, kot točke na simbolu, saj je tako simbol bolje predstavljen.



2	2	5	7	12	17	25	30	35	37	39	36	32	26	19	13	8	6	5	4
2	3	4	6	10	17	25	32	38	42	42	40	39	32	24	16	11	7	6	6
4	4	6	7	12	18	28	37	43	45	47	47	44	38	28	20	14	8	6	6
6	6	6	8	13	19	28	37	40	43	47	48	47	41	31	22	16	10	7	4
8	8	10	11	18	24	32	36	38	41	44	45	45	42	34	24	16	10	7	5
6	7	9	13	21	28	33	36	39	40	40	40	43	43	37	27	18	9	7	5
4	5	10	14	24	31	35	37	36	35	35	35	40	43	38	30	20	12	9	6
3	7	13	19	28	37	42	41	37	32	34	34	38	41	38	32	25	17	13	7
3	6	13	20	30	37	41	40	35	27	26	30	34	34	32	32	27	20	14	7
3	7	15	24	36	42	43	43	38	30	27	29	31	32	32	32	29	22	18	11
3	7	16	26	38	43	43	40	35	33	32	30	30	34	37	36	31	28	27	18
3	7	15	25	37	41	39	35	34	34	34	32	33	36	36	36	34	31	29	20
5	11	21	30	39	41	42	37	36	37	38	37	37	34	33	35	34	31	26	18
5	13	23	31	35	38	38	33	31	32	38	40	39	36	35	36	36	33	28	19
9	18	28	35	39	37	35	28	27	28	33	36	37	34	33	37	38	36	31	21
14	23	32	41	43	38	31	23	19	20	25	28	30	30	33	38	43	44	41	31
22	31	39	48	48	40	33	21	16	14	17	19	20	24	30	38	45	50	48	38
25	37	45	50	49	40	30	19	12	10	9	9	11	16	24	35	46	52	50	39
27	36	44	47	43	35	28	16	9	5	5	4	5	10	18	30	42	48	45	37
23	31	34	37	32	25	20	11	4	2	3	3	4	8	14	24	34	39	38	31

Slika 2.2: *Ena izmed najbolje ovrednotenih razporeditev 140 žarnic na prikazovalniku velikosti 20 x 20 za predstavitev črke A pri mejni osvetljenosti 30 (levo) in matrika osvetljenosti te razporeditve (desno).*

Na sliki 2.2 je razvidno, da osvetljenost vseh točk črke A presega spodnjo mejo 30. Vendar je očitno, da žarnice niso razporejene zgolj na mestih točk črke A, pač pa tudi drugod po prikazovalniku, pri tem pa mesta črke A ostajajo prazna.



```
2  3  5  7  14 22 34 44 51 53 53 51 44 33 20 10 4  1  _  _
3  4  6  11 19 31 45 57 64 66 66 63 56 43 26 14 5  2  _  _
2  3  7  12 21 35 52 64 70 70 69 66 62 49 31 16 7  3  1  _
1  4  8  14 23 37 55 67 70 67 65 63 61 52 35 20 9  5  2  1
1  4  10 16 25 38 54 65 67 61 61 59 59 53 41 27 15 9  5  2
1  4  9  16 25 38 53 62 61 57 55 57 59 55 46 32 21 12 6  4
2  3  7  13 23 36 51 59 58 52 54 58 63 58 49 37 24 13 6  2
1  2  3  9  17 29 41 51 51 47 50 59 64 60 53 41 28 15 6  2
_  1  3  6  13 21 32 41 44 43 49 56 63 62 57 44 30 17 6  1
_  _  1  4  6  12 20 27 31 34 42 52 61 64 60 48 32 21 9  3
_  _  _  2  2  5  11 16 20 25 38 52 63 67 65 53 39 25 14 6
_  _  1  2  1  2  5  10 13 21 38 55 65 69 68 60 46 31 17 7
_  1  2  3  2  1  3  6  11 20 38 56 64 68 69 64 52 36 20 8
1  2  3  4  3  2  1  4  9  19 34 51 59 63 66 64 56 41 25 12
2  3  4  5  4  3  3  4  8  16 28 42 50 57 62 63 57 47 33 19
1  2  3  4  3  2  1  3  5  10 19 29 38 50 58 63 61 54 43 28
_  1  2  3  2  1  1  2  3  6  12 21 31 44 56 65 66 63 51 35
_  _  1  2  1  1  2  3  2  3  9  18 27 41 54 66 70 67 56 40
_  _  _  1  1  2  3  4  3  4  9  17 26 39 51 62 67 64 54 39
_  _  _  1  2  3  4  5  5  6  9  15 23 34 43 51 54 51 44 32
```

Slika 2.3: *Ena izmed najbolje ovrednotenih razporeditev 140 žarnic na prikazovalniku velikosti 20 x 20 za predstavitev črke A pri mejni osvetljenosti 50 (levo) in matrika osvetljenosti te razporeditve (desno).*

Slika 2.3 nazorno prikazuje, kaj se zgodi, če poskusimo povečati spodnjo mejo osvetljenosti na 50 v želji, da bi funkcija bolje ovrednotila razporeditve, ki imajo čim več žarnic razporejenih na mesta točk črke A.
Funkcija najbolje ovrednoti razporeditev žarnic, kjer so žarnice razporejene zgolj ob en del črke A, drug del črke pa je skoraj popolnoma zanemarjen.

Sklepamo, da je slaba lastnost tako zastavljene funkcije v mejni osvetljenosti. Določena vrednost mejne osvetljenosti povzroči povsem drugačno vrednotenje razporeditev žarnic ob prikazovalnikih različnih velikosti in za simbole z različnim številom točk. Poleg tega, da bi bilo potrebno mejo natančno določiti za vsako velikost prikazovalnika in za vsak simbol, bi za različne velikosti prikazovalnika bilo potrebno spreminjati tudi matriko svetilnosti žarnice. Oboje bi se zagotovo izkazalo za dodaten neodvisen problem.

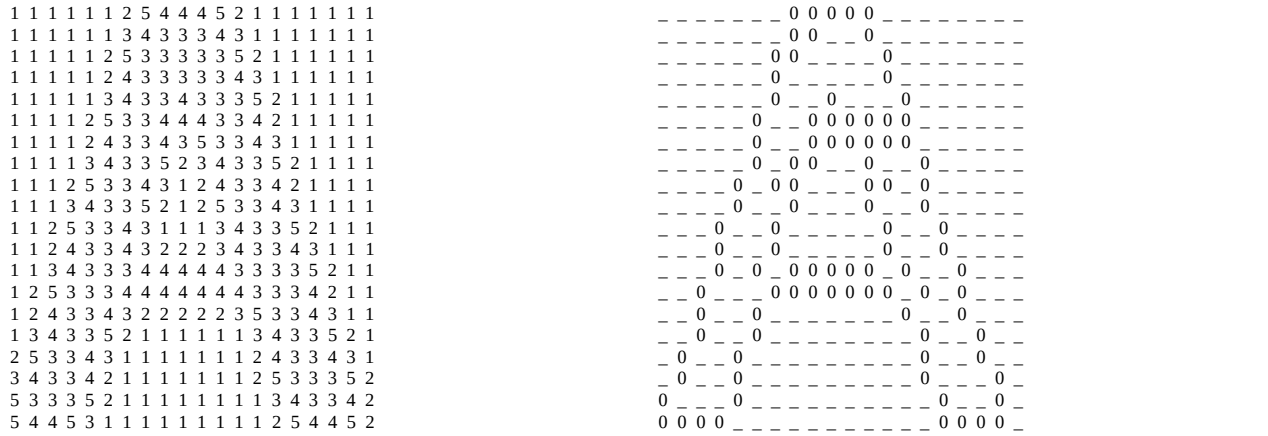
2.1.2 Oglišča in robovi simbola

Zaradi ugotovitev iz prejšnjega podpoglavja se lotimo drugačnega pristopa k ideji razvoja ocenjevalne funkcije osvetlitve. Človeški možgani imajo sposobnost nadomeščanja informacij, ki jih nimajo. Raziskovalci so odkrili, da lahko zaradi slepe pege očesa možgane pretentamo tako, da vidimo prekinjeno črto kot neprekinjeno [11]. Pa tudi sicer možgani radi razberejo znane vzorce, včasih tudi tam, kjer vzorcev ni [12]. Najpreprostejši vzorec, ki ga lahko razpoznamo, je ravna črta. Ob pomanjkanju žarnic zato želimo, da se čim več le teh nahaja na robu simbola in bi jih oko oz. možgani lahko povezali v linijo, več linij pa skupaj v simbol. Žarnice v notranjosti simbola bi morale biti razporejene čimbolj enakomerno.

V skladu s temi ugotovitvami definirajmo funkcijo tako, da v vsaki točki na prikazovalniku določi kvaliteto na sledeči način:

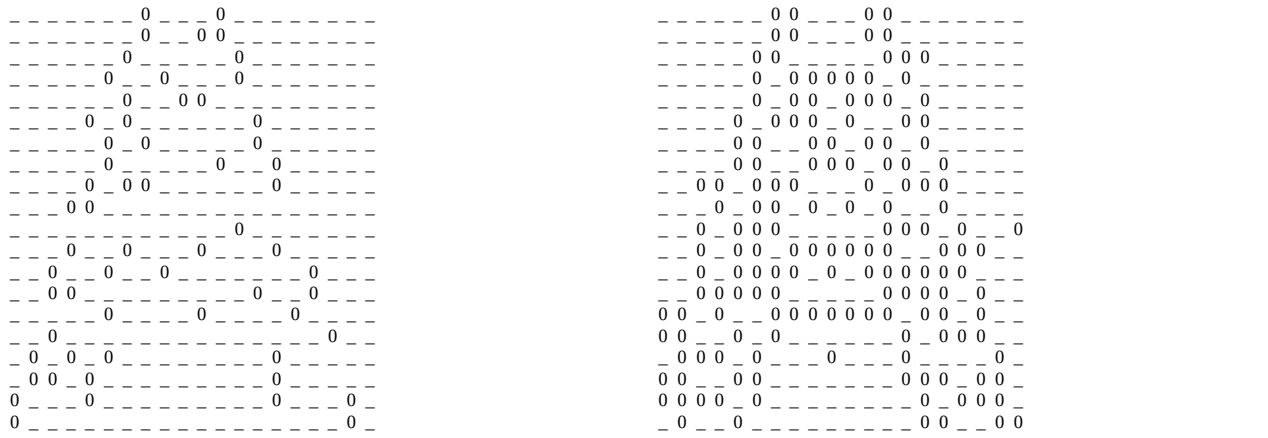
- če se točka ne nahaja v bližini simbola, je njena kvaliteta 1,
- če se točka nahaja ob zunanjem robu simbola (t.j. ne na simbolu, pač pa tik ob njem), je njena kvaliteta 2,
- če se točka nahaja v zunanjem oglišču simbola (t.j. ne na simbolu, pač pa tik ob njem v oglišču), je njena kvaliteta 3

- če se točka nahaja na simbolu, vendar ne na robu ali v oglišču, je njena kvaliteta prav tako 3
- če se točka nahaja na notranjem robu simbola, je njena kvaliteta 4 in
- če se točka nahaja v notranjem oglišču simbola, je njena kvaliteta 5.



Slika 2.4: *Matrika kvalitet točk črke A na prikazovalniku velikosti 20 x 20 žarnic (levo) in najbolje ovrednotena razporeditev 100 žarnic na prikazovalniku velikosti 20 x 20 žarnic za predstavitev črke A.*

Funkcija vsaki uporabljeni žarnici pripiše kvaliteto, ki jo ima točka na prikazovalniku simbola ter kvalitete žarnic sešteje. Iz slike 2.4 je razvidno, da tako zastavljena funkcija ovrednoti kot najboljše take razporeditve žarnic, ki imajo žarnice razporejene po robovih simbola.



Slika 2.5: *Najbolje ovrednotena razporeditev 60 (levo) in 160 (desno) žarnic za predstavitev črke A na prikazovalniku velikosti 20 x 20 žarnic, kadar funkcija vrednost kvalitete žarnice uteži z razdaljo do žarnice enake kvalitete.*

Še vedno pa želimo, da bi bile bolje ovrednotene razporeditve, kjer sta dve žarnici razporejeni na robu simbola, ena pa v notranjosti, kot pa če vse tri ležijo na istem robu. Zato funkcijo poskusimo izboljšati z utežitvijo kvalitete točke z razdaljo do naslednje točke enake kvalitete. Hkrati želimo doseči tudi to, da bi se točke razporejale čimbolj enakomerno po točkah simbola. Kadar točke A, B in C ležijo vse na isti premici, od katerih točka B leži med točkama A in C, se vsota razdalj AB in BC ne spreminja, ne glede na to, kje točka B leži. Če pa seštevamo korena razdalj, je seštevek največji takrat, ko točka B leži natanko na sredini med točkama A in C. Zato funkcija kvaliteto žarnice uteži še s korenom razdalje do najbližje

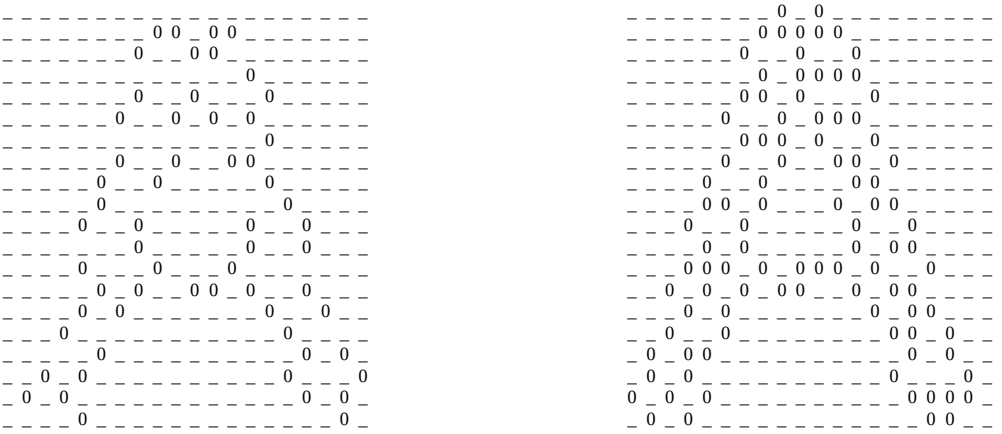
žarnice enake kvalitete. V podobnih primerih se običajno sicer res uporablja mera kvadrata razdalj, vendar, ker želimo vrednosti maksimizirati in ne minimizirati, je korenjenje bolj neposredna tehnika. Tako utežene kvalitete žarnic funkcija sešteje.

Iz slike 2.5 je razvidno, da tako zasnovana funkcija nabolje ovrednoti razporeditve, pri katerih je prikazani simbol dejansko razpoznaven. Pri tem se funkcija želeno odziva na število žarnic – če jih je manj, so bolje ovrednotene kadar so razporejene bolj ob robovih simbola, če jih je več, bolj nagrajuje polnjenje notranjosti simbola.

Črka A v izbranem naboru simbolov (priloga A) je sestavljena iz 146 točk. Pričakovali bi, da se pri najbolje ovrednoteni razporeditvi 160 žarnic skoraj ne bi smelo opaziti, da kakšna točka črke manjka. Kaj šele, da se moramo potruditi razbrati, kateri simbol je prikazan.

2.1.3 Združitev v skupno funkcijo

V končnem koraku združimo ideji, predstavljeni v razdelkih 2.1.1 in 2.1.2 v skupno funkcijo. Vsaka točka na simbolu dobi pripisano kvaliteto od 2 do 4 – če je točka zgolj na simbolu, dobi pripisano kvaliteto 2, če je na robu simbola, 3, če je v oglišču simbola, 4. Vsaka žarnica na prikazovalniku nato 'osvetli' svojo okolico s svojo kvaliteto. Matrika osvetljenosti je enake velikosti kot prikazovalnik. Vsaka žarnica v matriko osvetljenosti prišteje v svoji okolici velikosti 1 svojo pripisano kvaliteto. Funkcija vse vrednosti v matriki koreni in sešteje.



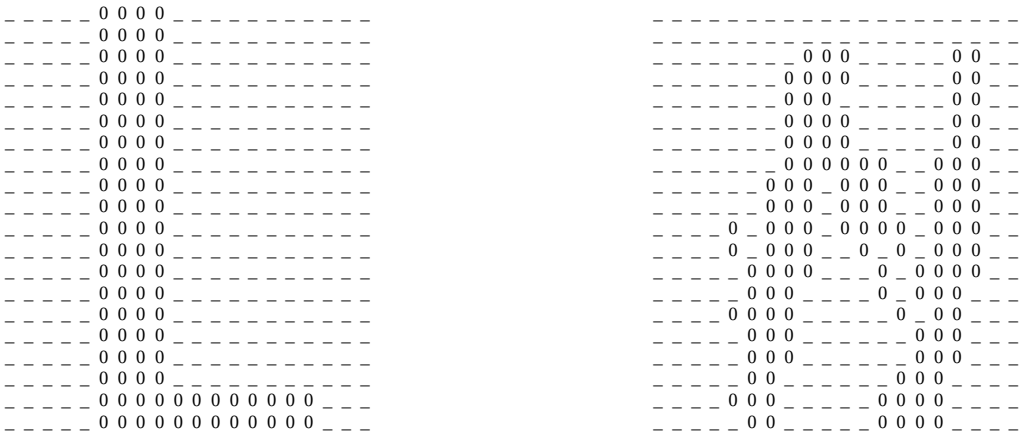
Slika 2.6: *Najbolje ovrednotena razporeditev 60 (levo) in 100 (desno) žarnic na prikazovalniku velikosti 20 x 20 žarnic za prikaz črke A s končno implementacijo funkcije.*

Iz slike 2.6 je razvidno, da končna funkcija najbolje ovrednoti razporeditev žarnic, pri kateri se da črko A vizualno razpoznati že pri zelo majhnem številu uporabljenih žarnic. Pri večih uporabljenih žarnicah funkcija najbolje ovrednoti tako razporeditev, kjer so žarnice na simbolu najbolj enakomerno razporejene.

S tem smo dobili mero, ki zadostno izpolnjuje prvo zahtevo, da mora hevristična ocena podati boljšo oceno takim razporeditvam žarnic, ki vizualno bolj kvalitetno prikazujejo simbole.

2.2 Združevanje vrednosti različnih predstavljenih simbolov

V prejšnjem razdelku smo razvili funkcijo, ki zna ovrednotiti kvaliteto predstavitve posameznega simbola z izbrano razporeditvijo žarnic na prikazovalniku. V nadaljevanju želimo vrednosti funkcije za posamezne simbole združiti v skupno oceno tako, da bo nova nadgrajena ocena bolje ocenjevala tako razporeditev žarnic, ki prikaže *vse simbole* čimbolj enako kvalitetno. V nasprotnem primeru bi se lahko zgodilo, da bi ocena bolje ocenjevala razporeditve, pri katerih je večina žarnic razporejenih na točke, ki si jih deli veliko simbolov (veliko simbolov ima naprimer navpično črto). V takem primeru bi ena žarnica povečala vrednost funkcije kvalitete za večje število simbolov hkrati. Vendar bi zato zmanjkalo žarnic za tista specifična mesta, ki jih potrebuje samo kakšen poseben simbol. Na primer, črki M in W sta taki posebnosti, ki potrebujeta mnogo žarnic, razporejenih vsepovsod po prikazovalniku. V kolikor kvalitete predstavljenih simbolov ne silimo k izenačevanju, sta ti dve črki radi zapostavljeni in zato slabše prikazani, kar prikazuje slika 2.7.



Slika 2.7: *Prikaz črk L in W pri najbolje ocenjeni razporeditvi 250 žarnic na prikazovalniku velikosti 20 x 20 žarnic. Ocena vrednosti kvalitete posameznih simbolov sešteva.*

Če želimo primerjati kvalitete posameznih simbolov med seboj, je treba njihove ocene normalizirati glede na število točk, iz katerih je simbol sestavljen. Sicer dobi simbol, ki je sestavljen iz več točk, večjo oceno, kot drug simbol, ki je sestavljen iz manj točk, čeprav ima lahko manjši simbol pokrite vse točke z žarnicami, večji simbol pa ne. Vsako vrednost funkcije kvalitete normaliziramo z vrednostjo, ki jo funkcija pripiše popolni predstavitvi simbola. Vrednosti kvalitete predstavitve simbolov se tako vedno gibljejo med 0 in 1.

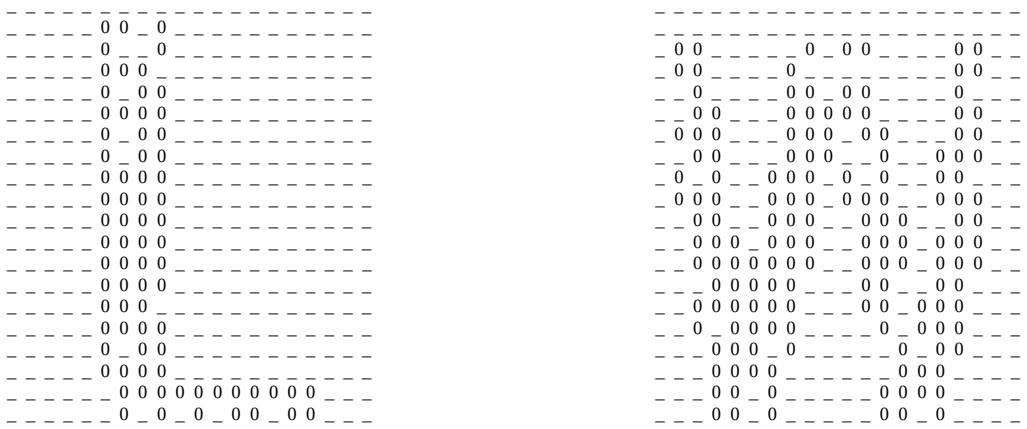
Veriga je toliko močna, kolikor je močan njen najšibkejši člen. Iz tega rekla lahko izluščimo, da, če želimo verigo izboljšati, moramo ojačati njen najšibkejši člen. Analogno, če želimo, da se izboljšuje skupina predstavljenih simbolov, za to poskrbimo tako, da izboljšamo predstavitev najslabše predstavljenega simbola.

Združeno oceno *o* izračunamo po enačbi

$$o = \sum_{x \in X} f(x) - |X| \left(\frac{\sum_{x \in X} f(x)}{|X|} - \min_{x \in X} f(x) \right), \qquad (1)$$

pri tem je *f(x)* normalizirana funkcija, razvita v prejšnjem poglavju, *x* posamezen simbol, *X* pa množica vseh simbolov, ki jih želimo prikazovati na prikazovalniku. Od seštevka

normaliziranih vrednosti kvalitete torej odštejemo razliko med povprečjem vrednosti in vrednostjo najslabše predstavljenega simbola. Ker razliko utežimo še s številom vseh predstavljivih simbolov, zagotovimo, da ocena ne more bolje oceniti razporeditve, ki bi se razlikovala samo v postavitvi ene žarnice in bi ta ena žarnica izboljšala prikaz vseh ostalih simbolov, razen najslabšega. Ocena lahko bolje oceni samo razporeditev, ki izboljša kvaliteto najslabše prikazanega simbola.



Slika 2.8: *Prikaz črk L in W pri najbolje ocenjeni razporeditvi 250 žarnic na prikazovalniku velikosti 20 x 20 žarnic. Ocena vrednosti kvalitet simbolov sešteje in zmanjša za razliko najslabše vrednosti kvalitete do povprečja kvalitet, pomnožene s številom simbolov.*

Na sliki 2.8 vidimo, da tako implementirana ocena ocenjuje razporeditve žarnic ustrezno z našimi željami. Tekom raziskovanja pa smo ugotovili, da se da enako doseči že tako, da za vrednost ocene uporabimo kar vrednost kvalitete najslabše predstavljenega simbola. Ker je taka ocena precej preprostejša od prejšnje, po principu Occhamovega rezila uporabimo preprostejši način in skupno oceno *o* namesto po enačbi (1), izračunamo po enačbi

$$o = \min_{x \in X} f(x). \qquad (2)$$

2.3 Minimizacija števila žarnic

V prejšnjih razdelkih smo implementirali oceno, ki zna efektivno razlikovati kvalitetne razporeditve žarnic od manj kvalitetnih. V tretjem delu jo želimo dopolniti tako, da bi bolje ocenjevala razporeditve, pri katerih je uporabljeno manjše število žarnic.

Postopamo tako, da vrednost ocene iz prejšnjega poglavja utežimo s številom uporabljenih žarnic. Vendar pa se pojavi težava pri izbiri ustreznih uteži: če je žarnica vredna preveč, ocena najbolje ovrednoti razporeditev z eno samo žarnico. V drugem primeru, ko je žarnica vredna premalo, je najbolje ocenjena razporeditev žarnic, kjer so uporabljene vse žarnice na prikazovalniku, ki jih simboli potrebujejo za svoj prikaz.

Odločimo se, da eksplicitno določimo želeno najmanjšo kvaliteto simbolov. Vse razporeditve žarnic, ki ne predstavijo vseh simbolov dovolj kvalitetno, označimo za slabe. Da pa bi algoritmi vseeno razlikovali med boljšimi in slabšimi slabimi razporeditvami, slabih razporeditev ne odrežemo s tem, da jim damo enako oceno. Boljša slaba razporeditev žarnic mora še vedno imeti boljšo oceno. Kvaliteto simbolov utežimo s številom mest na

prikazovalniku, če kvaliteta simbolov ne presega izbrane meje. Če pa jo, jo utežimo s številom uporabljenih žarnic. Končno oceno izračunamo po enačbi

$$ocena = \begin{cases} \dfrac{o}{\check{s}t.\ mest\ na\ prikazovalniku}, & o < meja \\ \dfrac{o}{\check{s}t.\ uporabljenih\ žarnic}, & o \geq meja \end{cases}, \tag{3}$$

kjer *o* izračunamo po enačbi (2).

Več uporabljenih žarnic se tako kaznuje samo tedaj, kadar je kvaliteta vseh predstavljenih simbolov zadostna. Do takrat se večanja števila uporabljenih žarnic ne kaznuje.

Z dobljeno končno hevristično oceno se lotimo drugega dela diplomske naloge: pregleda različnih algoritmov umetne inteligence in iskanja razporeditve žarnic na prikazovalniku z največjo oceno.

3 Preiskovalni algoritmi

S predlagano hevristično oceno želimo poiskati tako razporeditev žarnic, ki kar najbolj ustreza ciljem zastavljenega problema in za katero pričakujemo, da ji bo ocena pripisala najvišjo vrednost. Najbolj preprost način, kako bi lahko ustrezno razporeditev žarnic našli, je tako, da bi sestavili vse možne razporeditve žarnic in vsako ocenili. Vendar se tak pristop izkaže kot kombinatorično trd oreh, saj je različnih kombinacij postavitve žarnic na prikazovalniku dimenzij N x N kar 2^{N^2}. Za prikazovalnik velikosti 20 x 20 žarnic, bi, ob predpostavki, da za oceno posamezne razporeditve potrebujemo samo en takt 10 GHz procesorja, potrebovali kar 8,18 x 10^{102} let. Kar je precej več, kot znanstveniki ocenjujejo, da je staro naše vesolje (13,782 x 10^9 let).

Problem, ki ga rešujemo, je tipičen optimizacijski problem, kjer z algoritmi preiskujemo prostor hipotez in ocenjujemo kakovost posamezne hipoteze s uporabo kakovostne funkcije (angl. *fitness function*). Za reševanje problema optimizacije cene svetlobnih prikazovalnikov bomo za kakovostno funkcijo uporabili hevristično oceno, razvito v prejšnjem poglavju. Opišimo pa še prostor hipotez.

Model preiskovanega prostora je predstavljen z oštevilčenjem žarnic. Vsako žarnico na prikazovalniku oštevilčimo tako, da velja

št. žarnice = (vrstica – 1) x št. žarnic v vrstici + stolpec - 1. (4)

Žarnica v gornjem levem kotu ima tako oznako 0, žarnica v spodnjem desnem kotu pri velikosti prikazovalnika 20 x 20 pa oznako 399. Posamezno hipotezo v prostoru preiskovanja predstavlja seznam številk žarnic, ki naj bodo na prikazovalniku prisotne.

V opisu algoritmov se bomo večkrat sklicevali na vse žarnice, ki so lahko prisotne na prikazovalniku. Takrat bomo uporabili notacijo *Sign.Bulbs*.

3.1 Genetski algoritmi

Genetski algoritmi (angl. g*enetic algorithms*) si za vzor jemljejo evolucijo življenja. V naravi se svet razvija iz generacije v generacijo s prenašanjem in kombiniranjem lastnosti staršev k njihovim potomcem. Vsak od potomcev podeduje od staršev drugačno kombinacijo njunih genov, določeni geni pa celo niso podedovani, pač pa so spremenjeni. Pravimo, da so mutirani. V posamezni generaciji preživijo samo najboljši predstavniki populacije, torej tisti z za preživetje najbolje prilagojenimi lastnostmi. Slabše prilagojeni predstavniki kmalu odmrejo (ptica, ki ne more leteti, kaj hitro postane plen plenilcev, mladič, ki ni odporen na običajno bolezen, zboli...) in posledično ne morejo posredovati svojega (slabega) genskega zapisa svojim potomcem. Naravna selekcija tako poskrbi za razvoj oz. izboljševanje vrste, saj se povprečna prilagojenost predstavnikov iz roda v rod izboljšuje.

Pri genetskih algoritmih hipotezo poimenujemo kar *osebek*, njene komponente pa *geni*. Algoritem deluje nad seznamom hipotez, ki ga poimenujemo *populacija*. Običajno je začetna populacija ustvarjena iz naključnih osebkov. V vsaki iteraciji se iz populacije stohastično generira nova populacija z uporabo treh mehanizmov evolucije [10]:

- **križanje** (angl. *crossover*): pari osebkov (v nekaterih različicah pa tudi trojice ali n-terice) si izmenjajo del genskega zapisa. V najbolj pogosti različici se določi naključna točka v nizu genov, prvi naslednik podeduje genski niz prvega osebka do presečne točke in genski niz drugega osebka od presečne točke dalje, drugi naslednik pa preostanek genskih nizov,
- **mutacija**: vsak gen v zapisu osebkov se z majhno verjetnostjo spremeni – mutira. Naloga mehanizma mutacije je, da preiskovanje ne obstane v lokalnem minimumu, saj vnaša v populacijo raznolikost ob vsaki iteraciji,
- **reprodukcija**: boljša, kot je kvaliteta določenega osebka, večjo možnost ima za preživetje in nadaljevanje vrste.

Skozi izvajanje algoritma se skupna kvaliteta populacije veča in ob zaključku ostanejo v populaciji hipoteze z najbolj optimalno oceno.

Implementacija algoritma

V nadaljevanju je opisana implementacija algoritma genetskih algoritmov, ki preiskuje po prostoru razporeditev žarnic. Postopek je prikazan kot Algoritem 3.1.

Vhodni parametri:
- N – velikost populacije
- E – odstotek elitnih osebkov
- P_c – verjetnost križanja
- P_m – verjetnost mutacije

Funkcije:
- *generate_random_hypothesis(N):* generira N naključnih razporeditev žarnic.
- *sort(seznam):* razvrsti razporeditve v seznamu padajoče po oceni.
- *random():* vrne naključno število iz intervala [0, 1).
- *random(n):* vrne naključno celo število iz intervala [1, n].
- *length(seznam):* vrne dolžino seznama.
- *crossover(H_1, H_2, $cross_1$, $cross_2$):* ustvari novo razporeditev žarnic, v njej so vsebovane žarnice prve razporeditve (H_1) do žarnice $cross_1$ in žarnice druge razporeditve (H_2) od žarnice $cross_2$ naprej.
- *GetRandomSubject(seznam razporeditev):* vrne naključno razporeditev iz seznama. Pri tem imajo razporeditve z boljšo oceno sorazmerno večjo možnost za izbor.
- *bestTwo(seznam razporeditev):* vrne tisti dve razporeditvi iz seznama, ki imata najboljšo vrednost ocene.

Algoritem 3.1 Genetski algoritmi

```
function GeneticSearch(N : integer, E, Pc, Pm : real) : Hypothesis;
var
H : array [1 .. N] of Hypothesis;  (* Trenutna populacija *)
H' : array [1 .. N] of Hypothesis; (* Nasledna populacija *)
j : integer                        (* Števec nove populacije *)
H1, H2, H1', H2' : Hypothesis;     (* Starša in naslednika *)
begin
  H := generate_random_hypothesis(N);
  sort(H);
  repeat
    j := 0;
```

```
  for i := 1 to E * N do                          (* Vsi elitni osebki se obvezno uparijo *)
  begin
    H1 := H[i];
    H2 := GetRandomSubject(H);
    cross1 := random(length(H1));
    cross2 := random(length(H2));
    H1' := crossover(H1,H2, cross1, cross2);
    H2' := crossover(H2,H1, cross2, cross1);
    H' := H' ∪ bestTwo(H1, H2, H1', H2');
    H := H / {H1, H2};
    j := j + 2;
  end
  repeat                                          (* Upari se še preostali delež osebkov *)
    H1 = GetRandomSubject(H);
    H2 = GetRandomSubject(H);
    cross1 := random(length(H1));
    cross2 := random(length(H2));
    H1' := crossover(H1,H2, cross1, cross2);
    H2' := crossover(H2,H1, cross2, cross1);
    H' := H' ∪ bestTwo(H1, H2, H1', H2');
    H := H / {H1, H2};
    j := j + 2;
  until j > Pc * N
  for i := 1 to length(H) do                      (* Mutacija genov preostalih osebkov *)
  begin
    H1 := H[i];
    foreach Bulb in Sign.Bulbs do
      if random() < Pm then if Bulb ∈ H1 then H1 := H1 / Bulb else H1 = H1 ∪ Bulb;
    H' := H' ∪ H1;
  end;
  sort(H);
 until stop_condition;
 return(H[1]);
end
```

Algoritem na začetku izbere *N* naključnih razporeditev žarnic - osebkov. Ti predstavljajo začetno populacijo. Z vsako iteracijo (generacijo) se izvedejo sledeči koraki:

- iz populacije se izloči množica elitnih osebkov. To je prvih *E* x *N* osebkov, sortiranih po kvaliteti od najboljšega do najslabšega,
- vsakemu elitnemu osebku se naključno izbere partner iz preostalih osebkov v populaciji. Pri tem je verjetnost, da bo nek osebek izbran za uparitev, premo sorazmerna z njegovo kvaliteto,
- ko so uparjeni vsi elitni osebki, se uparijo še preostali osebki. Pri tem zopet velja, da je verjetnost izbora za parjenje sorazmerna s kvaliteto osebka. Uparjevanje poteka, dokler se ne upari P_c x *N* osebkov (všteti tudi elitni osebki),
- nad vsakim parom osebkov se izvede križanje. Pri tem nastane iz vsakega para množica štirih osebkov – originalna osebka (starša) in naslednika (otroka). V populacijo naslednje generacije se vključita tista osebka iz množice teh štirih, ki imata najboljšo kvaliteto. Križanje poteka tako, da se v vsakem staršu določi druga naključna presečna točka. Prvi otrok dobi vse gene (žarnice) prvega starša do njegove presečne točke in vse žarnice drugega starša od presečne točke drugega starša naprej. Drugi otrok dobi preostanek žarnic (prvega starša od presečne točke naprej in drugega starša do presečne točke). Če je neka žarnica prisotna v obeh starših, jo podedujeta oba otroka,

- preostanek populacije mutira. V vsakem osebku ima vsaka žarnica s prikazovalnika verjetnost P_m, da se vključi v osebek, če še ni vključena oz., da se odstrani iz osebka, če je že vključena.

Velikost populacije se ne spreminja. Sam algoritem križanja je nastavljen tako, da iz vsakega para staršev natane natanko en par potomcev. Ob koncu izvajanja je najbolje ocenjena razporeditev žarnic v zadnji populaciji cilj preiskovanja.

3.2 Iskanje harmonije

Zong Woo Geem, Joong Hoon Kim in G.V. Loganathan so leta 2001 objavli članek [7], v katerem so predstavili nov stohastičen algoritem iskanja harmonije (angl. *harmony search*). Ta, podobno kot mnogi algoritmi, temelji na pojavu, opaženem v naravi. Z algoritmom so posnemali improvizacijo glasbenikov v iskanju melodične harmonije ob igranju Jazza.

Kvaliteta zvoka glasbenega instrumenta je načeloma odvisna od višine tona, barve glasu in glasnosti zvoka. Vendar je zmožnost igranja akordov odvisna predvsem od frekvenčnega razpona določenega instrumenta. Harmonija je hkratno usklajeno igranje večih različnih tonov. Že Pitagora je odkril, da harmonijo tvorijo oktave v razmerju 1:2 ali pa naprimer toni z razmerjem 2:3. Vendar je težko verjetno, da bi lahko z igranjem naključnih tonov zaigrali prijetno harmonijo.

Ko izkušen glasbenik improvizira, ima na voljo tri možosti – iz spomina lahko natančno zaigra naučeno melodijo, lahko zaigra melodijo, ki je podobna naučeni malodiji, ali pa zaigra skladbo naključnih tonov. Podobno tudi algoritem iskanja harmonije tvori nove hipoteze s kombinacijo uporabe znanja iz spomina, lokalnega prilagajanja in naključnega preiskovanja.

Algoritem začne s seznamom naključnih hipotez – spominom naučenih melodij. V vsaki iteraciji sestavi novo hipotezo tako, da za vsako komponento hipoteze:

a) z verjetnostjo P_{accept} izbere vrednost komponente iz ene od hipotez v spominu in z verjetnostjo 1 - P_{accept} dodeli komponenti naključno vrednost. Običajno je vrednost parametra P_{accept} med 0,75 in 0,95 [13]. Če je vrednost premajhna, se iskanje obnaša preveč naključno in zato konvergira prepočasi. Če je vrednost prevelika, potem se nove hipoteze generirajo predvsem iz starih hipotez in je preiskovanje novega prostora premajhno,
b) če je vrednost komponente izbrana iz spomina, jo algoritem z verjetnostjo P_{pa} še malenkostno spremeni, največ za vrednost parametra b_{range}. V glasbenem svetu je to enakovredno majhni prilagoditvi frekvence tona. Nova vrednost komponente se tako izračuna po enačbi

$$x_{new} = x_{old} + b_{range} * \varepsilon, \quad (5)$$

kjer je x_{old} vrednost komponente iz spomina, x_{new} vrednost komponente po prilagajanju, b_{range} največja dovoljena vrednost prilagoditve in ε naključna spremenljivka. Namen takega prilagajanja komponent je predvsem v lokalni optimizaciji hipotez – preiskovanju prostora okrog dobrih rešitev.

Če je kvaliteta novo ustvarjene hipoteze večja od najslabše ocenjene hipoteze v spominu, potem novo ustvarjena hipoteza nadomesti tisto z najslabšo kvaliteto, sicer se zavrže.

Implementacija algoritma

V nadaljevanju je opisana naša implementacija algoritma iskanja harmonije, ki preiskuje po prostoru razporeditev žarnic. Postopek je prikazan kot Algoritem 3.2.

Vhodni parametri:

- P_{accept} – spominski koeficient
- P_{pa} – koeficient prilagajanja
- N – velikost spomina

Funkcije:

- *generate_random_hypothesis(n):* vrne n naključno generiranih razporeditev žarnic.
- *random():* vrne naključno število iz intervala [0, 1).

Algoritem 3.2 Iskanje harmonije

```
function HarmonySearch(P_accept, P_pa : real; N : integer) : Hypothesis
var
H_new : Hypothesis;                (* Nova razporeditev *)
H_min : Hypothesis;                (* Najslabša razporeditev v spominu *)
M : array [1 .. N] of Hypothesis;  (* Spomin *)
include : boolean;                 (* Ali naj se žarnica vključi v razporeditev *)
totalQuality, Q : real;
k : integer;
begin
  M := generate_random_hypothesis(N);
  repeat
    foreach Bulb in Sign.Bulbs do
    begin
      if random() < P_accept then
      begin
        totalQuality := sum_{H,H∈M} (q(H));
        Q := random() * totalQuality;
        k = -1;
        while Q >= 0 do
        begin
          k := k + 1;
          Q := Q – q(M[k]);
        end
        include := Bulb ∈ M[k];
        if random() < P_pa then include := not include;
        if include = true then H_new := H_new ∪ Bulb;
      end
      else if random() < 0.5 then H_new := H_new ∪ Bulb;
    end
    H_min := arg min_{H,H∈M} q(H);
    if q(H_new) > q(H_min) then M := (M / H_min) ∪ H_new;
  until stop_condition
  return( arg max_{H,H∈M} q(H) );
end
```

Algoritem na začetku izbere *N* naključnih razporeditev žarnic in jih shrani v seznam – spomin. V vsaki iteraciji ustvari novo razporeditev in, če njena ocena prekaša najslabše

ocenjeno razporeditev iz spomina, slednjo nadomesti z novo razporeditvijo. Sicer jo zavrže. Velikost spomina se ne spreminja in ves čas vsebuje natanko N razporeditev.

Novo razporeditev algoritem ustvari tako, da se za vsako žarnico na prikazovalniku odloči, ali jo bo vključil v novo razporeditev glede na njeno prisotnost v spominu. Verjetnost za tako odločitev je enaka vhodnemu parametru P_{accept}. Tedaj naključno izbere eno izmed razporeditev iz spomina, pri tem pa imajo boljše ocenjene razporeditve premo sorazmerno večjo verjetnost za izbor. Če je žarnica v izbrano razporeditev vključena, jo vključi tudi v novo razporeditev, sicer ne. Z verjetnostjo P_{pa} se izvede še prilagajanje. Takrat se v novo razporeditev vključena žarnica izključi in obratno.
Kadar se vključevanje žarnice v novo razporeditev ne izvaja glede na njeno prisotnost v spominu, se žarnico vključi v novo razporeditev povsem naključno. Tedaj je verjetnost, da se žarnica vključi v razporeditev, enaka 0,5.
Ob zaključku preiskovanja je rezultat algoritma najboljše ocenjena razporeditev v spominu.

3.3 Optimizacija z roji delcev

Algoritem optimizacije z roji delcev (angl. *particle swarm optimization*) je začel svoj razvoj kot model, s katerim bi grafično simulirali let jate ptic. Tekom razvoja modela so se avtorji zavedli, da pravzaprav razvijajo optimizacijski algoritem.
Osnovna ideja je precej preprosta [6]. Algoritem prične z množico naključno generiranih hipotez. Množico poimenujemo *roj*, posamezno hipotezo pa *delec*. Vsakemu delcu je pripisana hitrost, s katero se giblje v preiskovanem prostoru d dimenzij. Iteracija algoritma sestoji iz dveh delov:

1. Vsak delec prilagodi svoj vektor hitrosti (pospeši), delno proti točki, v kateri je delec dosegel svojo najboljšo oceno (p_i) in delno proti točki, v kateri je kateri koli delec iz roja dosegel najboljšo oceno preiskovanja (p_g). V enačbi pospeševanja je trenutna hitrost utežena s parametrom w, sprememba proti osebno najboljši točki je utežena s parametrom c_1, sprememba proti globalno najboljši točki pa je utežena s parametrom c_2. Nekatere implementacije imajo hitrost navzgor omejeno s parametrom v_{max}. Namen parametra je preprečiti, da bi posamezni delci preveč pospešili in preleteli optimalno rešitev. Vendar pa, če je omejitev prenizka, se lahko zgodi, da delci raziščejo premajhen del prostora in tako izvedejo zgolj lokalno optimizacijo namesto iskanja najboljše hipoteze.

   Enačba pospeševanja, ko se delec X_i nahaja v točki x_i, je enaka
   $$v_{id} = w * v_{id} + c_1 * \text{random} * (p_{id} - x_{id}) + c_2 * \text{random} * (p_{gd} - x_{id}). \quad (6)$$

2. Vsak delec izračuna svoje novo mesto v prostoru po enačbi
   $$x_{id} = x_{id} + v_{id}. \quad (7)$$
   Na novi poziciji se izračuna kvaliteta in osveži delčeva najboljša točka in globalno najboljša točka, če ocena nove hipoteze presega stare.

Za preiskovanje binarnih prostorov, kjer vsaka pozicija predstavlja kombinacijo d binarnih spremenljivk, je potrebno algoritem prilagoditi. Hitrost poda zgolj verjetnost, da spremenljivka preide iz stanja 0 v stanje 1 (ali obratno). Z večanjem hitrosti se veča verjetnost, da bo spremenljivka spremenila svojo vrednost.

Enačba za izračun nove pozicije se zato spremeni [9] iz (7) v

$$x_{id} = \begin{cases} 1, & random < s(v_{id}) \\ 0, & sicer \end{cases}, \tag{8}$$

pri tem pa je

$$s(v_{id}) = \frac{1}{1+e^{-v_{id}}}. \tag{9}$$

Implementacija algoritma

V nadaljevanju je opisana naša implementacija algoritma optimizacije z roji delcev. Postopek je prikazan kot Algoritem 3.3.

Vhodni parametri:

- N – velikost roja
- c_1 – utež pospeševanja k osebno najboljši točki v prostoru
- c_2 – utež pospeševanja k globalno najboljši točki v prostoru
- w – utež ohranjanja trenutne hitrosti

Funkcije:

- *generate_random_hypothesis(n):* vrne n naključno ustvarjenih razporeditev žarnic.
- *random():* vrne naključno število iz intervala [0, 1).

Algoritem 3.3 Optimizacija z roji delcev

```
function ParticleSwarmOptimization(w, c1, c2 : real; N : integer) : Hypothesis
var
Hgbest = Hypothesis;                          (* Globalno najboljša razporeditev *)
Hpbest : array [1 .. N] of Hypothesis;        (* Seznam osebno najboljših razporeditev *)
H : array [1 .. N] of Hypothesis;             (* Roj delcev *)
V : array [1 .. N, 1 .. Sign.Bulbs.Count] of real;  (* Seznam hitrosti *)
Hnew : Hypothesis;                            (* Nova razporeditev *)
s, v : real;
begin
 H := generate_random_hypothesis(N);
 Hbest := arg max_{H',H'∈H} q(H'));
 for i := 1 to N do foreach Bulb in Sign.Bulbs do V[i, Bulb] := random();
 repeat
  for i := 1 to N do
  begin
   foreach Bulb in Sign.Bulbs do
   begin
    if Bulb ∈ H[i] then xid := 1 else xid = 0;
    if Bulb ∈ Hpbest[i] then pid := 1 else pid = 0;
    if Bulb ∈ Hgbest then pgd := 1 else pgd = 0;
    v := w * V[i, Bulb] + c1 * random() * (xid – pid) + c2 * random() * (xid – pgd);
    s := 1/(1+e^{-V});
    if random() < s then Hnew := Hnew ∪ Bulb;
    V[i, Bulb] := v;
   end
   if q(Hnew) > q(H[i]) then H[i] := Hnew;
   if q(Hnew) > q(Hgbest) then Hgbest := Hnew;
  end
 until stop_condition
 return(Hgbest);
end
```

Algoritem začne z generiranjem roja. Naključno ustvari N delcev – razporeditev žarnic in vsakemu delcu pripiše svoj vektor hitrosti. Velikost vektorja hitrosti je enaka številu mest na prikazovalniku. Vsak delec vodi za vsako žarnico na prikazovalniku svojo hitrost. Ta ima na začetku naključno vrednost. Poišče se delec z najbolje ocenjeno razporeditvijo. Ta razporeditev postane globalno najboljša in se ji vsi ostali delci pričnejo približevati.

V vsaki iteraciji algoritem vsakemu delcu izračuna novo razporeditev žarnic: za vsako žarnico na prikazovalniku ugotovi, ali je prisotna v trenutni razporeditvi žarnic, v globalno najboljši razporeditvi in v delčevi osebno najboljši razporeditvi. Glede na te tri prisotnosti žarnice in glede na hitrost, ki jo delec za dotično žarnico vodi, po enačbah (6), (8) in (9) določi, ali naj bo žarnica prisotna v novi razporeditvi ali ne. Hkrati za vsako žarnico shrani njeno novo hitrost v delčev vektor hitrosti. Če delec z oceno svoje nove razporeditve žarnic preseže oceno svoje najboljše razporeditve, potem nova razporeditev postane delčeva nova osebno najboljša razporeditev. Če ocena nove razporeditve preseže oceno globalno najboljše razporeditve, nova razporeditev postane tudi nova globalno najboljša razporeditev.

3.4 Požrešno iskanje z naključnimi ponovitvami

Požrešno iskanje z naključnimi ponovitvami (angl. *greedy search with random restarts*) je različica navadnega požrešnega iskanja [10], kjer se izvajanje algoritma požrešnega iskanja v enem preiskovanju prostora večkrat ponovi. Požrešno iskanje pri preiskovanju prostora hipotez v vsaki iteraciji razvije samo tistega naslednika H_{max} trenutne hipoteze H, ki maksimizira vrednost kvalitete $H_{\max} = \arg\max_{H', H \to H'} q(H')$. Vsi preostali nasledniki se ne raziščejo.

Požrešno iskanje torej vedno razišče samo tisto poddrevo v drevesu naslednikov, kjer je vrh poddrevesa najboljši od vseh naslednikov. Pri tem se ne ozira na preteklo preiskovanje ali na to, da si lahko z razvojem lokalno sicer najbolj obetavnega naslednika zapre pot v obetavnejše dele prostora. Velikanska slabost algoritma je ta, da redko kdaj najde optimalno hipotezo problema. Način, kako preiskuje prostor, ga sili k temu, da se hitro ujame v lokalni optimum. Zato algoritem večkrat ponovimo in s tem povečamo možnost, da najde dober oz. boljši lokalni optimum.

Požrešno preiskovanje se daleč najbolje odreže takrat, kadar se da problem razdeliti na podprobleme in je optimalna hipoteza osnovnega problema sestavljena iz samih optimalnih hipotez podproblemov. Primer učinkovitega požrešnega algoritma je Dijkstrin algoritem za iskanje najkrajše poti v grafu [4].

Implementacija algoritma

V nadaljevanju je opisana naša implementacija algoritma požrešnega iskanja z naključnimi ponovitvami. Postopek je prikazan kot Algoritem 3.4.

Funkcije:

- *generate_random_hypothesis():* generira naključno razporeditev žarnic.

Algoritem 3.4 Požrešno iskanje z naključnimi ponovitvami

```
function GreedySearch() : Hypothesis;
var
H_best : Hypothesis;          (* Najboljša odkrita razporeditev *)
H_Lbest : Hypothesis;         (* Najboljša odkrita razporeditev trenutnega izvajanja *)
H : Hypothesis;               (* Trenutna razporeditev *)
H' : Hypothesis;              (* Naslednik trenutne razporeditve *)
begin
  H_best := generate_random_hypothesis();
  repeat
    H_Lbest := generate_random_hypothesis();
    repeat
      H := H_Lbest;
      foreach Bulb in Sign.Bulbs do
      begin
        if Bulb ∈ H then H' := H / Bulb; else H' := H ∪ Bulb;
        if q(H') > q(H_Lbest) then H_Lbest := H';
      end
    until H = H_Lbest;
  if q(H_Lbest) > q(H_best) then H_best := H_Lbest;
  until stop_condition
  return(H_best);
end
```

Algoritem v vsaki ponovitvi prične z naključno generirano razporeditvijo žarnic. V posamezni iteraciji predstavljajo množico naslednikov trenutne razporeditve *H* take razporeditve žarnic *H'*, ki se od *H* razlikujejo v prisotnosti natanko ene žarnice. Algoritem v trenutno razporeditev žarnic vsako žarnico iz prikazovalnika ali vključi, če v njej še ni vključena, ali pa izključi, če je že. Nato vsako tako ustvarjeno razporeditev oceni. Najbolje ocenjena razporeditev *H'* postane *H* v naslednji iteraciji.

Algoritem konča s preiskovanjem, ko se ocene razporeditve žarnic ne da več izboljšati s spremembo prisotnosti samo ene žarnice. Tedaj preveri, če ni najboljša razporeditev žarnic trenutne ponovitve boljša od najboljše najdene razporeditve v vseh ponovitvah. V kolikor je, si jo zapomni. Preiskovanje nato ponovi in ga ponavlja, dokler ni izpolnjen kriterij, ki ustavi izvajanje (na primer čas preiskovanja doseže mejno dolžino).

3.5 Iskanje s tabuji

Algoritem iskanja s tabuji (angl. *tabu search*) je algoritem lokalne optimizacije. V vsakem koraku preiskovanje iz trenutne hipoteze *H* razvije tisto hipotezo *H'*, ki ima v preiskovanem prostoru v lokalni okolici hipoteze *H* najboljšo oceno. Da pa bi algoritem preiskal tudi tiste dele prostora, ki bi zaradi lokalne naravnanosti preiskovanja ostali neraziskani, algoritem ves čas spreminja razumevanje pojma lokalnosti. Z izkoriščanjem različnih metod definira, katere hipoteze se smatrajo za okolico hipoteze *H*.

Najbolj osnovna metoda je seznam tabujev [8]. Ta je mišljen kot kratkoročni spomin, v katerem algoritem ves čas vodi seznam hipotez, ki jih je algoritem pred kratkim obiskal (največ *n* korakov nazaj, kjer je *n* dolžina seznama). Algoritem te hipoteze izloči iz okolice hipoteze *H*. Včasih iskanje s tabuji v seznam tabujev ne shrani celotnih hipotez, ampak samo prepovedane prehode v preiskovanem prostoru ali pa sestavne dele hipotez.

Takšen način preiskovanja omogoča, da se iskanje izkoplje iz lokalnega optimuma – takrat so vse sosednje hipoteze obvezno slabše od trenutne, vendar je algoritem prisiljen sprejeti najboljšo izmed njih. Brez seznama tabujev bi se iskanje takoj za tem spet vrnilo nazaj v lokalni optimum, vendar, ker je bil ta prehod v zadnjem koraku shranjen v seznam, ni več dovoljen. Velikost seznama tabujev tako določa, kako globoke lokalne optimume lahko algoritem zaobide.

Hkrati pa seznam tabujev lahko privede do druge skrajnosti. Kadar se v seznam doda določena komponenta hipoteze, se s tem iz preiskovanja lahko izloči večji del prostora. V tem delu prostora, ki se mu algoritem sedaj izogiba, pa se lahko nahajajo ravno najboljše hipoteze. Zato se algoritem dostikrat razširi tako, da se mu v določenih primerih omogoči ignoriranje elementa iz seznama tabujev. Najbolj pogost tak primer je ta, da algoritem dovoli prepovedan premik, kadar ta najde hipotezo, ki je boljša od trenutno najboljše najdene.

Implementacija algoritma

V nadaljevanju je opisana naša implementacija algoritma iskanja s tabuji. Postopek je prikazan kot Algoritem 3.5.

Vhodni parametri:

- N – velikost seznama tabujev

Funkcije

- *generate_random_hypothesis():* generira naključno razporeditev žarnic.
- *length(seznam):* vrne dolžino seznama.

Algoritem prične z naključno generirano hipotezo. Ves čas izvajanja vodi seznam tabujev, v katerem je shranjenih največ *N* žarnic. Na začetku je ta seznam prazen. Posamezen korak je podoben algoritmu požrešnega iskanja. Iz trenutne razporeditve žarnic se ustvari množica naslednikov s tem, da se posamezna žarnica s prikazovalnika v razporeditev naslednika doda, če v trenutni razporeditvi ni prisotna, ali pa se iz razporeditve naslednika odstrani, če je v trenutni razporeditvi prisotna. Iz množice naslednikov so izločene tiste razporeditve žarnic, ki bi se od trenutne razlikovale po prisotnosti žarnice, ki je v seznamu tabujev. Razporeditve žarnic v množici naslednikov trenutne razporeditve se razlikujejo v prisotnosti natanko ene žarnice.

Tista razporeditev žarnic izmed naslednikov, ki ima najboljšo oceno, se sprejme in postane osnovna razporeditev v naslednji iteraciji. Pri tem se žarnica, ki se je v novo razporeditev dodala ali odstranila, doda na konec seznama tabujev. S tem se njena prisotnost v razporeditvah naslednjih *N* iteracij ne sme več spremeniti. Ko seznam tabujev preseže *N* elementov, se prva žarnica iz seznama odstrani in tako postane prosta za spreminjanje njene prisotnosti v naslednikih. Kadar ocena razporeditve žarnic enega izmed naslednikov preseže oceno najboljše razporeditve, razporeditev postane nova najboljša razporeditev žarnic.

Algoritem 3.5 Iskanje s tabuji

```
function TabuSearch( N : integer) : Hypothesis;
var
H_best : Hypothesis;        (* Najboljša odkrita razporeditev *)
H : Hypothesis;             (* Trenutna razporeditev *)
H' : Hypothesis;            (* Naslednik trenutne razporeditve *)
```

```
Tabu : array [1 .. N] of integer;          (* Seznam prepovedanih sprememb *)
Bulb_tabu : integer;                       (* Žarnica najboljšega naslednika *)
H_tabu : Hypothesis;                       (* Najboljši naslednik *)
first = boolean;
begin
  H := generate_random_hypothesis();
  repeat
    first := true;
    foreach Bulb in Sign.Bulbs do
    begin
      if Bulb ∉ Tabu then
      begin
        if Bulb ∈ H then H' := H / Bulb; else H' := H ∪ Bulb;
        if q(H') > q(H_best) then H_best := H';
        if first = true OR q(H') > q(H_tabu) then
        begin
          first := false;
          H_tabu := H';
          Bulb_tabu := Bulb;
        end
      end
    end
    Tabu := Tabu ∪ Bulb_tabu;
    if length(Tabu) > N then Tabu := Tabu / Tabu[1];
    H := H_tabu;
  until stop_condition;
  return(H_best);
end
```

3.6 Skrajna optimizacija

Tudi algoritem skrajne optimizacije (angl. *extremal optimization*) se zgleduje po naravi. V naravi se posamezne enote povezujejo v kohezivne strukture z namenom optimizacije porabe resursov. V tem procesu se, podobno kot pri prodiranju vode po pobočju, podirajo tiste ovire, ki so najšibkejše, obstanejo pa najmočnejše in nenamerno hkrati najbolj prilagojene. Optimalnost se tako pojavi naravno, s postopkom naravne negativne selekcije proti najšibkejšim členom strukture in to naravno selekcijo algoritem posnema z odstranjevanjem najslabših komponent hipotez.

Stefan Boettcher in Allon G. Percus sta s principom naravne selekcije v mislih zasnovala algoritem za preiskovanje težkih optimizacijskih problemov [3]. Pri tem sta se opirala na Bak – Sneppnov model biološke evolucije. Ta predpostavlja, da so posamezne vrste med seboj neopredeljivo povezane. V modelu je vsaka vrsta postavljena na rob mreže povezav in ima pripisano oceno kvalitete – številko med 0 in 1. V vsakem koraku se vrste uredi glede na oceno in tisti z najslabšo oceno (najslabše prilagojena vrsta) se le-ta zamenja z novo, naključno izbrano oceno. Ker pa sprememba ene vrste vpliva na vse povezane vrste, se tudi vsem povezanim vrstam zamenja ocena. Ta korak se nato ponavlja poljubno dolgo. Po določenem številu korakov celoten sistem preide v visoko korelirano stanje poznano tudi kot samoorganizirana kritičnost (angl. *self-organized criticality*). V tem stanju imajo vse vrste pripisano kvaliteto nad neko mejo. Vendar pa lahko oslabitev le ene od povezanih vrst povzroči spremembo vsake vrste, to pa lahko sproži verižne reakcije, kjer se spremenijo večji deli sistema, s čimer je mogoča praktično vsaka ureditev sistema.

Po vzoru tega modela sta Boettcher in Pecus zasnovala algoritem skrajne optimizacije. Algoritem začne z naključno izbrano hipotezo. Hipoteza je sestavljena iz posameznih komponent, pri tem pa se da vsaki komponenti x_i pripisati oceno, koliko prispeva s skupni kvaliteti hipoteze. V vsakem koraku se najslabši del hipoteze nadomesti z drugo, naključno izbrano komponento. Ker pa se v primeru, ko algoritem preiskuje prostor binarnih komponent, kjer vsaka komponenta ali je ali pa ni prisotna v hipotezi, tako zastavljen algoritem slej ko prej ustavi v lokalnem minimumu, avtorja algoritem razširita s parametrom τ. In sicer se za zamenjavo lahko izbere z verjetnostjo

$$P(j = k) = k^{-\tau} \quad (10)$$

vsaka komponenta x_j hipoteze, pri čimer je k zaporedna številka komponente v urejeni vrsti po oceni.

Algoritem je na prvi pogled podoben požrešnemu preiskovanju. Požrešno preiskovanje v vsakem koraku stremi k temu, da največ, kar se da, izboljša trenutno hipotezo. In kako to storiti lažje, če ne z zamenjavo najslabše komponente hipoteze, saj ta skupno kvaliteto najbolj ovira. Vendar se v tej točki podobnost med algoritmoma konča. Medtem, ko bi požrešni algoritem najslabšo komponento zamenjal z najboljšim nadomestkom, algoritem skrajne optimizacije poišče nadomestno komponento povsem naključno. V razširjeni različici pa algoritem celo ne menja vedno najslabše komponente, pač pa z določeno majhno verjetnostjo zamenja tudi manj slabe komponente.

Implementacija algoritma

V nadaljevanju je opisana naša implementacija algoritma skrajne optimizacije. Postopek je prikazan kot Algoritem 3.6.

Vhodni parametri:

- Thau – koeficient izbora naslednika

Funkcije:

- *generate_random_hypothesis():* vrne naključno razporeditev žarnic.
- *sort(seznam):* razvrsti razporeditve v seznamu naraščajoče po oceni.
- *random():* vrne naključno število iz intervala [0, 1).
- *random(n):* vrne naključno celo število iz intervala [1, n].
- *length(seznam):* vrne dolžino seznama.

Algoritem prične z naključno generirano razporeditvijo žarnic po prikazovalniku. V vsaki iteraciji določi oceno kvalitete vsaki žarnici na prikazovalniku. Po priporočilu [1] se ocena kvalitete posamezne žarnice izračuna tako, da se žarnica ali doda k trenutni razporeditvi H, če še ni vključena, ali pa odstrani, če je že vključena, in tako ustvari sosednjo razporeditev žarnic H'. Razlika med oceno razporeditve žarnic H' in oceno trenutne razporeditve je enaka doprinosu žarnice k kvaliteti. Ker pa si razlike ocen sledijo po vrsti popolnoma enako, kot ocene razporeditev H', preskočimo nepotrebni korak računanja razlike in zato za oceno kvalitete žarnice vzamemo kar oceno razporeditve H'.

Algoritem 3.6 Skrajna optimizacija

```
function ExtremalOptimization(Thau : real) : Hypothesis;
var
H0 : hypothesis;                    (* Trenutna razporeditev *)
H' : hypothesis;                    (* Naslednik trenutne razporeditve *)
Hbest : hypothesis;                 (* Najboljša razporeditev *)
fitness : array[1 .. N] of Hypothesis;   (* Seznam naslednikov *)
begin
  H0 := generate_random_hypothesis();
  repeat
    foreach Bulb in Sign.Bulbs do
    begin
      if Bulb ∈ H0 then H' := H0 / Bulb else H' = H0 ∪ Bulb;
      fitness := fitness ∪ H';
    end
    sort(fitness);
    repeat
      i := random(length(Sign.Bulbs));
    until (random() < i -Thau)
    H0 := fitness[i];
    if q(H0) > q(Hbest) then Hbest := H0;
  until stop_condition
  return(Hbest);
end
```

Osnovni algoritem narekuje, da po tem, ko identificiramo najslabšo komponento hipoteze, le to zamenjamo z drugo, naključno komponento. Ker so komponente razporeditve žarnic prisotnosti in odsotnosti posameznih žarnic v razporeditvi, se komponenta lahko nadomesti le z lastno negacijo, torej, ali določeno žarnico izključimo ali pa vključimo. Ob takem nadomestilu najslabše komponente pa bi se ustvarila točno enaka razporeditev žarnic, s katero ocenimo vrednost komponente – žarnice. Tako tudi ta korak preskočimo in zato algoritem razvrsti po vrsti kar sosednje razporeditve žarnic in ne posameznih žarnic. Razporedi jih po oceni, naraščajoče.

Nato se naključno izbere *i*-ta sosednja razporeditev v vrsti in sicer z verjetnostjo $i^{-\tau}$. Tako izbrana razporeditev se sprejme in postane naslednja trenutna razporeditev *H*. Če ocena nove razporeditve žarnic boljša od najboljše razporeditve žarnic, nova razporeditev postane nova najboljša razporeditev. Za vrednost τ smo po priporočilu [2] uporabili vrednost 1,3.

3.7 Simulirano ohlajanje

Algoritem simuliranega ohlajanja (angl. *simulated annealing*) je dobil ime po postopku v metalurgiji, pri katerem surovino segrevajo in nadzorovano ohlajajo. S tem izboljšajo kvaliteto sestave materiala. Ob segrevanju se namreč atomi osvobodijo molekulskih vezi in prosto gibljejo, počasno ohlajanje pa jim daje več priložnosti, da najdejo stanje z manjšo energijo.

Algoritem simulira ta proces [10, 5] tako, da v vsaki iteraciji izmed vseh naslednikov hipoteze *H* izbere naključnega naslednika *H'*. Pri tem velja, da naslednika vedno sprejme, če je njegova kvaliteta boljša od kvalitete originalne hipoteze. Če pa je kvaliteta slabša, ga algoritem sprejme z verjetnostjo

$e^{\frac{q(H')-q(H)}{T}}$, (11)

pri čemer je T temperatura sistema. Čim večja, kot je temperatura, oz. manj, kot je slabša nova hipoteza H', tem večja je verjetnost, da jo bo algoritem sprejel. V kolikor algoritem sprejme hipotezo H', ta služi za osnovno hipotezo naslednje iteracije, če pa jo zavrne, v naslednji iteraciji zopet izbira med nasledniki hipoteze H. Temperatura sistema se počasi niža in počasneje kot se niža, večji del prostora bo algoritem preiskal in večja je verjetnost, da bo algoritem našel optimalno stanje. Običajno se temperatura sistema spreminja po enačbi

$T = \lambda T$, (12)

pri tem pa je $\lambda < 1$.

S tem, ko algoritem dovoli sprejemati hipoteze, ki imajo slabšo kvaliteto, omogoči, da se iskanje premakne iz lokalnega optimuma, kadar se znajde v njem. Nižja, kot je temperatura sistema, vedno bolj bo algoritem sprejemal zgolj tiste naslednike, ki izboljšajo trenutno hipotezo ali pa jo le malenkostno poslabšajo. Proti koncu, ko je temperatura sistema že zelo majhna, se torej algoritem vede vedno bolj deterministično, torej kot navadna lokalna optimizacija.

Implementacija algoritma

V nadaljevanju je opisana naša implementacija algoritma skrajne optimizacije. Postopek je prikazan kot Algoritem 3.7.

Parametri:

- T – število iteracij ohlajanja

Funkcije:

- *generate_random_hypothesis():* vrne naključno razporeditev žarnic.
- *random():* vrne naključno število iz intervala [0, 1).
- *random(n):* vrne naključno celo število iz intervala [1, n].

Algoritem prične z naključno generirano razporeditvijo žarnic. To v vsakem koraku izvajanja spremeni tako, da naključno žarnico ali odstrani iz razporeditve, če je v njej vsebovana, ali pa jo doda, če še ni vsebovana. Če je ocena nove razporeditve žarnic boljša od prejšnje, se nova razporeditev sprejme za osnovo v naslednjem koraku. Če pa je ocena nove razporeditve slabša od ocene prejšnje razporeditve, jo algoritem sprejme z verjetnostjo, izračunano po enačbi (11). Verjetnost sprejema je torej odvisna od temperature, v kateri se sistem nahaja. V tej točki se naš algoritem razlikuje od standardne inačice. Začetna temperatura namreč ni vhodni podatek, pač pa je vedno enaka 0,1. Tudi končna temperatura je določena na 10^{-7}. Vhodni parameter nadzoruje zgolj to, kako hitro se bo temperatura ohlajala od začetne proti končni vrednosti.

Razlika v kvaliteti dveh hipotez, ki se razlikujeta le v eni žarnici, je v slabem primeru reda 0,01, v dobrem primeru, ko se kvaliteta poslabša minimalno, pa reda 10^{-5}. Za algoritem želimo, da na začetku z veliko verjetnostjo (0,9) sprejema tako velika kot mala poslabšanja kvalitete razporeditve. Ker je $e^{-x} > 0{,}9$ za $x < 0{,}1$, razlike kvalitet v enačbi (11) ne smemo deliti s temperaturo manjšo od 0,1. Hkrati ne želimo, da bi se iskanje preveč časa obnašalo popolnoma naključno, zato začetna temperatura ne sme biti dosti večja od 0,1. Glede na to, da je začetna hipoteza generirana naključno, bi bilo dolgotrajno popolnoma naključno preiskovanje brezpredmetno, saj je verjetnost, da se že na začetku generira neka razporeditev

žarnic enaka verjetnosti, da se taka rešitev pojavi po določenem času popolnoma naključnega preiskovanja.

Na podoben način določimo tudi končno temperaturo. Ker je $e^{-x} < 0{,}001$ za $x > 6{,}9$ in želimo, da bi v zadnji stopnji preiskovanja algoritem sprejemal čim manj tudi samo malo slabših rešitev (z verjetnostjo manjšo od 0,001), mora biti temperatura sistema manjša od 10^{-6}. Da pa se preiskovanje ne bi ustavilo takoj, ko se verjetnost sprejema slabše rešitve pade pod 0,001, končno temperaturo postavimo na 10^{-7}. Tako se algoritem nekaj časa obnaša skoraj povsem deterministično. V vsaki iteraciji se tako temperatura zmanjša za faktor

$$\lambda = 10^{-6/T}. \qquad (13)$$

Ne glede na to, ali algoritem sprejme boljšo ali slabšo razporeditev žarnic, se v tem koraku oceni, če ima nova razporeditev boljšo oceno od najboljše najdene razporeditve. Če jo ima, razporeditev žarnic postane nova najboljša razporeditev. Algoritem se ustavi po T korakih izvajanja.

Algoritem 3.7 Simulirano ohlajanje

```
function SimulatedAnnealing( T : integer) : Hypothesis;
var
H : Hypothesis;                 (* Trenutna razporeditev *)
H' : Hypothesis;                (* Naslednik trenutne razporeditve *)
Hbest : Hypothesis;             (* Najboljša najdena razporeditev *)
t : real;                       (* Temperatura sistema *)
begin
  H := generate_random_hypothesis();
  Hbest := H;
  λ := 10^(-6/T) ;
  t := 0,1;
  while (T > 0) do
  begin
    Bulb := random(Sign.Bulbs);
    if Bulb ∈ H then H' := H / Bulb else H' = H ∪ Bulb;
    if q(H') > q(H) then H := H'; else
    begin
      d := e^((q(H')-q(H))/t) ;
      if random() < d then H := H';
    end
    if q(H') > q(Hbest) then Hbest := H';
    t := t * λ;
    T := T – 1;
  end
  return(Hbest);
end
```

3.8 Naključno preiskovanje

Naključno preiskovanje generira popolnoma naključne hipoteze. Preiskovanje ni na noben način usmerjano. Služi kot referenčni indikator kvalitete posameznega stohastičnega

preiskovanja prostora – če so ocene hipotez, ki jih generira naključno preiskovanje, blizu ocen hipotez usmerjenega preiskovanja, potem metoda ni zadovoljivo učinkovita.

Implementacija algoritma

V nadaljevanju je opisana naša implementacija algoritma naključnega preiskovanja. Postopek je prikazan kot Algoritem 3.8.

Funkcije:

- *generate_random_hypothesis():* vrne naključno razporeditev žarnic.

Algoritem generira naključne razporeditve žarnic in v vsakem koraku oceni, ali je naključna razporeditev boljša od do tega trenutka najbolje ocenjene razporeditve žarnic. Če je, ta postane nova najboljša odkrita razporeditev žarnic.

Algoritem 3.8 Naključno preiskovanje

```
function RandomSearch() : Hypothesis;
var
H_best : Hypothesis;          (* Najboljša odkrita razporeditev *)
H : Hypothesis;               (* Trenutna razporeditev *)
begin
  H_best := generate_random_hypothesis();
  repeat
    H := generate_random_hypothesis();
    if q(H) > q(H_best) then H_best := H;
  until stop_condition;
  return(H_best);
end
```

4 Empirično testiranje algoritmov

4.1 Eksperimentalno okolje

Testiranje učinkovitosti algoritmov smo izvajali na osebnem računalniku. Strojna oprema računalnika je zajemala trojedrno procesorsko enoto AMD Phenom II X3 720 s frekvenco 2.8 GHz, 384KB prvo-nivojskega predpomnilnika, 1536 KB drugo-nivojskega predpomnilnika in 6144 KB tretje-nivojskega predpomnilnika ter 4 GB DDR2 pomnilnika (2x2 GB, 1066 Mhz). Nameščeni operacijski sistem je bil Windows 7, 64 bitni.

4.2 Parametri algoritmov

Algoritme smo poganjali za prikazovalnike velikosti 20 x 20 žarnic. Ciljna množica predstavljenih simbolov je v prikazana v prilogi A. Vsak algoritem smo pognali 10x in ga pustili teči natanko 20 minut. Po 20 minutah smo izvajanje prekinili. Edina izjema je bil algoritem simuliranega ohlajanja. Dolžina izvajanja tega algoritma je odvisna od vhodnega parametra začetne temperature. Ta je bil nastavljen tako, da bi se izvajanje na testnem okolju izteklo približno v 20 minutah. Dejanski čas izvajanja je bil odvisen tudi od poteka preiskovanja in algoritem je lahko končal s preiskovanjem pred iztekom dvajsetih minut. V kolikor je algoritem presegel mejni čas 20 minut, smo tudi njegovo izvajanje prekinili.
Kot ustavitveno vrednost hevristične ocene smo postavili vsem algoritmom enako na 0,95.

V tabeli 4.1 so prikazani vrednosti parametrov posameznih algoritmov.

Algoritem	Parameter	Vrednost
Genetski algoritmi	N	400
	E	0,1
	P_c	0,8
	P_m	0,02
Iskanje harmonije	N	400
	P_{accept}	0,999
	P_{pa}	0,001
Optimizacija z roji delcev	N	400
	w	1,0
	c_1	1,0
	c_2	1,0
Požrešno iskanje z naključnimi ponovitvami	/	
Iskanje s tabuji	N	25
Skrajna optimizacija	Thau	1,3
Simulirano ohlajanje	T	225000
Naključno preiskovanje	/	

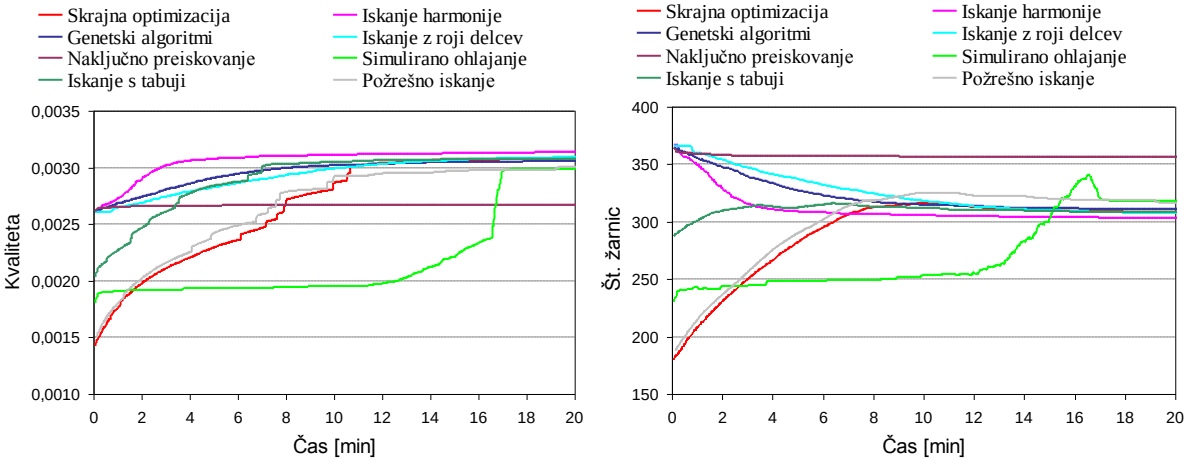
Tabela 4.1 *Parametri algoritmov*

5 Rezultati

Ob izvajanju algoritmov smo opazovali, kako se s časom izvajanja spreminja kvaliteta in število žarnic, uporabljenih v najboljši razporeditvi, ki jo algoritem najde do danega trenutka. Primerjava rezultatov vseh algoritmov je zbrana v tabeli 5.1, slika 5.1 pa prikazuje graf gibanja povprečne kvalitete razporeditev in povprečnega števila žarnic vseh testiranih algoritmov. Na slikah 5.2 – 5.9 so prikazani grafi za posamezne preiskovalne algoritme, kjer je s sivo barvo prikazano, kako se gibljejo kvaliteta in število žarnic v razporeditvah za vsako izmed desetih izvajanj vsakega algoritma. Z rdečo barvo je prikazano gibanje povprečnih vrednosti obeh metrik.

	Najmanj. št. žarnic	Največje št. žarnic	Povpr. št. žarnic	Najslabša kvaliteta	Najboljša kvaliteta	Povpr. kvaliteta
Genetski algoritmi	309	316	311	0,003007	0,003076	0,003055
Iskanje harmonije	**302**	**304**	**303**	**0,003126**	**0,003147**	**0,003134**
Optimizacija z roji delcev	305	310	307	0,003066	0,003115	0,003093
Požrešno iskanje z naključnimi ponovitvami	307	325	317	0,002924	0,003095	0,003001
Iskanje s tabuji	304	316	309	0,003007	0,003129	0,003080
Skrajna optimizacija	305	313	309	0,003037	0,003120	0,003081
Simulirano ohlajanje	311	326	318	0,002916	0,003055	0,002986
Naključno preiskovanje	353	359	356	0,002659	0,002692	0,002670

Tabela 5.1 *Primerjava najboljših, najslabših in povprečnih rezultatov vseh algoritmov*

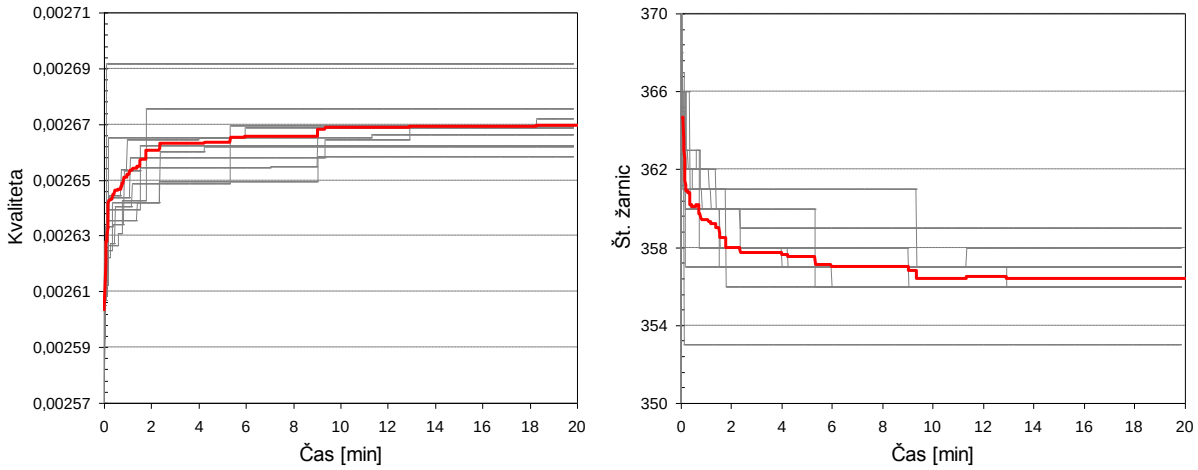


Slika 5.1 *Graf gibanja povprečne kvalitete razporeditev in povprečnega števila žarnic* ***vseh algoritmov***

Pri primerjavi povprečnih kvalitet preiskovanj (slika 5.1) najbolj izstopajo trije algoritmi: simulirano ohlajanje, naključno preiskovanje in iskanje harmonije. Algoritem naključnega preiskovanja izstopa zato, ker se na prvi pogled kvaliteta hipotez skoraj ne izboljšuje. A če se osredotočimo na grafa s slike 5.2, vidimo, da algoritem dejansko ves čas izboljšuje kvaliteto najdenih razporeditev, žal pa so te izboljšave povsem minimalne. Sicer je res, da s časom preiskovanja verjetnost, da naključno preiskovanje vendarle najde optimalno rešitev, narašča. Vendar bi v primeru, ko imamo na voljo dovolj časa, da bi čakali naključni algoritem, da

najde optimalno razporeditev, ta čas bolje izkoristili, če bi prostor raje preiskovali sistematično z generiranjem vseh možnih razporeditev žarnic.

Vendar pa kar nekaj algoritmov začne svoje preiskovanje s krepko slabšimi razporeditvami žarnic in porabi kar nekaj časa preiskovanja, da se kvaliteta razporeditev približa naključnemu preiskovanju. Pravzaprav se vsi algoritmi, ki preiskujejo prostor z izboljševanjem samo ene hipoteze, na začetku preiskovanja obnašajo slabše kot naključno preiskovanje. Iz tega bi lahko zaključili, da je naključno preiskovanje celo bolj smiselno uporabiti takrat, kadar smo zelo omejeni s časom, ki ga ima preiskovanje na voljo in iz takega ali drugačnega razloga ne moremo implementirati preiskovanj s populacijo hipotez (na primer zaradi omejenega spomina, ki je na voljo). Oziroma bolje, algoritme požrešnega iskanja, skrajne optimizacije in iskanja s tabuji bi bilo smiselno implementirati tako, da iskanja ne bi začeli s prvo naključno hipotezo, pač pa z najboljšo izmed večih naključnih hipotez. Tako bi povečali hitrost konvergiranja iskanj k najboljšim hipotezam.

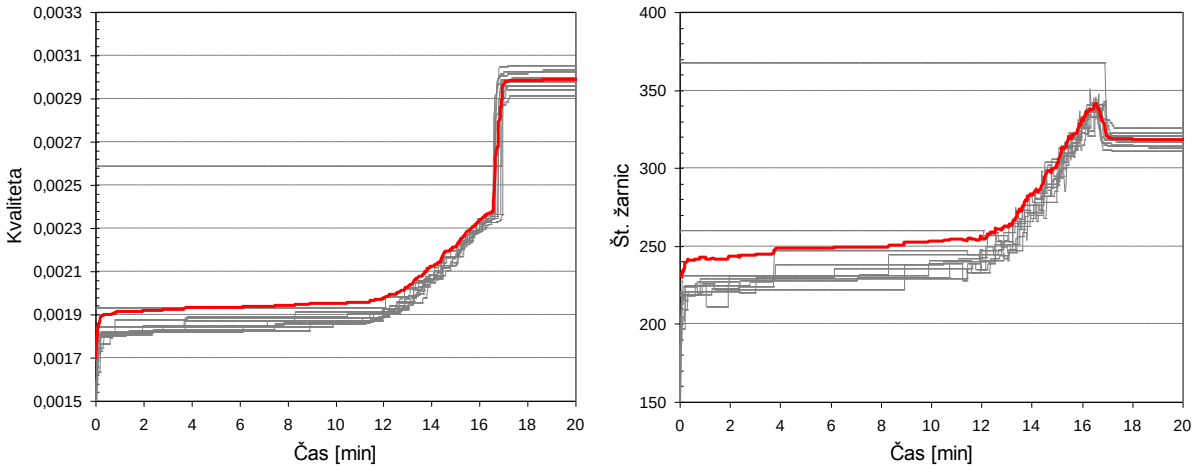


Slika 5.2 *Graf gibanja kvalitete razporeditev in števila žarnic ob izvajanju* ***naključnega preiskovanja***

Simulirano ohlajanje. Naslednji algoritem, ki na sliki 5.1 izstopa, je simulirano ohlajanje. Ta izstopa zato, ker več kot polovico časa (12 minut) algoritem preiskuje prostor precej naključno. Izboljšave v kvaliteti razporeditev so zato samo občasne, sama kvaliteta najdenih razporeditev pa precej nizka, kar se vidi tudi na sliki 5.3. Šele od dvanajste minute naprej se temperatura sistema zniža dovolj, da se iskanje usmeri k boljšim razporeditvam. Razlog sicer verjetno res tiči v slabo kalibriranem sistemu oz. previsoki začetni temperaturi sistema. Vendar učinkovito prikaže slabost nekaterih algoritmov: preobčutljivost na vrednost vhodnih parametrov. Ravno tej preobčutljivosti smo se želeli z določitvijo konstantne začetne in končne temperature sistema izogniti, vendar žal ne najbolj uspešno. Težko si predstavljamo, kako je lahko algoritem učinkovit v izvirni implementaciji, kjer je začetna temperatura odvisna izključno od vhodnega parametra. Višina temperature ne more uravnavati kvalitete preiskovanja, pač pa zgolj to, kdaj se bo prava optimizacija sploh začela. Vsekakor smatramo, da je v algoritem smiselno uvesti parameter, ki omogoča nastavljanje hitrosti ohlajanja.

Ker algoritem prične preiskovanje z naključno generirano razporeditvijo žarnic, nato pa to eno optimizira z lokalnimi spremembami, se v večini primerov zgodi, da v začetni razporeditvi ni prisotnih dovolj žarnic, da bi kvaliteta najslabše predstavljenega simbola presegla izbrano mejo ocene. Zato algoritem najprej najhitreje izboljšuje kvaliteto razporeditev predvsem z večanjem števila prisotnih žarnic. Opazen je skok kvalitete v približno sedemnajsti minuti.

Takrat iskanja presežejo mejno kvaliteto najslabše predstavljenega simbola, zato kvaliteta razporeditev poskoči, saj ocena postane utežena s številom dejansko uporabljenih žarnic. Za tem skokom se kvaliteta razporeditev izboljšuje predvsem z manjšanjem števila uporabljenih žarnic. Sistem se okoli osemnajste minute ohladi do te mere, da optimizacija postane skoraj deterministična, zato se iskanja ujamejo v lokalni optimum in kvaliteta razporeditev žarnic prične stagnirati.



Slika 5.3 *Graf gibanja kvalitete razporeditev in števila žarnic ob izvajanju* ***simuliranega ohlajanja***

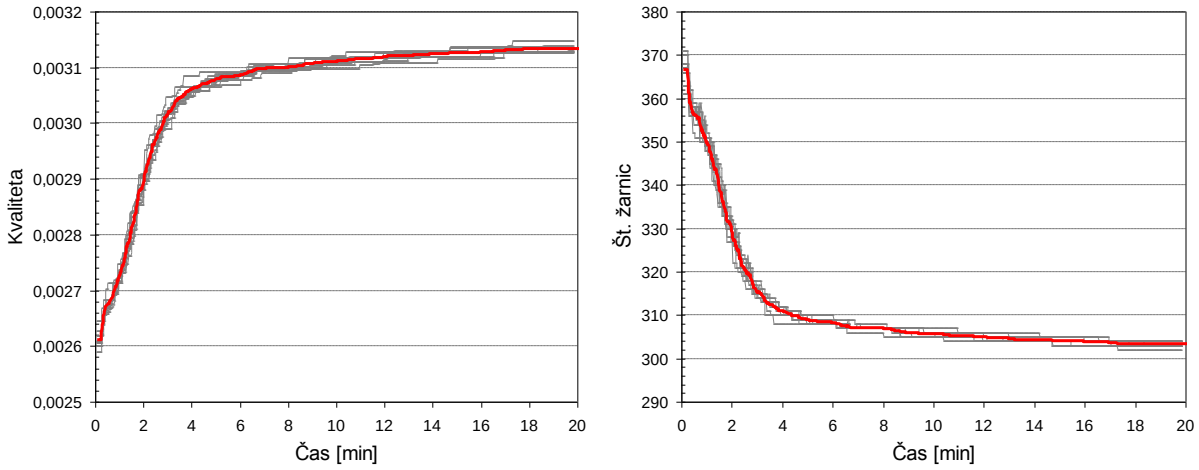
Glede na to, da iskanja prvo polovico časa preiskovanja izredno slabo izboljšujejo kvaliteto razporeditev žarnic, bi lahko sklepali, da implementacija algoritma začne s preveliko začetno temperaturo sistema. Če bi se ta temperatura znižala na nižjo vrednost, bi algoritem že prej začel usmerjeno izboljševati razporeditve in bi v takem stanju iskal več časa. Ker je po polovici iskanja temperatura enaka $0{,}1*(10^{-6/T})^{T/2} = 0{,}1*10^{-3} = 10^{-4}$, bi bilo verjetno bolj smotrno to vrednost uporabiti za začetno temperaturo sistema in s tem podaljšati čas usmerjenega preiskovanja.

Iskanje harmonije. Tretji izstopajoč algoritem je iskanje harmonije. Ta izstopa zato, ker se je neizpodbitno izkazal za najbolj učinkovitega, kar je razvidno tudi iz tabele 5.1. Kvaliteta razporeditev žarnic je že v začetku najboljša in zelo hitro zraste proti najbolj optimalnim vrednostim, sama rast kvalitete pa se ne ustavi do konca preiskovanja in ves čas ohranja prednost pred vsemi ostalimi algoritmi. Slika 5.4 kaže še drugo posebnost algoritma. Kvalitete preiskovanih najboljših razporeditev se ves čas iskanja pri vseh izvajanjih algoritma gibljejo skoraj identično.

Vsi algoritmi so si v eni stvari zelo podobni. V vsaki iteraciji izboljšajo del hipoteze in počasi se, del za delom, izboljšuje in izoblikuje končna hipoteza. Ker je v našem primeru ocena kvalitete razporeditve žarnic časovno precej zahtevna oz. potratna, je očitno, da ne glede na vrsto algoritma, iskanje največ časa porabi za ocenjevanje kvalitete hipotez. Učinkovitost algoritma iskanja harmonije je verjetno najbolj učinkovito zato, ker v vsakem koraku izvede zgolj eno ocenjevanje kvalitete hipoteze. To pomeni, da lahko v omejenem času izvede največje število korakov in lahko zato kar najbolje oz. največkrat izvede optimizacijo. Hkrati algoritem vsebuje tri ključne dejavnike za uspešno preiskovanje:

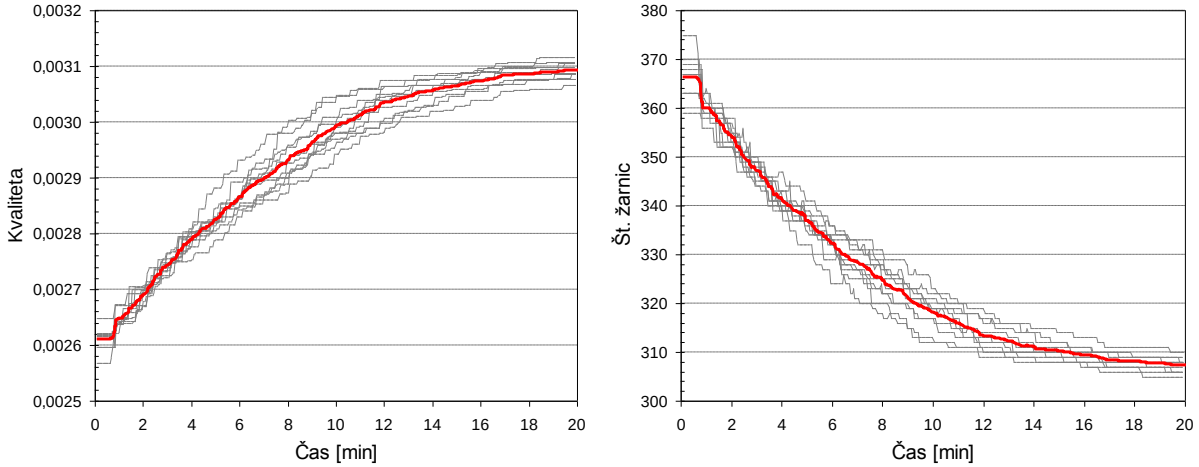
- prostor preiskuje z množico hipotez, torej lahko preišče večji del prostora,
- smer preiskovanja usmerja glede na kvaliteto hipotez v populaciji in tako koncentrira preiskovanje v bolj obetavnih delih prostora,

- ob vsakem generiranju nove hipoteze z malo verjetnostjo izbira dele hipoteze popolnoma naključno, s čimer ima vedno odprto možnost, da se iskanje preseli v še popolnoma neobiskan del prostora.



Slika 5.4 *Graf gibanja kvalitete razporeditev in števila žarnic ob izvajanju* ***iskanja harmonije***

Oba druga dva algoritma, ki preiskujeta s populacijo hipotez, v enem koraku preiskovanja spremenita in ponovno ocenita celotno populacijo hipotez. S tem pa izgubita na hitrosti izvajanja in posledično na hitrosti konvergiranja proti boljšim hipotezam, kar je razvidno tudi iz slike 5.5.



Slika 5.5 *Graf gibanja kvalitete razporeditev in števila žarnic ob izvajanju* ***optimizacije z roji delcev***

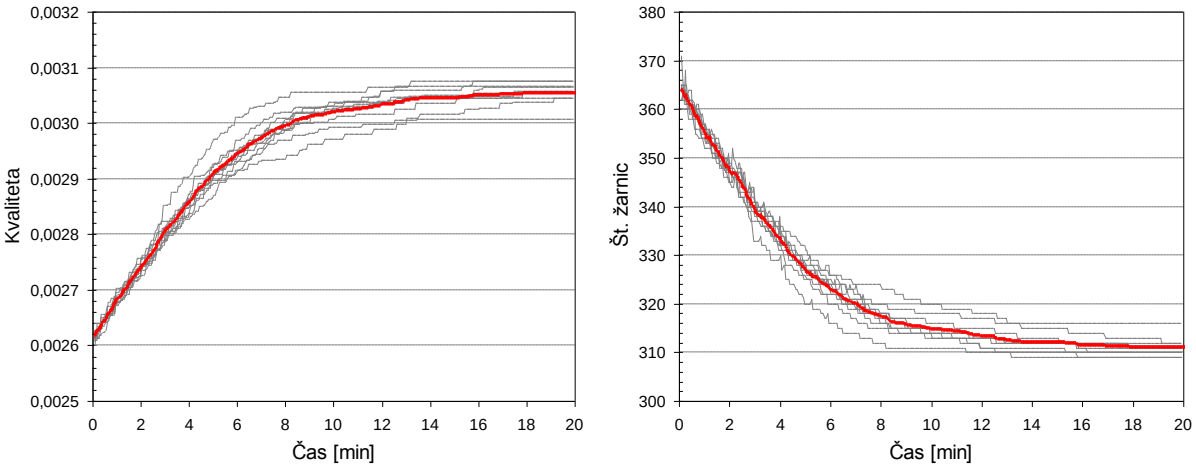
Algoritem preiskovanja z rojem delcev ima v primerjavi z iskanjem harmonije še dve drugi slabosti. Kot že omenjeno, vsi delci vedno pospešujejo zgolj proti globalno najboljšemu delcu, čeprav so morda drugi delci v drugih delih prostora odkrili samo malo slabše, a vseeno obetavne hipoteze. Algoritem bi zato bilo smiselno spremeniti v tej smeri, da bi delci pospeševali proti večim obetavnim delom prostora, ne le proti trenutno najbolj obetavnemu. Smer iskanja posameznega delca bi lahko usmerjali ne samo proti lastni in globalno najboljši točki v prostoru, pač pa proti kar vsem ostalim delcem ali njihovim najbolje ocenjenim hipotezam. Pri tem bi bilo tako spreminjanje smeri uteženo na primer s kvaliteto posameznega

delca. Tako bi bilo sodelovanje med delci v roju še povečano in bi se preiskovanje lahko osredotočilo proti več obetavnim delom prostora, ne le proti najboljšemu.

Druga slabost algoritma je pomanjkanje naključnosti. Na začetku je sicer res vsak delec postavljen na naključno mesto v prostoru in nato naključno hitro pospešuje proti odkritemu najbolj optimalnemu delu prostora, vendar se po daljšem času iskanja, ko se vsi delci zelo približajo odkritemu lokalnemu optimumu, vsi delci v ta optimum tudi ujamejo. Manjka namreč komponenta algoritma, ki bi vnašala nove, sveže dele hipotez. Morda bi lahko v trenutku, ko določen delec preleti najboljšo hipotezo ali pa se ji dovolj približa, ta delec zavrgli in ga nadomestili z novim, naključno generiranim. Tako preiskovanje nikoli ne bi obtičalo v odkritem optimumu, ampak bi se vedno še naprej razvijalo.

Kar zbode na sliki 5.5, je to, da tekom preiskovanja kvaliteta razporeditev žarnic počasi, a vztrajno raste in to tudi tik pred koncem preiskovanja. Sklepamo, da iskanja še niso obtičala v lokalnem optimumu in da bi se lahko kvaliteta še nadalje izboljšala, če bi bilo na voljo več časa.

Genetski algoritmi si razen te slabosti, da v vsaki iteraciji ocenijo kvaliteto celotne populacije in zato iščejo počasneje, delijo vse ostale pozitivne lastnosti z algoritmom iskanja harmonije: iskanje poteka v različnih delih prostora hkrati, boljše hipoteze imajo večjo verjetnost, da bodo izbrane za križanje, in z metodo mutacije se ves čas v populacijo vnašajo novi geni, torej novi deli hipotez. Pa vendar se v povprečju genetski algoritmi odrežejo skoraj najslabše od vseh implementiranih algoritmov. Tudi tokrat razlog za to verjetno tiči v naši implementaciji algoritma. V eni iteraciji namreč tudi najboljše hipoteze zgolj enkrat prispevajo svoj genski zapis v razmnoževanje in s tem v naslednjo generacijo. Morda bi bilo bolje, da bi se boljše razporeditve žarnic množile večkrat in imele več potomcev – z isto ali različnimi drugimi razporeditvami. Tako bi se obetavnejši del prostora preiskal bolj temeljito in bi zato preiskovanje konvergiralo hitreje kot na sliki 5.6, najdene razporeditve žarnic pa bi bile boljše kvalitete.

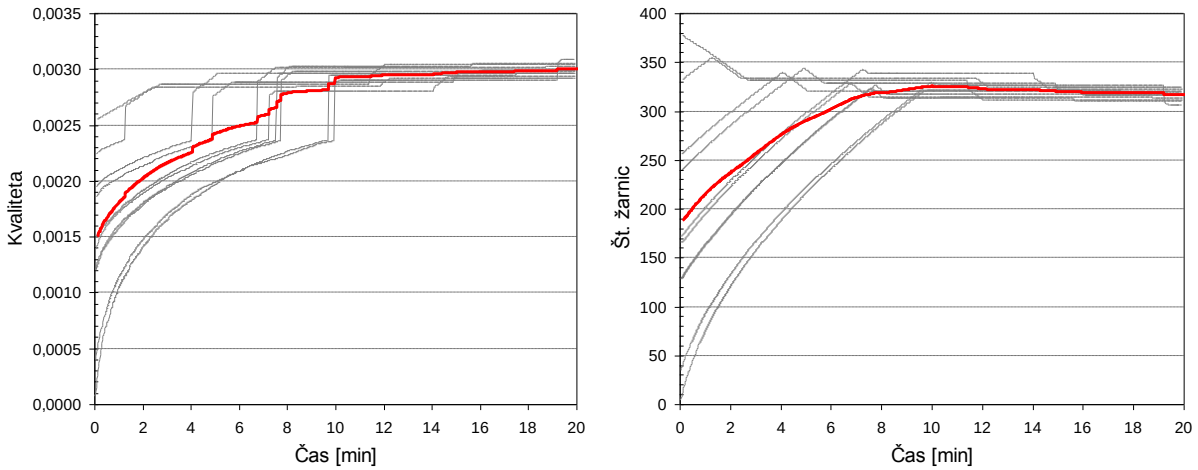


Slika 5.6 *Graf gibanja kvalitete razporeditev in števila žarnic ob izvajanju* ***genetskih algoritmov***

Preostali štirje algoritmi spadajo v povsem drugo kategorijo algoritmov. Vsi namreč izboljšujejo samo eno hipotezo. Kar pomeni, da se od iskanja harmonije razlikujejo po tem, da naenkrat preiskujejo samo en del prostora, tega pa običajno v smeri, ki lokalno izgleda optimalna, čeprav ni nujno, da je smer preiskovanja optimalna tudi globalno.

Požrešno iskanje in **iskanje s tabuji** sta tudi popolnoma deterministična algoritma. Edina stohastičnost je naključno generiranje začetne hipoteze. Preiskovanje se tako zelo lahko ujame v lokalni optimum, čemur se preiskovanje s tabuji sicer trudi izogniti. Na drugačen način, z naključnostjo pa se poskuša izogniti lokalnim optimumom skrajna optimizacija. Glede na to, da sta oba algoritma – skrajna optimizacija in iskanje s tabuji – precej primerljivo uspešna, čeprav uporabljata popolnoma različni tehniki izogibanja lokalnim optimumom, bi bilo zanimivo videti, kako bi se obnašal algoritem, ki bi bil kombinacija obeh. Skrajna optimizacija lahko zaradi naključnosti usmeri preiskovanje proti vsakemu delu prostora, vendar se smer preiskovanja lahko zelo hitro obrne, saj nima nobenega varovalnega mehanizma, ki bi preprečil, da se iskanje ne vrača k že preiskanim delom prostora. To lastnost ima iskanje s tabuji, kateremu pa manjka raznolikosti, ki jo prinese naključno izbiranje naslednika hipoteze.

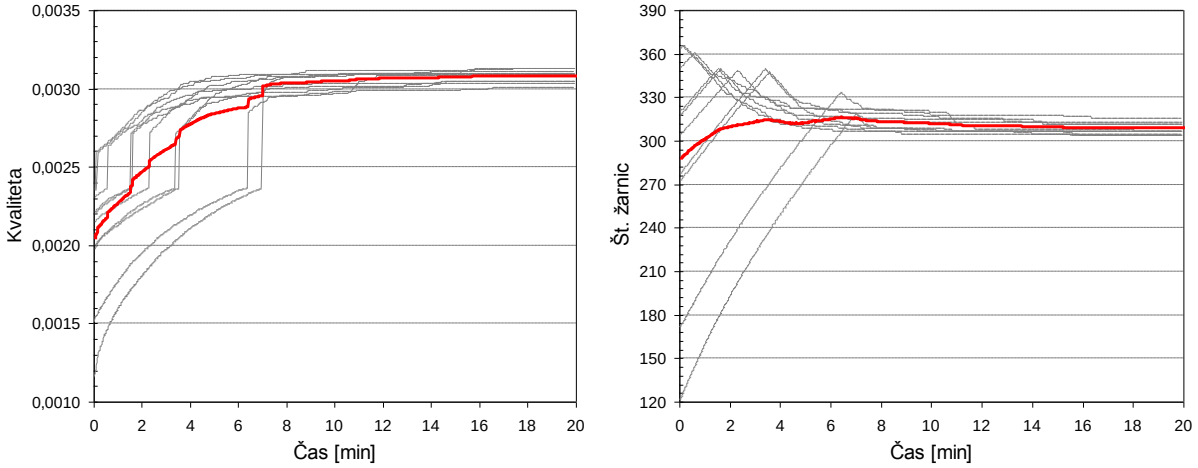
Na sliki 5.7 se dobro vidi lastnost algoritmov, ki eno hipotezo oblikujejo in izboljšujejo v boljšo: ob vsakem izvajanju algoritem prične z naključno hipotezo, ki je lahko bolj ali pa manj kvalitetna. Zato je razlika v hitrosti konvergiranja k nekemu optimumu med posameznimi izvajanji lahko zelo velika. V najboljšem primeru se iskanje že po treh minutah iskanja stabilizira ob precej dobri razporeditvi žarnic in od tam naprej le še malenkostno izboljšuje kvaliteto, v drugem primeru pa v tako stanje preide šele po desetih minutah iskanja – torej dvakrat kasneje.



Slika 5.7 *Graf gibanja kvalitete razporeditev in števila žarnic ob izvajanju* ***požrešnega iskanja z naključnimi ponovitvami***

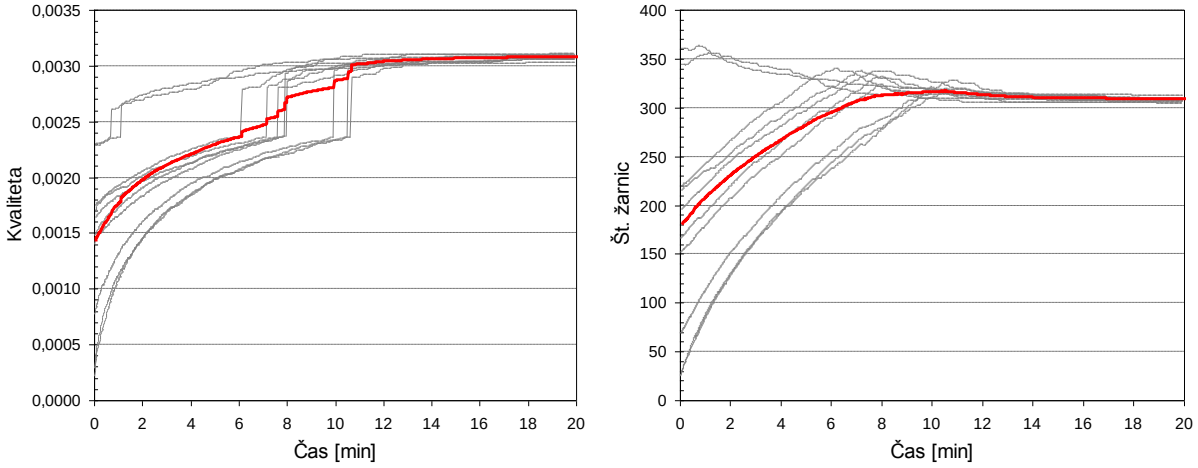
Opazimo tudi, da kvaliteta najdenih razporeditev žarnic raste nekako do desete minute, ko krivulja kvalitete naredi večji skok. Za tem se kvaliteta spremeni le še z nekaj malimi skoki med štirinajsto in devetnajsto minuto. Sklepamo, da algoritem za eno izvajanje ponovitve iskanja potrebuje približno deset minut. Torej se je v dvajsetih minutah, kolikor so imela izvajanja na voljo, zgodila samo po ena ponovitev preiskovanja. V splošnem naj bi požrešno iskanje z naključnimi ponovitvami našlo precej dobre hipoteze, vendar pa se bi morale ponovitve izvršiti večkrat, ne le dvakrat.

Če primerjamo sliki 5.7 in 5.8 opazimo, da se iskanje s tabuji v začetku preiskovanj obnaša precej podobno, kot požrešno iskanje. To ni nič presenetljivega, saj je osnova obeh algoritmov enaka. Vendar se izboljševanje kvalitete hipotez ne ustavi skoraj do konca preiskovanja. Seznam tabu torej uspešno opravlja svojo nalogo preprečevanja, da bi se iskanje ustavilo v lokalnem optimumu, kar se sicer zgodi pri požrešnem iskanju.



Slika 5.8 *Graf gibanja kvalitete razporeditev in števila žarnic ob izvajanju* ***iskanja s tabuji***

Na grafu gibanja števila žarnic v razporeditvah na sliki 5.8 so zelo izstopajoči vrhovi krivulj. Sklepamo, da algoritem na začetku preiskovanja zgolj povečuje število žarnic v razporeditvah vse do trenutka, ko lahko z razporeditvijo žarnic vse simbole prikaže s kvaliteto, ki presega izbrano mejo. Do tega trenutka niti ne poskuša izboljšati kvalitete s prerazporejanjem enakega števila žarnic. V trenutku, ko graf doseže vrh, algoritem začne število žarnic zmanjševati, saj, zaradi take zasnove hevristične ocene, na ta način najhitreje izboljšuje kvaliteto razporeditev. Glede na hitrost zmanjševanja žarnic takoj po dosegu vrha, sklepamo, da se v razporeditvah ves čas nahajajo povsem odvečne žarnice, ki jih lahko brez prerazporejanja algoritem odstrani, vendar tega ne naredi, dokler ocena kakovosti ne začne v oceni upoštevati števila žarnic.

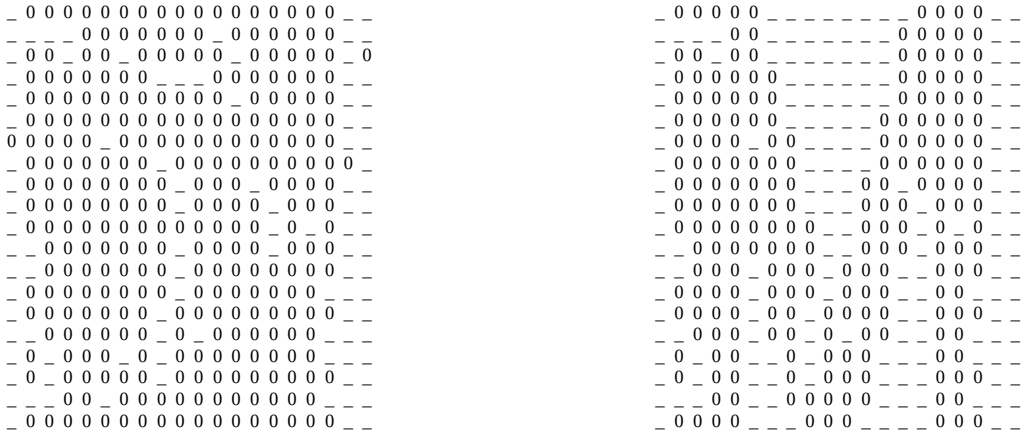


Slika 5.9 *Graf gibanja kvalitete razporeditev in števila žarnic ob izvajanju* ***skrajne optimizacije***

Tudi gibanje kakovosti razporeditev žarnic skrajne optimizacije se giblje podobno, kot pri požrešnem iskanju. Med slikama 5.7 in 5.9 je opazna razlika v vrhu grafa števila uporabljenih žarnic. Ta se od ostalih algoritmov, ki temeljijo na požrešnem iskanju, razlikuje, saj se število žarnic ne prične strmo zmanjševati v hipu, ko doseže vrh, pač pa je greben daljši. Ustavljanje povečevanja števila žarnic je torej bolj postopno. Razlog za ta pojav tiči v verjetnosti, s katero algoritem sprejme za naslednika najboljšo hipotezo. Ta verjetnost ni enaka 1, pač pa je precej manjša, možnost za izbor pa imajo tudi slabši nasledniki. Z istim razlogom se iskanje tudi ne

ustavi v lokalnem optimumu. Izboljševanja kvalitete hipotez je opaziti tudi še čisto pred koncem preiskovanja.

Za konec si oglejmo še razporeditev žarnic, ki jo algoritem iskanja harmonije označi za najboljšo izmed vseh razporeditev. Na sliki 5.10 je prikazana razporeditev vseh žarnic na prikazovalniku in prikaz črke M, ki je pri dani razporeditvi žarnic najslabše prikazana. Funkcija kvalitete prikaza simbola ovrednoti kvaliteto prikaza črke M z 0,9504, kar je za malenkost nad izbrano ustavitveno mejo ocene kakovosti.



Slika 5.10 *Najboljša odkrita razporeditev žarnic (levo) in prikaz najslabše predstavljenega simbola – črke M (desno)*

Črka M je na prikazovalniku vsekakor razpoznavna in verjetno v primeru, da bi bila dejansko prikazana na svetlobnem prikazovalniku, niti ne bi opazili, da nekaj njenih točk ni osvetljenih.

6 Zaključek

V diplomski nalogi smo se lotili problema optimizacije strukture svetlobnih prikazovalnikov. Pri tem smo predvsem želeli poceniti izdelavo takih prikazovalnikov z zmanjšanjem števila žarnic, ki jih sestavljajo. Hkrati pa nismo želeli bistveno okrniti kvalitete simbolov, ki so jih prikazovalniki zmožni prikazati.

Pri iskanju rešitve problema smo uporabili naslednje optimizacijske tehnike: genetske algoritme, iskanje harmonije, optimizacijo z roji delcev, požrešno iskanje z naključnimi ponovitvami, iskanje s tabuji, skrajno optimizacijo in simulirano ohlajanje. Kot primer neinformiranega preiskovanja smo implementirali tudi naključno preiskovanje, s katerim smo želeli ponazoriti, da se vsako od informiranih preiskovanj obnese bolje, kot neinformirano.

Ugotovili smo, da se vsi implementirani algoritmi bolj ali manj uspešno kosajo z izpostavljenim problemom. Algoritmi, ki prostor preiskujejo s populacijo hipotez, v povprečju sicer hitreje konvergirajo k boljšim rešitvam, a kljub temu kvalitete njihovih končnih rešitev ne odstopajo bistveno od kvalitet rešitev preostalih algoritmov. Rezultati nakazujejo, da je pri problemih, kjer je ocenjevanje kvalitete hipoteze zahtevna operacija, daleč najbolj pomembna lastnost algoritma število ocenjevanj, ki jih potrebuje izvesti za napredovanje k boljšim hipotezam. Izmed osmih primerjanih algoritmov se je najbolje izkazal algoritem iskanja harmonije, ki v vsakem koraku napredovanja k boljšim hipotezam oceni zgolj eno novo hipotezo.

V okviru eksperimentalnih rezultatov smo število žarnic na prikazovalniku velikosti 20 x 20 žarnic uspeli zmanjšati s 400 na 302. Torej smo ceno izdelave prikazovalnika zmanjšali za ceno 24,5% žarnic, pri tem pa se kvaliteta predstavljenih simbolov ni bistveno zmanjšala. Sicer celotni proizvodni stroški takega prikazovalnika gotovo ne zajemajo zgolj stroškov žarnic, vendar zmanjšanje v takem obsegu zagotovo ni zanemarljivo. S takim rezultatom bi bil marsikateri proizvajalec zadovoljen, z rezultati pa smo zadovoljni tudi sami.

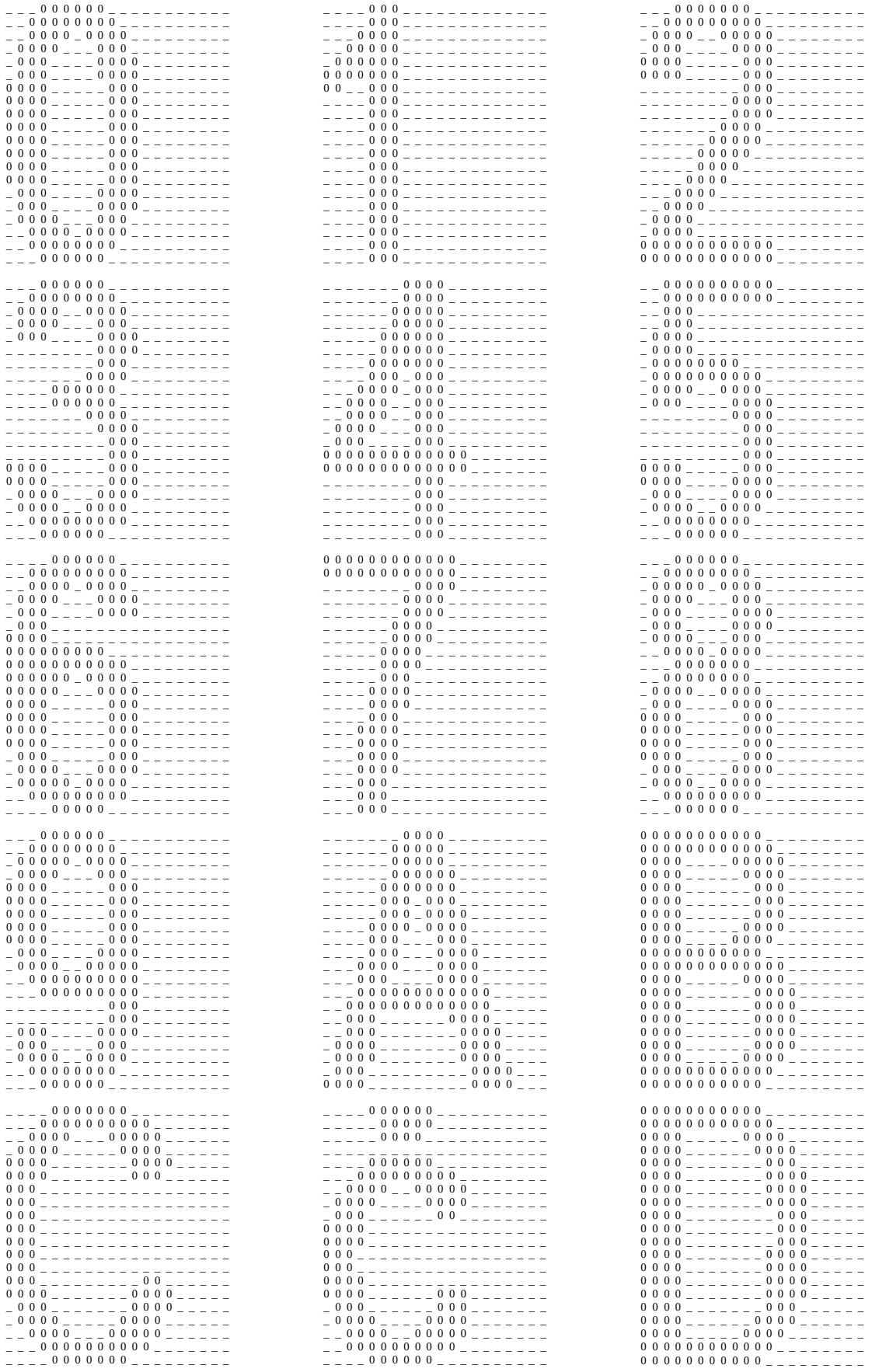
Ideje za nadaljnje delo na tem področju vključujejo:

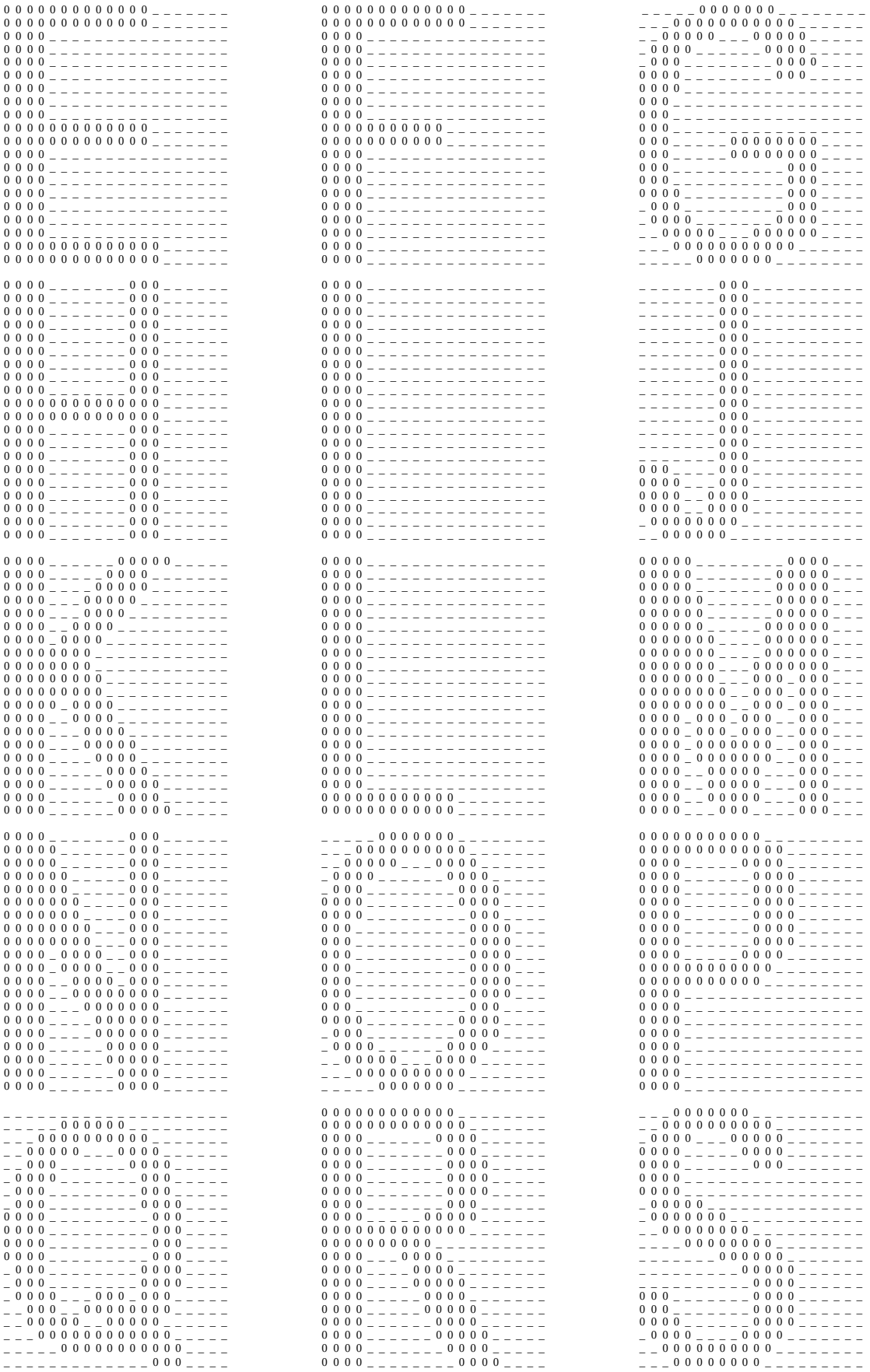
- opustitev omejitve, da se žarnice na prikazovalniku nahajajo zgolj v vodoravnih in navpičnih vrstah. Žarnice, ki bi se lahko prosto postavljale, bi lahko bolje hkrati prikazovale različne simbole, saj bi lahko bolje zapolnile vrzeli, ki nastanejo v prikazu simbolov zaradi zmanjšanega števila žarnic,
- raziskati, ali se da na podoben način optimizirati tudi barvne prikazovalnike, kjer bi se lahko manjkajoče žarnice posamezne barve do določene meje nadomestilo z bližnjimi žarnicami drugih barv,
- na podoben način bi lahko optimizirali tudi vzdržljivost prikazovalnikov – ugotoviti bi bilo potrebno, katere žarnice so za kvaliteto prikazanih simbolov najbolj odgovorne in na njihovih mestih uporabiti bolj vzdržljive žarnice.

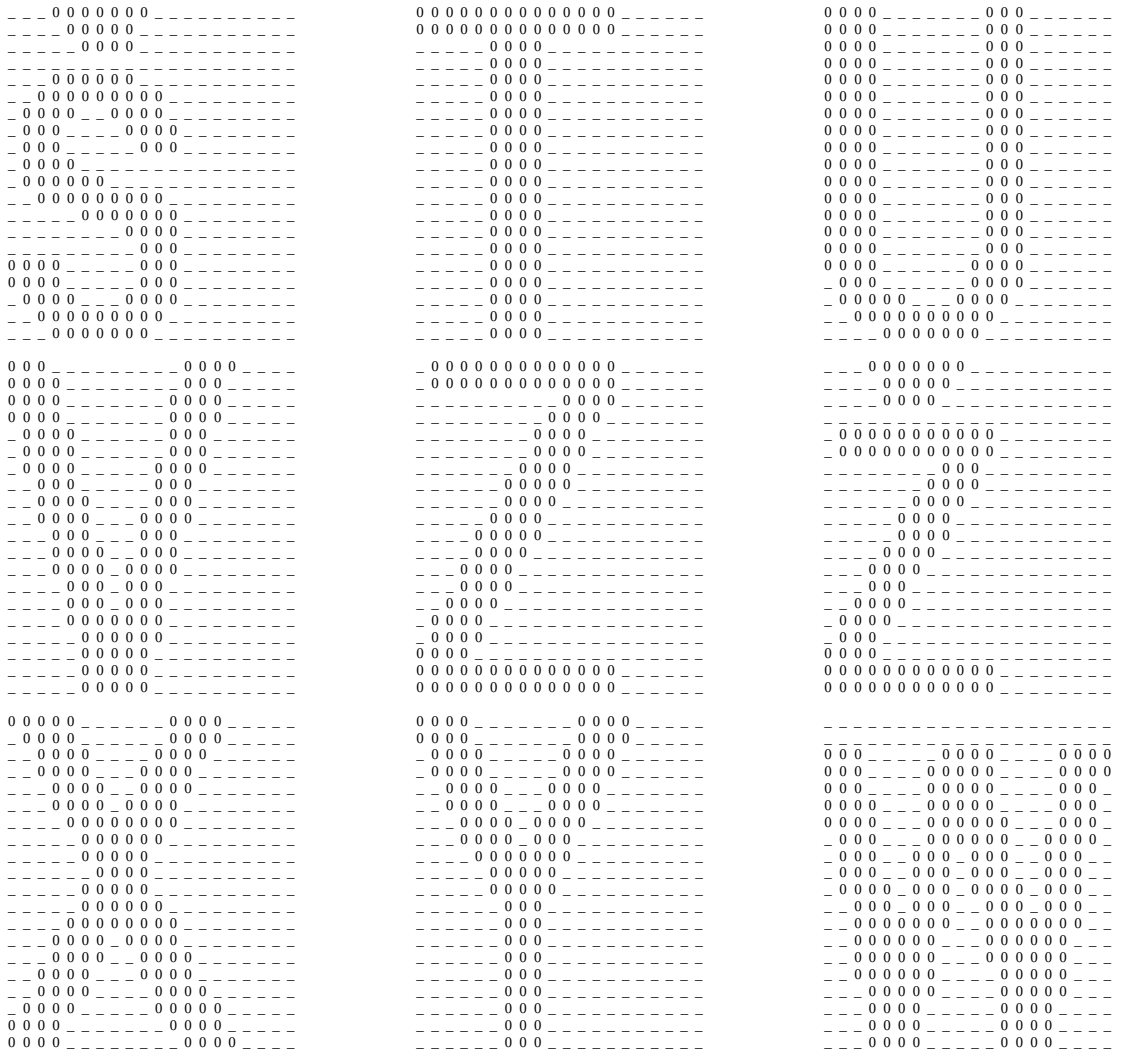
Slike

Tabele

Priloga A Prikaz prikazovalnikov z izpisanimi vsemi 39 simboli







Literatura

[1] B. T. Abreu, E. Martins, F. L. de Suosa, Generalized Extremal Optimization Applied to Path Testing, *Supplementary proc. of the 17th IEEE Int. Symp. on Software Reliability Engineering*, št. 17, 2006.

[2] S. Boettcher, A. G. Percus, Extremal optimization for graph partitioning, *Phisical Review E*, št. 64 026114, str. 1 – 13, 2001.

[3] S. Boettcher, A. G. Percus, Optimization with Extremal Dynamics, *Complexity*, št. 8, zvezek 2, str. 57 – 62, 2003.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Dijkstra's algorithm, *Introduction to Algorithms*, MIT Press, str. 595 – 601, 2001.

[5] J. Dreo, A. Petrowski, P. Siarry, E. Taillard, *Metaheuristics for Hard Optimization*, Springer, Berlin, 2006.

[6] R. C. Eberhart, Y. Shi, Particle Swarm Optimization: Developments, Applications and resources, *Proceedings of IEEE Congress on Evolutionary Computation 2001*, IEEE service center, Piscataway, NJ, Seoul, Korea, str. 81 – 86, 2001.

[7] Z. W. Geem, J. H. Kim, G. V. Loganathan, A new heuristic optimization algorithm: Harmony search, *Simulation*, št. 76, str. 60 – 68, 2001.

[8] F. Glover, M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Boston, 1997.

[9] J. Kennedy, R. C. Eberhart, A Discrete Binary Version of the Particle Swarm Algorithm, *IEEE International Conference on Systems, Man and Cybernetics*, št. 5, str. 4104 – 4108, 1997.

[10] I. Kononenko, *Strojno učenje*, Fakulteta za računalništvo in informatiko, Ljubljana, 2005.

[11] D. L. Mann, D. R. Highfill, R. B. Day, *Investigative laboratory on "filling in" by the brain of the blindspot of the visual field*, str. 391 – 397, 2004.

[12] M. Shermer, Patternicity: Noun. The tendency to find meaningful patterns in meaningless noise, S*cientific American*, december 2008, str. 48.

[13] X.-S. Yang, Harmony Search as a Metaheuristic Algorithm, v zborniku Music-Inspired Harmony Search Algorithm: Theory and Applications, *Studies in Computational Intelligence*, Springer, Berlin, št. 191, str. 1 – 14, 2009.