SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5189

Optimizacija funkcije evolucijskim algoritmom uz analizu strukture rješenja

Nikola Zadravec

Zagreb, lipanj 2017.

SADRŽAJ

1. Uvod

Problem optimizacije sveprisutan je u području računarske znanosti kao i u mnogim drugim područjima. U svakodnevnom životu se uobičajeno pojavljuju teški, ručno neriješivi kombinatorni problemi čije je uspješno rješavanje od vrlo velike važnosti za bolje funkcioniranje društva. Neki od takvih problema su: problem pakiranja, planiranje, izrada rasporeda sati, itd. Iako su današnji optimizacijski algoritmi dovoljno uspješni za niz takvih problema, s obzirom na njihovu sveprisutnost te veliku ekonomsku važnost njihova što učinkovitijeg rješavanja, optimizacijski algoritmi su i dalje područje intenzivnog istraživanja. Kod ovakvih problema dimenzionalnost domene pri tome predstavlja ograničavajući faktor.

Jedan od pristupa koji bi kod određenih vrsta problema mogao rezultirati povećanjem učinkovitosti pronalaska kvalitetnih rješenja jest dinamičko smanjenje dimenzionalnosti problema na način da se pojedine komponente domene grupiraju. Primjer takvog pristupa opisan je u radu "Transforming Evolutionary Search into Higher-Level Evolutionary Search by Capturing Problem Structure", IEEE Trans. on Evolutionary Computation, October 2014, Vol 18, No 5.

U okviru ovog rada proučit će se navedeni rad te će se ostvariti programska implementacija predloženog pristupa, provest će se vrednovanje i usporediti rezulatati s onima objavljenim u navedenom radu. Ostatak rada je organiziran na sljedeći način. Nakon uvoda slijedi pregled područja u kojem opisujemo problem optimizacije, nabrajamo metode provođenja postupka optimizacije i definiramo alat za vrednovanje efikasnosti algoritama. Nakon toga slijedi poglavlje u kojem se ukratko govori o algoritmima temeljenim na izgradnji modela. Definicija našeg algoritma je dana u petom poglavlju te u šetom se apriori analizira implementacija algoritma i daji rezultati eksperimenata. Sedmo poglavlje donosi zaključak.

2. Pregled područja

2.1. Problem optimizacije

U matematici i računarskoj znanosti, problem optimizacije je problem pronalaženja najboljeg rješenja od svih mogućih rješenja. Karakteristike optimizacijskih problema su:

- prostor evaluacijske funkcije,
- prostor rješenja te
- ograničenja.

Prostor evaluacijske funkcije određuje prirodu optimizacijskog problema. Optimizacijske probleme se dijele na probleme jednokriterijske optimizacije i probleme višekriterijske optimizacije. Razlika između ova dva skupa problema je ta što je kod jednokriterijske optimizacije evaluacijska funkcija skalar, a kod višekriterijskih je vektor (postoji više kriterija po kojima se uspoređuju rješenja).[1] Višekriterijski problemi se mogu svesti na jednokriterijske tako da se više kriterija iskombinira u jedan kriterij pomoću parametara važnosti pridjeljujući pojedinim kriterijima iz višekriterijske optimizacije različite vrijednosti ovisno o bitnosti kriterija. Dakako, važnijim kriterijima se pridjeljuje veća vrijednost parametra.

Optimizacijski problemi mogu se podijeliti u dvije kategorije s obzirom na prostor rješenja, odnosno vrstu argumenta evaluacijske funkcije. Ta podjela odnosi se na vrijednost koju argument evaluacijske funkcije može poprimiti, a može biti element prebrojivog ili neprebrojivog skupa, odnosno element diskretne ili kontinuirane domene. U raznoj literaturi diskretan optimizacijski problem se još može naći i pod nazivom kombinatorički problem.

Ograničenja na moguća rješenja najčešće su ugrađena u samu evaluacijsku funkciju. Ograničenja s obzirom na strogoću prihvatljivosti rješenja se dijele na tvrda i

[1] Ovdje i u nastavku pod kriterijem se misli na evaluacijsku funkciju.

meka. Tvrda ograničenja u najužem smislu odjeljuju potprostor valjanih od potprostora nevaljanih u ukupnom prostoru svih rješenja, dok meka ukazuju na to koliko je rješenje zadovoljavajuće.

U većini slučajeva narav problema određuje tražimo li maksimum ili minimum evaluacijske funkcije. Evaluacijske funkcije koje je potrebno minimizirati, funkcije kazne (engl. *cost functions*), često se prevode u komplementarne funkcije koje je potrebno maksimizirati. U tom slučaju riječ je o funkcijama pogodnosti ili dobrote (engl. *fitness functions*) koje po defniciji daju to veću vrijednost što je rješenje bolje. Ako je zadana funkcija f koju je potrebno minimizirati, najjednostavniji način izgradnje pripadne funkcije dobrote jest uzeti funkciju $-f$.

Budući da se u ovom radu analizira algoritam za rješavanje diskretnih optimizacijskih problema, slijedi opis kombinatornih problema. Primjerak problema kombinatorne optimizacije je uređeni par (S, f), gdje je S prebrojiv skup rješenja, a $f\colon S \to \mathbb{R}$ evaluacijska funkcija koja svakom rješenju $s \in S$ pridružuje procjenu kvalitete rješenja. Problem kombinatorne optimizacije je skup primjeraka problema kombinatorne optimizacije. Svim primjercima istog problema najčešće su zajednički način prikaza problema te funkcija evaluacije, a različitost se očituje u prostoru rješenja.

2.2. Točni algoritmi, heuristike, metaheuristike

Algoritmi za rješavanje problema kombinatorne optimizacije dijele se u dvije skupine: točni i približni algoritmi. Točnim algoritmima je osiguran pronalazak optimuma problema zbog načina na koji je zasnovana metoda pretraživanja prostora rješenja, a koja odgovara sistematičnom ispitivanju rješenja i uzimanjem najboljeg među njima. Zbog takvog načina pretraživanja točni algoritmi nisu značajni za praktične primjene jer je u realnim problemima dimenzija prostora rješenja prevelika za ispitivanje cijelog prostora rješenja što je motiviralo uvođenje približnih algoritama.

Približni algoritmi, poznatiji pod nazivom heuristike, ne nalaze nužno optimum već se osnovna ideja njihovog rada zasniva na pronalasku dovoljno kvalitetnih rješenja uz relativno nisku računalnu složenost. Srećom pronalazak optimalnog rješenja nije od presudne važnosti u praksi, pa nije potrebno zadovoljiti i svojstvo optimalnosti algoritma nije potrebno zadovoljiti. Prema načinu dolaska do konačnog rješenja heuristike se dijele na konstruktivne algoritme i algoritme lokalne pretrage.

Konstruktivni algoritmi rješenje problema grade dio po dio sve do završetka konstrukcije čitavog rješenja. Jedan od najpoznatijih primjera koji se najčešće navodi prilikom obrade konstruktivnih algoritama je algoritam najbližeg susjeda.

Algoritmi lokalne pretrage rješenje problema započinju od nekog slučajno generiranog rješenja koje se nakon toga postupno poboljšava. Glavna ideja ostvarivanja procesa kojeg provodi algoritam lokalne pretrage jest da se nad trenutnim rješenjem rade pretvorbe kojima dobivamo susjedna rješenja.

Bitno je još istaknuti skup algoritama čiji je nastanak motiviran iz potrebe za generalizacijom heuristika s ciljem ostvarivanja algoritama čija primjena pokriva velik skup problema, a riječ je o metaheuristikama. Metaheuristika je skup skica algoritama koje sadrže apstraktne i općenite strategije pretraživanja prostora rješenja. Smisao tih algoritama je u tome da se uz vrlo male preinake lako prilagode sprecifičnom problemu.

Primjeri metaheuristika su simulirano kaljenje, tabu pretraživanje, algoritmi evolucijskog računanja itd. Postoje dvije velike porodice algoritama: algoritmi koji rade nad jednim rješenjem (algoritam simuliranog kaljenja te algoritam tabu pretrage) te na populacijske algoritme koji rade sa skupom rješenja (algoritmi evolucijskog računanja).

2.3. Evolucijski algoritmi

Evolucijski algoritmi (engl. *evolutionary algoritms*) su postupci optimiranja, učenja te modeliranja koji se temelje na mehanizmu prirodne evolucije. To su formalni sustavi koji nastoje biti izomorfni s prirodnom evolucijom. Evolucijski algoritmi nastali su iz dviju razloga: želje za boljim razumijevanjem prirodne evolucije i pokušaja primjene načela prirodne evolucije pri rješavanju različitih problema (Grundler, 2001).

Zajednička svojstva evolucijskih algoritama su (Dalbelo-Bašić, 2004):

- algoritmi su zasnovani na populaciji rješenja,
- jedinke su međusobno usporedive prema dobroti,
- populacija jedinki se s vremenom mijenja, evoluira jer se provodi postupak selekcije jedinki,
- svojstva jedinki prenose se s roditelja na djecu te
- prostor rješenja se pretražuje osim usmjerenim pretraživanjem i slučajnim procesom.

Algoritmi su zasnovani na populaciji rješenja: najčešće se algoritam inicijalizira tako da se slučajno generira početna populacija rješenja. U početnu populaciju se mogu usaditi i rješenja dobivena nekim drugim algoritmom kako bi se ubrzala konvergencija algoritma. Tijekom evolucijskog procesa broj jedinki u populaciji (veličina populacije) se najčeše ne mijenja.

Jedinke su međusobno usporedive prema dobroti: jedinka predstavlja moguće rješenje problema. Primjerice, ako se traži maksimum funkcije $f(x)$, gdje je $x \in [dg, gg]$ tada jedinku predstavlja bilo koji broj između donje (dg) i gornje granice (gg) i za svaka dva broja $x1, x2 \in [dg, gg]$ može se odrediti je li $f(x1)$ manji, veći ili jednak $f(x2)$. Treba napomenuti da ovo svojstvo imaju algoritmi primijenjeni na probleme jednokriterijske optimizacije; kod višekriterijskih optimizacijskih problema ovo svojstvo općenito ne vrijedi.

Populacija jedinki se s vremenom mijenja, evoluira jer se provodi postupak selekcije jedinki: u procesu selekcije bolje jedinke imaju veću vjerojatnost preživljavanja od onih lošijih.

Svojstva jedinki prenose se s roditelja na djecu: U stvaranju nove jedinke (novog rješenja) sudjeluju izabrane (u pravilu bolje) jedinke iz prošle populacije. Drugim riječima, nad jedinkama djeluju operatori u stvaranju nove populacije. Prema tome, prostor rješenja se usmjereno pretražuje na temelju već postignutih rješenja kao primjerice u slučaju operatora križanja kod genetskih algoritama.

Prostor rješenja se pretražuje osim usmjerenim pretraživanjem i slučajnim procesom: evolucijski proces je u velikoj mjeri stohastičke prirode. Jedinke se slučajno odabiru, samo neke s većom, a neke s manjom vjerojatnošću. Tako i promjene koje unose operatori mogu biti slučajne.

2.4. Složenost algoritama

Razvojem algoritama pojavila se potreba njihovog međusobnog razlikovanja s obzirom na efikasnost, odnosno brzinu (broj operacija) te potrebnog prostora za izvođenje algoritma koja ovisi o količini ulaznih podataka, radi prebiranja najefikasnijeg među njima za rješavanje problema.

Prije same analize vremenske složenosti našeg algoritma potrebno je pomoću nekoliko definicija uvesti alat za provođenje te analize.

Definicija 1 (O notacija)**.** Kažemo da je funkcija f ograničena funkcijom g ako postoji konstanta C takva da za sve dovoljno velike n vrijedi

$$f(n) \leqslant C \cdot g(n).$$

Pišemo $f(n) = O(g(n))$.

Definicija 2 (Ω i Θ notacija)**.** Ako postoji konstanta $B > 0$ takva da za sve dovoljno velike n vrijedi

$$f(n) \geqslant B \cdot g(n)$$

tada ćemo pisati $f(n) = \Omega(g(n))$. Ako istovremeno vrijedi $f(n) = O(g(n))$ i $f(n) = \Omega(g(n))$, tj., ako postoje konstante B i C takve da za sve dovoljno velike n vrijedi

$$B \cdot g(n) \leqslant f(n) \leqslant C \cdot g(n)$$

tada ćemo pisati $f(n) = \Theta(g(n))$.

Definicija 3 (Asimptotski ekvivalentne funkcije)**.** Ako vrijedi

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1,$$

tada kažemo da su f i g asimptotski jednake kad $n \to \infty$ i pišemo

$$f \sim g \qquad \text{kad } n \to \infty.$$

Definicija 4 (o i ω notacija)**.** Ako za svaku pozitivnu konstantu $c > 0$ postoji $n_0 > 0$ takav da vrijedi

$$f(n) < cg(n)$$

onda kažemo da funckija g raste brže nego funkcija f i pišemo $f(n) = o(g(n))$. Zapis ekvivalentan ovome je $g(n) = \omega(f(n))$.

Proizvoljne apstraktne funkcije f iz prethodnih definicija u kontekstu ovog potpoglavlja dobivaju ulogu opisa vremenske složenosti. Time je uspješno ostvaren matematički aparat za analizu vremenske složenosti algoritma koji se svodi na određivanje funkcije "omeđenosti" dane funkcije f kojom je predstavljeno vrijeme (broj operacija) potrebno za izvođenje algoritma u ovisnosti o ulaznom broju podataka.

3. Algoritmi izgradnje modela

Modeliranje je važan koncept i metodologija u raznim disciplinama. Danas, u znanstvenom kontekstu pojam se odnosi na pojednostavljeni, apstraktni ili dobro strukturirani dio stvarnosti koji se proučava.

Model se smatra matematičkim modelom postupka ili problema ako sadrži tipične matematičke objekte (varijable, izraze, relacije). Matematički model predstavlja stvarni problem zapisan jezikom matematike.

Pri samoj izgradnji modela jako je važno definirati i precizno iskazati njegovu svrhu. U inženjerstvu, model se može koristiti za konstruiranje nekih strojeva. U istraživanju i optimizaciji, modeli se često koriste za podupiranje strateških ili operativnih odluka (Kallrath i Maindl, 2006). Modeli omogućavaju učenje i razumijevanje pojava čije efekte želimo znati predviđati. Predviđanje je korisno jer omogućava izbjegavanje provođenja dugotrajnih, skupih te možda čak i opasnih eksperimenata kako bismo saznali ponašanje i posljedice pojave.

Kako se ovaj rad temelji na optimizaciji, slijede dva razreda algoritama za optimizaciju temeljenih na izgradnji modela.

3.1. Algoritmi temeljeni na vjerojatnosnim razdiobama

Algoritmi temeljeni na vjerojatnosnim razdiobama (engl. *Estimation of distribution algorithms* (EDAs)) su stohastičke metode optimizacije koje usmjeravaju pretragu ka boljim rješenjima gradnjom i uzorkovanjem eksplicitnih vjerojatnosnih modela obećavajućih rješenja. [1] Optimizacija se očituje kao niz inkrementalnih ažuriranja vjerojatnosnog modela, počevši od modela koji kodira jednoliku razdiobu nad dopuštenim rješenjima i završava modelom koji generira samo globalni optimum (Pelikan et al., 2012). Algoritam 1 prikazuje pseudokôd algoritma temeljenog na vjerojatnosnim razdiobama. U idealnom bi se slučaju kvaliteta generiranih rješenja poboljašavala tijekom

[1] U nastavku ćemo za ove algoritme upotrebljavati kraticu EDA algoritmi.

vremena i nakon razumnog broja iteracija, izvršenje ovih tri koraka generiralo bi globalni optimum ili točnu aproksimaciju. Različiti algoritmi implementiraju navedena tri koraka na različite načine, ali ideja ostaje ista - iterativno ažuriranje postupka za generiranje kandidatnih rješenja tako da generirana kandidatna rješenja stalno poboljšavaju kvalitetu.

Algoritam 1: EDA

```
1 t = 0;
2 generiraj početni model M(0) da reprezentira jednoliku razdiobu nad
    dopuštenim rješenjima;
3 while nije zadovoljen uvjet završetka evolucijskog procesa do
4     P = generiraj N>0 kandidatnih rješenja uzorkovanjem M(t);
5     F = evaluiraj kandidatna rješenja u P;
6     M(t+1) = prilagodi model(P, F, M(t));
7     t = t + 1;
```

EDA algoritmi pripadaju razredu evolucijskih algoritama. Glavna razlika između EDA algoritama i većine uobičajenih evolucijskih algoritama je da evolucijski algoritmi generiraju nova rješenja koristeći implicitnu distribuciju definiranu jednim ili više varijacijskim operatorima, dok EDA algoritmi koriste eksplicitnu distribuciju kodiranu pomoću Bayesove mreže, multivarijatne normalne distribucije ili neki drugi model.

Ključna ideja EDA algoritama je gledati populaciju prethodno posjećenih dobrih rješenja kao podatak, naučiti model (ili teoriju) tih podataka i koristiti rezultirajući model koji će utvrditi gdje bi mogla biti druga dobra rješenja. Ovaj je pristup snažan, omogućujući algoritmu za pretraživanje da uči i da se prilagođava s obzirom na optimizacijski problem koji rješava.

Korištenjem eksplicitnih vjerojatnosnih modela u optimizaciji omogućilo je rješavanje optimizacijskih problema koji su teški za većinu uobičajenih evolucijskih algoritama i tradicionalne optimizacijske tehnike, kao što su problemi s visokom razinom epistaze.[2] Velika prednost EDA algoritama je ta da pružaju stručnjaku za optimizaciju niz vjerojatnosnih modela koji otkrivaju puno informacija o problemu koji se rješava. Te se infomacije mogu koristiti za dizajniranje problemski specifičnih operatora susjedstva za lokalno pretraživanje, za utjecanje na buduće pokretanje EDA algoritma na sličnom problemu ili za stvaranje učinkovitog računalnog modela problema.

[2]U genetici epistaza je pojava u kojoj vrijednost jednog gena ovisi o prisutnosti jednog ili više modifikatora gena.

3.1.1. Jednostavna simulacija EDA algoritama

Kako bi bolje razumijeli postupak EDA algoritama, dati ćemo jednostavnu simulaciju izvođenja. U simulaciji pretpostavimo da su kandidacijska rješenja predstavljena nizom bitova fiksirane veličine. Evaluacijska funkcija koja se maksimizira je OneMax, definirana kao suma jedinica u ulaznom nizu bitova $(X_1, X_2, \ldots, X_n)$:

$$\text{OneMax}(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{n} X_i.$$

Kvaliteta kandidacijskog rješenja je razmjerna broja jedinica u ulaznom nizu, a optimum je niz svih jedinica.

Simulacija koristi vjerojatnosni vektor p koji za n-bitne binarne nizove ima n komponenti, $p = (p_1, p_2, \ldots, p_n)$. Komponenta p_i predstavlja vjerojatnost pojave jedinice na poziciji i u rješenju. Da bi naučili vjerojatnosni vektor, p_i se postavlja kao omjer jedinica u položaju i promatranom u odabranom skupu rješenja. Kako bi uzeli uzorak novog kandidacijskog rješenja $(X_1, X_2, \ldots, X_n)$, ispitane su komponente vjerojatnosnog vektora i svaka X_i je postavljen na jedinicu s vjerojatnosti p_i i na nulu s vjerojatnosti $1 - p_i$.

Kako bi simulacija ostala jednostavna, funkcija OneMax je definirana na 5 bitova, veličina populacije je $N = 6$ i izbor skraćivanjem s pragom 50%. Tablice u nastavku prikazuju inicijalizaciju i sljedeće dvije iteracije simulacije EDA. Inicijalna populacija kandidatnih rješenja je slučajno izgenerirana. Izbor skraćivanjem izabere 50% najboljih kandidacijskih rješenja na temelju evaluacije koristeći funkciju OneMax (dobrota) kako bi stvorio skup obećavajućih rješenja.[3] Vjerojatnosni vektor stvara se na temelju odabranih rješenja i uzorkuje se razdioba koju kodira vjerojatnosni vektor za generiranje novih kandidacijskih rješenja. Dobivena populacija zamjenjuje prijašnju populaciju i postupak se ponavlja.

U obje iteracije simulacije, prosječna vrijednost funkcije OneMax u novoj populaciji veća je od prosječne vrijednosti u populaciji prije odabira. Budući da se vrijednost funkcije OneMax povećava s brojem jedinica, možemo očekivati da će ukupna kvaliteta populacije rasti tijekom vremena.

[3] U tablicama su 50% boljih rješenja iznad crte.

Rješenje	Dobrota
10111	4
11101	4
11001	3
10010	2
00001	1
00010	1

Rješenje	Dobrota
11101	4
11011	4
11101	4
11011	4
10101	3
10101	3

Rješenje	Dobrota
11111	5
11101	4
11011	4
11101	4
11101	4
11001	3

Ovo bio je primjer najjednostavnijih vrsta EDA. Model, vjerojatnosni vektor, ima fiksnu strukturu. Ovaj razred je prilično ograničen u onim što može učiniti. Postoje drugi razredi EDA algoritama koji omogućuju obilate vjerojatnosne modele sposobne za određivanje povezanosti između varijabli određenog problema. To rezultira u složenijem postupku izgradnje modela, no pokazalo se da je dodatni napor vrijedan, pogotovo prilikom rješavanja teških optimizacijskih problema.

3.2. Višerazinsko pretraživanje

Algoritmi višerazinskog pretraživanja (engl. *multi-scale search*) imaju svojstvo da im se operatori pretraživanja uzastopno redefiniraju (mijenjaju), stvarajući time novi postupak pretraživanja na višoj razini uređenosti. Ova redefinicija operatora pretraživanja postiže se identifikacijom jako koreliranih podskupova varijabli višestrukih rješenja dobivenih lokalnom pretragom u trenutnom prostoru pretraživanja time smanjujući dimenzionalnost naknadnog pretraživanja. Izrada operatora makro-varijacije iskorištava te korelacije omogućavanjem specifičnih istovremenih promjena vrijednosti u varijablama problema, tj. ovi operatori rade modularnu varijaciju nad rješenjima. Metoda višerazinskog pretraživanja inicijalno pretražuje kombinacije izvornih varijabli problema i kako napreduje, mijenja prostor pretraživanja tako da počinje pretraživati u kombinacijama modula i ova se transformacija može ponoviti u nekoliko razina uređenosti (Mills et al., 2014). U nastavku ćemo operatore makro-varijacije zvati (varijacijske) jedinice.[4]

Glavna obilježja ovog pristupa su (Mills et al., 2014):

- evolucijsko pretraživanje s trenutnim jedinicama,
- redefiniranje varijacijskih jedinica,
- razdvajanje vremenskih razina te

[4]Opisni pridjev varijacijske ćemo koristiti kad želimo istaknuti da jedinice variraju tijekom vremena.

– raznoliko uzorkovanje.

Evolucijsko pretraživanje s trenutnim jedinicama: u srži višerazinskog pretraživanja je jednostavan proces varijacije i selekcije. Taj proces radi pretragu nad susjedstvom rješenja. Susjedstvo je definirano skupom promjenjivih jedinica koji se mijenja tijekom izvođenja algoritma. Djelovanjem pojedinačnih jedinica nad trenutnim rješenjem dobiju se njegova susjedna rješenja. Ovaj postupak je vrlo sličan lokalnom pretraživanju, no bitna razlika je u tome što jedinice koje određuju prelazak iz trenutnog rješenja u neko od njegovih susjednih postaju sve veće tijekom vremena te time mijenjaju (smanjuju) prostor pretraživanja. Kako bismo razlikovali ovaj postupak pretraživanja od lokalnog pretraživanja, nazovimo ga postupkom pretraživanja iskorištavanjem jedinica (engl. *unit-exploiting search*). U ovom radu taj postupak je izveden metodom uspona na vrh (engl. *hill climbing method*), no to može biti bilo koja metoda koja iskorištava jedinice trenutne razine.

Redefiniranje varijacijskih jedinica: početna razina pretraživanja jedinica je definirana danim varijablama problema, dok su iduće razine pretraživanja definirane kombinacijama (moduli) varijabli problema. Jedinice viših razina definiraju prostore manjih dimenzija, a temelje se na informaciji strukturnih veza koja je naučena iz skupa kvalitetnih rješenja (najčešće lokalni optimumi) dobivenih pretraživanjem na prethodnoj razini. U ovom radu informacija se zasniva na korelaciji između jedinica prisutnih u tim rješenjima.

Razdvajanje vremenskih razina: skup rješenja iz jedne razine, korišten za identificiranje strukture za sljedeću razinu, mora otkriti kombinacije jedinica visokih vrijednosti. Zbog toga se postupak pretraživanja iskorištavanjem jedinica ('lokalna' pretraga jedinicama trenutne razine) izvršava brzo, a nove jedinice se formiraju relativno sporo. S dovoljnim razmakom u tim vremenskim razmacima svako upotrebljeno rješenje će biti na ili u blizini lokalnog optimuma u susjedstvu koje definiraju trenutne jedinice. Time je pristup višerazinsko pretraživanje sačinjen ponavljanjem dviju faza: a) pretraživanje s trenutnim varijacijskim jedinicama i b) identifikacija povezanosti za stvaranje (formiranje) novih varijacijskih jedinica.

Raznoliko uzorkovanje: skup rješenja korišten za identifikaciju strukture za sljedeću razinu pretraživanja mora izbjegavati prekomjerno konvergiranje na određeni način u postizanju visoke vrijednosti. To se postiže tako da se početno rješenje pretraživanja iskorištavanjem jedinica svaki put generira slučajno.

4. Algoritam Macro-H

4.1. Pseudokôd algoritma

Macro-H (jednostavni algoritam višerazinske optimizacije) je definiran za optimizaciju funkcije $f\colon \{0,1\}^n \to \mathbb{R}$ što se vidi iz ulaza Algoritma 2.

Radi bolje preglednosti postupak pretraživanja iskorištavanjem jedinica je definiran zasebno kao Algoritam 3.

Na samom početku prije ulaska u glavnu petlju algoritma, obavlja se inicijalizacija skupa jedinica V sa svim primitivnim jedinicama; jedna jedinica za svaku vrijednost svake varijable problema (Algoritam 2, linija 1).[1] Nakon inicijalizacije slijede tri ponavljajuće faze.[2]

U prvoj fazi se radi postupak pretraživanje iskorištavanjem jedinica koristeći trenutne varijacijske jedinice za pronalazak nekoliko različitih lokalnih optimuma (Algoritam 2, linije 3-6, Algoritam 3). Kako se točno obavlja postupak pretraživanja bit će objašnjeno u nastavku.

Slijedi faza izračunavanja mjere zajedničke pojave u matrici udruženja A, na temelju korelacija između jedinica prisutnih u optimumima nađenim prijašnjom fazom (Algoritam 2, linije 7-12). U ovom dijelu pseudôkoda može se vidjeti relacija "se slaže s" između rješenja pretraživanja i jedinice no prije objašnjenja što ona točno znači te kako se izračunava mjera zajedničke pojave u matrici udruženja potrebno je objasniti notaciju jedinice.

Jedinica je uređeni par (l, a) gdje su $l \in \{0,1\}^n$ i $a \in \{0,1\}^n$. l predstavlja listu pripadnosti svake varijable u jedinici. Pod time se misli da ako je za određenu poziciju liste $l[i] = 1$ to znači da jedinica definira vrijednost za i-tu varijablu, dok za $l[i] = 0$

[1] U liniji 1 algoritma Macro-H koristimo notaciju za povezivanje slova u niz kao $b^k = bb\ldots b$. Na primjer, $0^3 10^2 = 000100$. b^0 predstavlja prazan niz.

[2] U prethodnom potpoglavlju smo rekli da je pristup višerazinsko pretraživanje sačinjen ponavljanjem dviju faza što i jest točno. Ovdje drugu fazu dijelimo na dva dijela zbog lakšeg objašnjavanja pseudokôda.

ne definira vrijednost i-toj varijabli. a predstavlja listu dodjeljivanja svakoj varijabli jedinice. Vrijednosti $a[j]$ za takve j gdje je $l[j] = 0$ nisu bitne i mogu se postaviti proizvoljno.

Sljedeći primjer ilustrira reprezentaciju pripadnosti i dodjeljivanja za 8-bitni problem. Ovdje pišemo $a[i] = *$ gdje jedinica ne specificira vrijednost za poziciju i (tj. $l[i] = 0$)

$$u_1 = (00100000, **0*****)$$

$$u_2 = (00010100, ***1*0**)$$

$$u_3 = (01000000, *1******)$$

$$u_4 = (11000010, 00****1*)$$

$$u_5 = (00101000, **1*1***)$$

$$u_6 = (00000001, *******1).$$

Sada možemo pojasniti prethodno spomenutu relaciju "se slaže s". Kažemo da se rješenje pretraživanja $y \in \{0,1\}^n$ slaže s jedinicom (l, a) ako vrijedi $l[i] = 1 \Rightarrow (y[i] = a[i])$ za sve $i \in \{1, 2, \ldots, n\}$. Ova relacija "se slaže s" između rješenja pretraživanja i jedinica je korisna kod određivanja snage udruživanja, tj. mjere zajedničke pojave između svakog para jedinica u i v pa se sada može objasniti kako se izračunava. Za neki par jedinica u i v se izračuna koliko rješenja dobivenih lokalnom pretragom par jedinica zadovoljava relaciju "se slaže s" te se taj broj podijeli s brojem rješenja koji zadovoljavaju relaciju s jednim od para jedinica. Ako je ovaj razlomak manji od 1, tj. ako postoji neko rješenje koje je u relaciji sa samo jednom jedinicom (u ili v), tada se u matrici udruženja na poziciji $A_{u,v}$ postavlja vrijednost 0, inače 1 (Algoritam 2, linija 9).

Algoritam 2: Algoritam Macro-H

Input: Ciljna funkcija $f\colon \{0,1\}^n \to \mathbb{R}$; broj 'paralelnih' lokalnih pretraga $d \in \mathbb{N}$

Output: Najbolja pronađeno rješenje pretraživanja

`// Inicijaliziraj skup jedinica V sa svim primitivnim jedinicama`

$V := \left\{(0^{i-1}10^{n-i}, 0^n), (0^{i-1}10^{n-i}, 1^n) | i \in \{1, \ldots, n\}\right\}$;

while *istina* **:**

  `// Izvrši 'lokalnu' pretragu` d `puta`

  for $\forall i \in \{1, \ldots, d\}$ **:**

    $y_i :=$ UE-Search(f, V); // Vraća $y_i \in \{0,1\}^n$

    if y_i *zadovoljava kriterij zaustavljanja* **:**

      return y_i

  `// Izračunaj mjeru zajedničke pojave` $A_{u,v}$

  for $\forall u, v \in V$ **:**

    if $\left\{i \in \{1, \ldots, d\} | y_i \text{ se slaze s } u \text{ ili } v\right\} \neq \emptyset$ **:**

      $c := \frac{|i \in \{1,\ldots,d\} | y_i \text{ se slaze s } u \text{ i } v|}{|i \in \{1,\ldots,d\} | y_i \text{ se slaze s } u \text{ ili } v|}$;

    else:

      $c :=$ nedefiniran;

    $A_{u,v} := \begin{cases} 1\text{, ako je } c = 1 \\ 0\text{, inače.} \end{cases}$

  `// Ažuriraj skup jedinica V`

  $V' := \emptyset$;

  for $\forall u \in V$ **:**

    Makni u iz V;

    if $\exists y \in \{y_1, y_2, \ldots, y_d\} : y$ se slaze s u **:**

      `// konstruiraj grupu` (l, a)

      $(l, a) := u$;

      for $\forall v = (l', a') \in V$ *u slučajnom redoslijedu* **:**

        if $A_{u,v} = 1$ **:**

          for $\forall i \in \{1, \ldots, n\}$ *sa* $l'[i] = 1$ *i* $l'[i] = 0$ **:**

            $l[i] := 1$;

            $a[i] := a'[i]$;

          Makni v iz V;

      $V' := V' \cup \{(l, a)\}$;

    $V := V'$;

Zadnja, i vjerojatno najvažnija faza je ažuriranje skupa jedinica V kombinirajući najsnažnije povezane jedinice u kompozitne jedinice (Algoritam refmacro, linije 13-25). Kako bismo odredili koji su parovi jedinica najsnažnije povezani koristimo mjeru zajedničke pojave koju smo ažurirali za svaki par jedinica u prethodnoj fazi. Macro-H radi čvrste veze između jedinica; stoga matrica udruženja za neki par jedinica u i v, $A_{u,v}$, može poprimiti vrijednost nula ili jedan. Na koji se točno način izvodi ovo kombiniranje jedinica u kompozitne jedinice objašnjeno je idućim primjerom koristeći definirane jedinice iz prethodnog primjera. Uzmimo da nakon druge faze matrica A ima sljedeće vrijednosti: $A_{1,2} = 1, A_{1,3} = 1, A_{1,v} = 0, \forall v \neq [2,3]$ i $A_{4,6} = 1, A_{4,v} = 0, \forall v \neq 6$, iz čega slijedi da nakon treće faze imamo kompozitne jedinice:

$$u_1^+ = (01110100, *101*0**)$$

$$u_4^+ = (11000011, 00****11).^3$$

Kako se vidi i iz navedenog primjera različite jedinice sudjeluju u različitim grupacijama i postoje situacije kada nisu sve jedinice korištene (kao što je u ovom slučaju to u_5).

Primijetimo kako, iako matrica asocijacije određuje povezivanje između para jedinica, u formiranju nove kompozitne jedinice može sudjelovati bilo koji broj sastavnih jedinica i kako nema uvjeta da sastavne jedinice budu susjedne niti da moraju biti istog reda. Pod susjedne se misli da su definirane pozicije varijabli u listi pojedine jedinice jedna do druge (kao što su u prijašnjim primjerima jedinice u_1 i u_3), a kod reda jedinice se misli na broj varijabli za koje jedinica definira vrijednost.

[3] Gornji indeks + u kompozitnim jedinicama samo naznačuje kako se više ne radi o početnim jedinicama u_1 i u_4.

Algoritam 3: Algoritam UE-Search

Input: Ciljna funkcija $f\colon \{0,1\}^n \to \mathbb{R}$; skup jedinica za pretraživanje V

Output: Najbolje pronađeno rješenje pretraživanja

```
// Konstruiraj početno rješenje x slučajno na temelju
   skupa V
```

1 $y := 0^n$;
2 **while** $\exists i \in \{1, \ldots, n\} : y[i] = 0$ **:**
3 **for** $\forall (l, a) \in V$ *u slucajnom redoslijedu* **:**
4 **for** $\forall i \in \{1, \ldots, n\}$ **:**
5 **if** $y[i] = 0$ *i* $l[i] = 1$ **:**
6 $y[i] := 1$;
7 $x[i] := a[i]$;
8 Fiksiraj slučajan redoslijed, ρ, na skupu V;
9 **while** *kvaliteta rješenja raste* **:**
  `// Stvori y kao slučajni 'susjed' x-a`
10 $y := x$;
11 **while** $y = x$ **:**
12 Neka $(l, a) \in V$ bude sljedeći element u redoslijedu ρ; Ako je ρ iscrpljen onda postavi ρ kao novi slučajni redoslijed na skupu V;
13 **for** $\forall i \in \{1, \ldots, n\}$ **:**
14 **if** $l[i] = 1$ **:**
15 $y[i] := a[i]$
  `// Izaberi bolje rješenje pretraživanja`
16 **if** $f(y) > f(x)$ **:**
17 $x := y$
18 **return** x

U prijašnjem opisu pseudokôda algoritma Macro-H preskočen je opis postupka pretraživanja kako se ne bi izgubio tijek misli.

Postupak pretraživanja, uz funkciju za koju se traži optimum, dodatno dobiva i skup jedinica V koje definiraju susjedstvo nekog rješenja u prostoru rješenja.[4]

Na početku se konstruira početno rješenje x u prostoru rješenja slučajno na temelju skupa V (Algoritam 3, linije 1-7).[5] Nakon što je kontruirano početno rješenje slijedi

[4] Ovo vidimo iz ulaza algoritma 2.

[5] Kada smo pričali općenito o pristupu višerazinsko pretraživanje rekli smo da je raznoliko uzorkovanje jako bitno kako bi izbjegavati prekomjerno konvergiranje na određeni način.

ponavljanje samo jedne faze.

Nad kopijom našeg početnog rješenja nadodajemo slučajnim redoslijedom jedinicu po jedinicu dok se ne promjeni sadržaj neke od varijabli (Algoritam 3, linije 11-15).[6] Novo rješenje je susjed početnog rješenja s obzirom na trenutne jedinice. Ako je dobrota novog rješenja bolja od dobrote početnog rješenja, njega postavljamo kao novo početno rješenje te se ovaj postupak iznova ponavlja (Algoritam 3, linije 16-17).

Uvjet izlaska iz ovog postupka je kada se trenutno rješenje ne uspije zamijeniti bilo kojim od svojih susjeda. Ovim uvjetom je osigurano nalaženje lokalnog optimuma (lokalnog s obzirom na trenutne jedinice) koji nam je potreban.[7]

4.2. Implementacija algoritma

Sama implementacija algoritma ne iziskuje puno vremena no ipak treba pripaziti kod nekih detalja. Nije uvijek dobro držati se strogo pseudôkoda jer to može dovesti do puno veće (asimptotske) složenosti nego što je to potrebno. Tako se i autor ovog rada tijekom prve implementacije druge faze (Algoritam 1, linije 5-10) držao samo pseudôkoda te time jako usporio izvođenje cijelog algoritma. Naime, složenost prve i treće faze je $O(n^2 \log(n))$ dok je složenost druge faze $O(n^3 \log(n))$.

U prvoj fazi postupka pretraživanja iskorištavanjem jedinica koji je složenosti $O(n^2)$ se poziva d puta, a d je parametar algoritma i kao što će se pokazati u idućem poglavlju vidjeti njega postavljamo na vrijednost $O(\log(n))$ iz čega slijedi da je složenost prve faze $O(n^2 \log(n))$. Za objašnjenje složenosti za treću fazu čitatelj se upućuje na rad (Mills et al., 2014).

U drugoj fazi, kôd unutar petlje *for* izvršava se $O(n^2)$ puta, a izračunavanje uvjeta unutar if-a je $O(nd)$. Da je izračunavanje uvjeta unutar naredbe *if* složenosti $O(nd)$ može se vidjeti pozivajući se na činjenicu da relacija "se slaže s" provjerava uvjet $l[i] = 1 \Rightarrow (y[i] = a[i])$ za svaku varijablu $i \in \{1, \ldots n\}$ te se dodatno ta relacija izvodi za svaki od d lokalnih optimuma. Time je potvrđena iznesena složenost druge faze. Ali ova složenost je takva zbog strogog praćenja pseudôkoda. Relaciju "se slaže s" možemo izračunati prije petlje *for* jer se rješenja pretraživanja i jedinice tijekom njenog izvođenja ne mijenjaju, te se time smanji složenost druge faze na $O(n^2 d)$. Implementacija druge faze može se vidjeti u isječku programskog koda 4.1. [8]

[6] Pod dodavanjem jedinice rješenju se misli na dodjeljivanje vrijednosti rješenja onim varijablama kojima jedinica definira vrijednost (Algoritam 3, linije 13-15).

[7] U idućem poglavlju ćemo vidjeti zašto je ovaj uvjet potreban.

[8] Samo mali osjećaj u kojoj mjeri ova složenost može utjecati na izvođenje algoritma. U idućem će

Isječak programskog koda 4.1: Ažuriranje mjere zajedničke pojave

```
private void updateA() {

  boolean[][] agreement = new boolean[units.size()][solutions.size()
      ];
  for(int u = 0; u < units.size(); u++) {
    for(int y = 0; y < solutions.size(); y++) {

      if(units.get(u).agrees(solutions.get(y))) {
        agreement[u][y] = true;
      }
    }
  }
  A = new boolean[units.size()][units.size()];
  for(int u = 0; u < units.size(); u++) {
    for(int v = u + 1; v < units.size(); v++) {

      boolean flag = false;
      for(int y = 0; y < solutions.size(); y++) {
        if(agreement[u][y] && agreement[v][y]) {
          flag = true;

        } else if(agreement[u][y] || agreement[v][y]) {
          flag = false;
          break;
        }
      }

      if(flag) {
        A[u][v] = true;
        A[v][u] = true;
      }
    }
  }
}
```

se poglavlju govoriti o analizi algoritma na funkciju SBB. Testira se utjecaj povećanja početnog ulaza na vrijeme izvođenje algoritma. Korištenjem prve implementacije ukupno trajanje izvođenja za sve dane ulaze je bilo gotovo 10 sati, a nakon ove male promjene niti 10 minuta .

5. Analiza i rezultati eksperimenata

Kao mjeru vremena za analizu izvođenja algoritma Macro-H koristit će se broj poziva funkcije za evaluaciju. Funkcije na kojima će se testirati naš optimizacijski algoritam nisu stvarni problemi u smislu da je njihovo rješenje (optimum) nepoznato i da ga je potrebno tražiti nego skup testova koji na način na koji su definirani, svaki daje određenu zahtjevnost kod njihove optimizacije koja se može naći i kod stvarnih optimizacijskih problema. Rezultati testiranja optimizacijskog algoritma na tom skupu funkcija dosta govore o kvaliteti samog algoritma te uvidom u te rezultate može se unaprijed u većoj ili manjoj mjeri predvidjeti ponašanje algoritma i na stvarnim problemima. Taj nam skup može ujedno i poslužiti za usporedbu među algoritmima.

Razred funkcija kojem pripadaju testovi imaju identificirajuće module koji aditivno pridonose vrijednosti funkcije. Takve funkcije se zovu (m, k)-odvojive. Slijedi njihova definicija.

Definicija 5. Funkcija $f\colon \{0,1\}^n \to \mathbb{R}$ je (m, k)-odvojiva, gdje su $m, k \in \{1, \ldots, n\}$, ako postoji particija $\{1, \ldots, n\}$ na m disjunktnih skupova $I_1, \ldots, I_m$, i ako postoji odgovarajući broj pseudo-Booleovih funkcija $g_1, \ldots, g_m$ gdje su $g_j\colon \{0,1\}^{|I_j|} \to \mathbb{R}$ takvih da vrijedi:

$$f(x) = \sum_{j=1}^{m} g_j\Big(x_{i_{j,1}} x_{i_{j,2}} \ldots x_{i_{j,|I_j|}}\Big)$$

za svaki $x = x_1 \ldots x_n \in \{0,1\}^n$, $I_j = \Big\{i_{j,1}, \ldots, i_{j,|I_j|}\Big\}$ i $|I_j| \leq k$ za svaki $j \in \{1, \ldots, m\}$.

Očigledno je svaka funkcija (1, n)-odvojiva. Ovdje ćemo raditi s točno $(n/k, k)$-odvojivim funkcijama. Najjednostavniji način izrade ovakvih tipova funkcija je da se uzme neodvojiva funkcija ((1, k)-odvojiva), te se zatim napravi konkatenacija n/k kopija te funkcije svaka definirana nad k bitova. Na takav način su i naše test funkcije izgrađene. Gore navedena definicija je primijenjenja u idućim potpoglavljima.

5.1. Eksperimenti na razredu problema SBB*

Prva test funkcija je izgrađena od niza neodvojivih funkcija TwoMax. TwoMax je gotovo simetrična funkcija s jednim globalnim (niz od samih jedinica) i jednim lokalnim optimumom (niz od samih nula). Konkatenaciji funkcija TwoMax dajemo naziv SBB (engl. *scalable building-block problem*).

Definicija 6. Funkcija $\mathrm{TwoMax}\colon \{0,1\}^n \to \mathbb{N}$ je definirana kao

$$\mathrm{TwoMax}(x) = \max\left\{ \sum_{i=1}^{n} x[i], n - \sum_{i=1}^{n} x[i] \right\} + c \prod_{i=1}^{n} x[i]$$

za svaki $x \in \{0,1\}^n$ i $c \geq 0 \in \mathbb{N}$.

Neka su $n, k \in \mathbb{N}$ takvi da vrijedi $k > 1$ i $n/k \in \mathbb{N}$. Funkcija $\mathrm{SBB}\colon \{0,1\}^n \to \mathbb{N}$ je definirana kao

$$\mathrm{SBB}(x) = \sum_{i=1}^{n/k} \mathrm{TwoMax}(x^{(i)})$$

za svaki $x = x[1]x[2]\ldots x[n] \in \{0,1\}^n$ gdje je $x^{(i)}$ i-ti dio duljine k u x, $x^{(i)} = x[(i-1)k+1]x[(i-1)k+2]\ldots x[ik]$.[1]

Recimo da imamo funkciju SBB definiranu kao konkatenacija 3 podfunkcija TwoMax svaka definirana na 3 varijable. Vrijednost ulaznog vektora funkcije SBB $x_1 \ldots x_9$ je TwoMax($x_1x_2x_3$) + TwoMax($x_4x_5x_6$) + TwoMax($x_7x_8x_9$).

U algebarskom izrazu TwoMax funkcije parametar c odvaja globalni od lokalnog optimuma. Nagib prema oba optimuma je jednak te se globalni optimum ističe samo u krajnjoj točki (niz jedinica). Način na koji je lokalna pretraga (Algoritam 2) implementirana, vjerojatnost pronalaska globalnog optimuma svake podfunkcije TwoMax je $\frac{1}{2}$, neovisno o njihovoj veličini, te je vjerojatnost pronalaska globalnog optimuma funkcije SBB sačinjene od k podfunkcija TwoMax jednaka $\frac{1}{2^k}$.

Izvođenje algoritma se neće testirati samo na funkciji SBB nego na razredu funkcija SBB, gdje pod razredom se misli na generalizaciju. Prije same formalne definicije, dat ćemo intuitivnu predodžbu na koji način se provodi generalizacija. Slijede dva koraka generalizacije:

- okretanje bitova te
- permutiranje.

[1] Odnosimo se na varijablu na poziciji i kao $x[i]$, a na više njih kao konkatenaciju, $x[i]x[j]x[z]$ za pozicije i, j, z.

Prvi korak: svaki ulazni vektor (argument funkcije) se preslikava u novi niz nula i jedinica proizvoljnim vektorom pomoću operacije XOR.[2] Operaciju XOR nad ulaznim i proizvoljnim vektorom bi bilo prikladno gledati na taj način da se na određenoj poziciji ulaznog niza bit mijenja iz jedan u nula ili obrnuto ako je na istoj poziciji u proizvoljnom vektoru jedinica, a ako je na istoj poziciji vrijednost nula u proizvoljnom vektoru, bit u ulaznom nizu se ne mijenja. Pokažimo ovo na primjeru.

Recimo da je rješenje 8-bitnog problema $x = 10010111$ i da je proizvoljni vektor dan kao $a = 00100110$. Ulazni vektor evaluacijske funckije je $x \oplus a = 10010111 \oplus 00100110 = 10110001$.

Drugi korak: permutiraju se bitovi ulaznog argument funkcije. Pokažimo i ovo na primjeru.

Recimo da je primjerak funkcije SBB definiran kao konkatenaciju 3 primjeraka funkcije TwoMax, gdje je svaka TwoMax funkcija definirana nad 4 bita. Ulazni vektor funkcije SBB možemo zapisati kao $a_1a_2a_3b_1b_2b_3c_1c_2c_3$, gdje je $a_1a_2a_3$ ulazni niz za prvu funkciju TwoMax, $b_1b_2b_3$ ulazni niz za drugu, te $c_1c_2c_3$ za treću funkciju TwoMax. Permutira li se redoslijed našeg vektora bitova s $\pi = (3, 2, 4, 6, 7, 5, 9, 1, 8)$ na ulazu u funkciji SBB će se pojaviti $a_3a_2b_1b_3c_1b_2c_3a_1c_2$. Ovo zapravo znači da mijenjanjem bitova našeg vektora na poziciji 1, 2 i 8 djelujemo nad ulaznim nizom prve podfunkcije TwoMax.

Za one koji više preferiraju strogi formalni izričaj prijašnje navedenih izjava slijedi definicija.

Definicija 7. Neka je $f\colon \{0,1\}^n \to \mathbb{R}$. Za bilo koji $a \in \{0,1\}^n$ definiramo $f_a\colon \{0,1\}^n \to \mathbb{R}$ kao $f_a(x) = f(x \oplus a)$. Za svaku permutacija π nad $\{1, 2, \ldots, n\}$ definiramo $f_\pi\colon \{0,1\}^n \to \mathbb{R}$ kao $f_\pi(x) = f(x[\pi(1)]x[\pi(2)]\ldots x[\pi(n)])$. Na kraju definiramo $f_{a,\pi} = (f_a)_\pi$.

Neka je $\mathcal{F} = \left\{f\colon \{0,1\}^n \to \mathbb{R}\right\}$ klasa funkcija. Definiramo $\mathcal{F}^* = \Big\{f_{a,\pi} | f \in \mathcal{F}$, $a \in \{0,1\}^n, \pi$ permutacija nad $\{1, 2, \ldots, n\}\Big\}$. Za $f\colon \{0,1\}^n \to \mathbb{R}$ definiramo $f^* := \{f\}^*$.[3] Ovime dobivamo da elementi modula (varijable podfunkcije koje aditivno pri-

[2] Hrv. naziv operacije XOR je isključivo ili.

[3] Može se pokazati da bilo koja heuristika pretraživanja koja tretira bitove nule i bitove jedinice simetrično kao i sve pozicije bitova će imati identičnu izvedbu za svaku funkciju $g \in f^*$ za svaku funkciju f, a tako i za funkciju SBB. Takve heuristike pretraživanja se zovu nepristranim i njih je smislenije koristiti nego pristane heuristike ako pristrana pretraga nije opravdana znanjem specifičnog problema. Većina je heuristika nasumičnog pretraživanja nepristrana u ovom smislu, to uključuje sve evolucijske algoritme bazirane na mutaciji and evolucijske algoritme sa uniformnim križanjem (Mills et al., 2014).

donose funkciji koju optimiraju, u ovom slučaju to su varijable podfunkcija TwoMax) nisu združene jedna uz drugu. Takvi problemi se zovu problemi sa slučajnim povezivanjem (engl. *random linkage problems*). Velika većina algoritama pretpostavlja da su elementi modula združeni jedan uz drugog, što ovakvom generalizacijom nije slučaj. Zbog toga pristupi poput genetskog algoritma ne uspijevaju rješavati ovakve tipove problema.

Teorem koji slijedi nam govori kakvi uvjeti moraju biti zadovoljeni kako bi algoritam mogao pronaći globalni optimum bilo koje funkcije $f \in \mathrm{SBB}^*$ u složenosti $O(n \log(n))$.

Teorem 1. Neka su $n, k \in \mathbb{N}$ dana kao $k > 1$ i $n/k \in \mathbb{N}$. SBB^* definiramo preko n bitova sa n/k dijelova jednakih duljina k. $\mathrm{Macro\text{-}H}$ pronalazi globalni optimum na bilo kojoj funkciji $f \in \mathrm{SBB}^*$ u $O(n + n \log(n/k))$ funkcijskih evaluacija sa vjerojatnosti $1 - O(n/k)$ ako su pojedinačni budžeti pretraživanja $\Theta(n)$ i dovoljno veliki i $d = \Theta(\log(n))$ i da vrijedi $d \geq 10.43 \ln(n)$.

Dokaz iznesenog teorema se može pronaći u radu (Mills et al., 2014). Ovdje se osvrćemo na dvije stvari koje su potrebne kod implementacije algoritma. Teorem kaže na koju vrijednost treba postaviti parametar d algoritma te da lokalnu pretragu treba izvoditi dovoljno dugo. U dokazu autori pod "dovoljno dugo" traže izvođenje lokalnog pretraživanja sve dok pretraživanje ne može naći bolje rješenje (lokalni optimum). To je razlog onakve implementacije algoritma UE-Search danog u prošlom poglavlju.

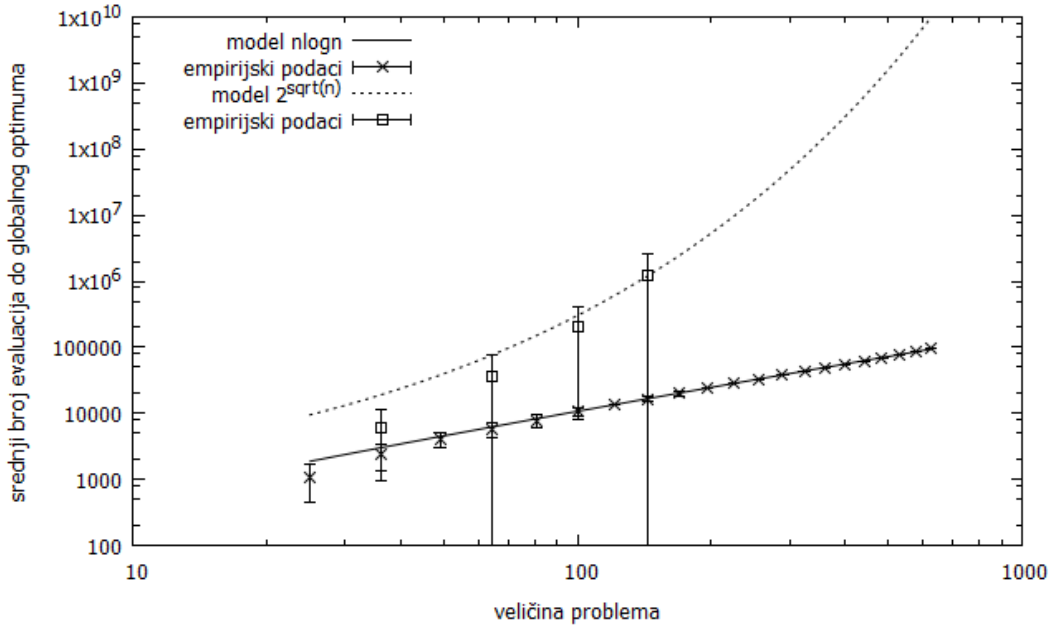
Problem SBB stvara poteškoće kod lokalne pretrage kad se sastoji od mnogo potproblema, a kod pretrage temeljene na mutaciji stvara probleme izlaska iz lokalnog optimuma u slučaju velikih potproblema. Iz toga razloga veličinu potproblema k postavljamo na $\sqrt{n}$ kako bismo uravnotežili ta dva faktora te time funkciju učinili maksimalno teškom. Iskoristit ćemo Teorem 1 kako bismo odredili vrijednost parametra d algoritma. Za testiranu veličinu problema $n \in [25, 625]$ parametar $d \in [34, 68]$. Dodatno prikazujemo izvođenje algoritma s onemogućenim mehanizmom pretvorbe jedinica (preskakanjem linija 7-25 Algoritma 1). Pretpostavka je da se bez pretvaranja jedinica algoritam ponaša identično kao ponovno pokretanje metode uspona na vrh te time predviđamo složenost izvođenja kao $o(2^{\sqrt{n}})$.[4] Slika 5.1 pokazuje rezultate testiranja. Vrijednosti na osima grafa prikazane su korištenjem logaritamske skale. Srednji broj evaluacija do globalnog optimuma je dobiven iz 100 nezavisnih ponav-

[4]Globalnu točku TwoMax podproblema nailazimo s vjerojatnosti $\frac{1}{2}$ te kako je SBB problem konstruiran od k takvih podproblema gdje je $k = \sqrt{n}$ slijedi navedena složenost za pronalazak globalne točke SBB problema.

ljanja. Točke na grafu predstavljene križićem i kvadratićem predstavljaju empirijske vrijednosti, križić s transformacijama jedinica, a kvadratić bez.

Pod pretpostavkom da je izvođenje algoritma složenosti $O(n\log(n))$ prema iskazu Teorema 1 i mi stoga prilagođujemo krivulju $b \cdot x\log(x)$ na podatke simulacije, onda bi za neki broj b ova krivulja trebala sve bolje opisivati kretanje podataka, tj. točke za sve veće veličine problema bi trebale biti bliže krivulji. To se svojstvo može vidjeti iz definicije 3 poglavlja 2.4.

Puno krivulja predstavlja model $y = b \cdot x\log(x)$ prilagođen točkama predstavljene križićima s vrijednosti $b = 23.2894$. Isprekidana krivulja predstavlja model $y = b \cdot 2^{\sqrt{x}}$ prilagođen točkama predstavljene kvadratićima s vrijednosti $b = 296.231$. Možemo primijetiti kako povećanjem veličine problema točke sve bolje leže na krivulji. Ovime potvrđujemo da je izvođenje algoritma Macro-H razmjerno očekivanju, $O(n\log(n))$, te isto tako vidimo važnost korištenja transformacije jedinica.



Slika 5.1: Srednji broj evaluacija do globalnog optimuma kod SBB funkcije

5.2. Eksperimenti na ulančanu funkciju zamke

Tipična funkcija zamke (engl. *trap function*) je dana izrazom (Whitley, 1993):

$$\mathrm{Trap}(x) = \begin{cases} \frac{a}{z}(z-u) & \text{, ako je } u \leq z \\ \frac{b}{l-z}(u-z) & \text{, inače,} \end{cases}$$

gdje su a i b konstante, z je položaj promjene nagiba, u je broj jedinica u ulaznom vektoru x i l broj varijabli ulaznog argument. Ulazni argument s nula jedinica ima vrijednost jednaku a. Kako broj jedinica raste, vrijednost funkcije linearno pada i ima vrijednost nula kada je broj jedinica jednak položaju promjene nagiba z. Poslije toga, vrijednost funkcije raste linearno i ima vrijednost jednaku $b > a$ za ulazni argument sa svim jedinicama. Varijabla z ima vrijednost $z = l - 1$.

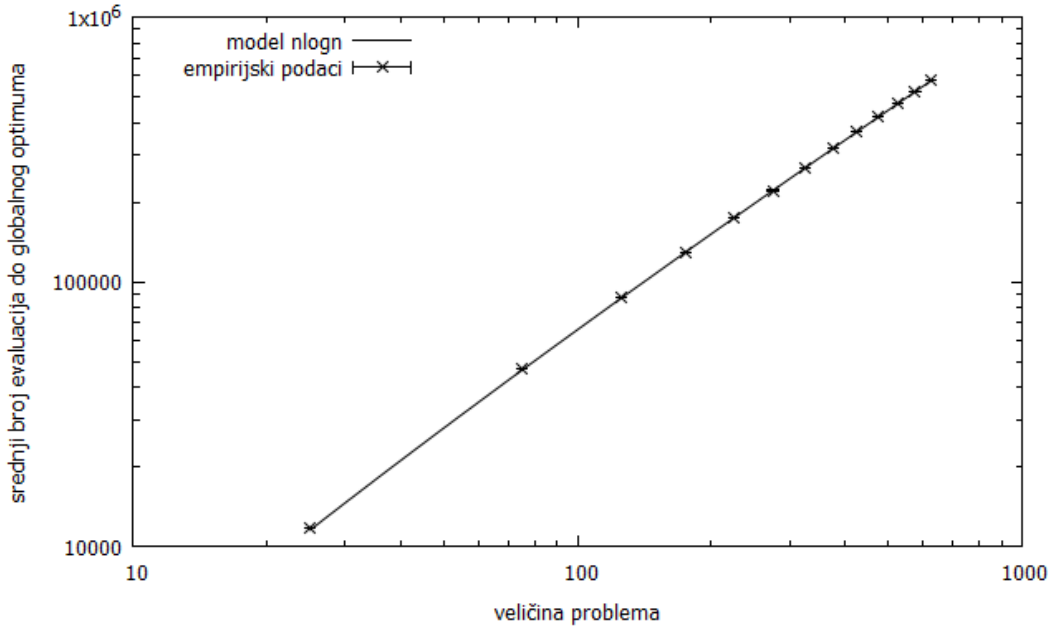
U radu (Mills et al., 2014) je navedeno proširenje ovdje danog Teorema 1 u prijašnjem potpoglavlju. Ovdje ćemo samo spomenuti kako iz tog proširenja slijedi da se i ova funkcija može riješiti u polinomijalnoj složenosti kada vrijedi za veličine podfunkcija zamke $k = O(\log(n))$ uz uvjet da se parametar d postavi na $\Theta(\ln(n)/p(n)^2)$, točnije $d \geq 3\ln(n)/p(n)^2$, gdje je $p(n)$ vjerojatnost pronalaska globalnog optimuma svake podfunkcije. Provjerimo ovaj iskaz.

Za ovaj eksperiment veličinu podfunkcija zamki postavljamo na $k = 5$. Kako je vjerojatnost pronalaska globalnog optimuma za funkciju zamke kao podfunkciju jednaka $\Theta(2^{-k})$, [5] slijedi da vrijednost parametra d moramo postaviti na $d \geq 96\ln(n)$, no mi ćemo postaviti na $d \geq 64\ln(n)$ kao u radu (Mills et al., 2014) što ne umanjuje asimtotsku složenost. Za testiranu veličinu problema $n \in [5, 125]$ parametar $d \in [104, 310]$. Slika 5.2 prikazuje izvođenje algoritma Macro-H. Vrijednosti na osima grada su prikazane u logaritamskoj skali. Srednji broj evaluacija do globalnog optimuma je dobiven iz 100 nezavisnih ponavljanja. Križić točke predstavljaju empirijske vrijednosti. Krivulja je model $y = b \cdot x\log(n)$ prilagođen križić točkama s vrijednosti $b = 143.088$. Na grafu možemo uočiti kako točke lijepo leže na krivulji čime potvrđujemo rješavanje problema konkatenacije funkcija zamke sa složenosti $O(n\log(n))$.

5.3. Eksperimenti na generalizaciju funkcije SBB*

Generaliziramo prethodno testiranu funkciju SBB* tako da dopuštamo bilo kakav odnos (blizinu) lokalnog i globalnog optimuma (ne nužno komplementarni, kao što je bio

[5] Za podfunkciju zamke veličine k lokalna pretraga će za bilo koju kombinaciju "upasti" u lokalni optimum osim ako je početno (slučajno izgenerirano) rješenje globalni optimum, ili, ako početno rješenje ima samo jednu nulu tada je vjerojatnost pronalaska globalnog optimuma $\frac{1}{2}$.



Slika 5.2: Srednji broj evaluacija do globalnog optimuma kod konkatenacije funkcija zamke

slučaj kod funkcije SBB*). Slijedi definicija funkcije ArbMax, podfunkcija naše test funkcije $\mathrm{SBB_A}$.

Definicija 8. Funkcija $\mathrm{ArbMax}\colon \{0,1\}^n \to \mathbb{N}$ je definirana kao

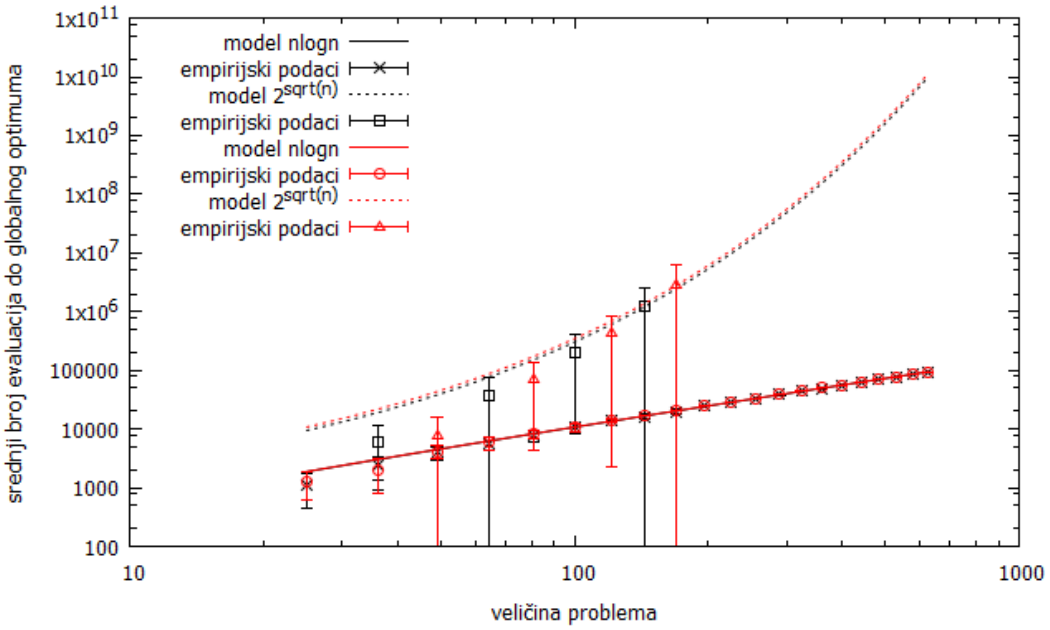
$$\mathrm{ArbMax}(x) = max\{n - \|t_1 - x\|, n - \|t_2 - x\|\} + \prod_{i=1}^{n}(t_1[i] \Leftrightarrow x[i])$$

za svaki $x \in \{0,1\}^n$, gdje je $\|a - b\|$ Hammingova udaljenost između uzoraka a i b, i gdje su $t_1, t_2 \in \{0,1\}^n$ ciljni nizovi bitova (ekstremi) s proizvoljnim položajima, tj. $\|t_1 - t_2\| > 1$.

$\mathrm{SBB_A}$ je konkatenacija n/k k-bitnih ArbMax podfunkcija.

Za testiranje ove funkcije svaka varijabla je postavljena ravnomjerno slučajno u t_1 i t_2, nezavisno za svaku podfunkciju i drugačije za svako ponavljanje. Za testiranu veličinu problema $n \in [25, 625]$ parametar $d \in [34, 68]$. Kao vrijednosti parametra d uzimaju se iste one koje smo koristili i u testiranju funkcije SBB*, točnije vrijednosti parametra d koje su zadovoljavale nejednakost $d \geq 10.43\ln(n)$. Razlog tome je pretpostavka da će se algoritam Macro-H ponašati jednako kao i kod funkcije SBB*. Srednji broj evaluacija do globalnog optimuma je dobiven iz 30 nezavisnih ponavljanja. Slika 5.3 pokazuje kako algoritam Macro-H gotovo identično rješava ovaj problem sa slučajnim ekstremima kao i specijalni slučaj kad imamo komplementarne ekstreme. Vrijednosti na osima grafa prikazane su korištenjem logaritamske skale. Podaci crne

boje se odnose na problem SBB^* dok se podaci crvene boje odnose na problem SBB_A. Za krivulje s omogućenim mehanizmom pretvorbe jedinica vrijednost parametra modela $y = b \cdot x\log(x)$ za problem SBB_A je $b = 23.4337$. Vrijednosti parametara problema SBB_A te problema SBB^* se tek na prvoj decimali razlikuju što potvrđuje našu pretpostavku.



Slika 5.3: Srednji broj evaluacija do globalnog optimuma kod SBB_A i SBB^* funkcija

Dvije značajke koje čine funkciju zamke i TwoMax funkciju teškim potproblemima su različite. Podfunkcija funkcije SBB, TwoMax, je fiksirane teškoće s povećanjem veličine problema te izazov za pronalazak globalnog optimuma funkcije SBB proizlazi povećanjem udaljenosti (veličina podfunkcija TwoMax) između lokalnog i globalnog optimuma što zahtjeva specifične k-bitne promjene da algoritma pobjegne iz lokalnog optimuma. Kod funkcije zamke vjerojatnost pronalaska globalnog optimuma pada s porastom veličine problema. Prethodni eksperimenti pokazuju da je Macro-H učinkovit u rješavanju problema SBB i konkatenacije funkcija zamke.

5.4. Eksperimenti na hijerarhijsku funkciju

Problemi ispitivani u prijašnjim potpoglavljima su jednorazinske strukture koje predstavlja značajne poteškoće za različite algoritamske pristupe. Macro-H rješava ove probleme tako da obuhvaća strukturu problema u kompozitne jedinice koje su, kada su prikadno iskorištene, dovoljne za rješavanje cjelokupnog problema. Dodatno, dvije

faze svake iteracije Macro-H mogu raditi s međusobnim izlazima te se time dobiva mogućnost rekurzije, tj. ponavljanja. Ova sposobnost je važna kod hijerarhijskih problema kod kojih interakcije na višoj razini uvode zavisnost između modula niže razine. Slijedi definicija hijerarhijske verzije problema SBB, HSBB na koji ćemo testirati naš algoritam.

Definicija 9. Varijable na jednoj razini pretvaraju se prije sudjelovanja u funkciji doprinosa na sljedećoj razini

$$\mathrm{Transformiraj}(x) = \begin{cases} 1 & \text{, ako je } x = 1^{|x|} \\ 0 & \text{, ako je } x = 0^{|x|} \\ \mathrm{null} & \text{, inače.} \end{cases}$$

Funkcija doprinosa koristi funkciju TwoMax,[6] i samo daje vrijednost različitu od nule ako su svi simboli (argumenti) definirani

$$\mathrm{F}(x) = \begin{cases} 1 & \text{, ako je } \exists x[i] = \mathrm{null} \\ \mathrm{TwoMax}(x) & \text{, inače.} \end{cases}$$

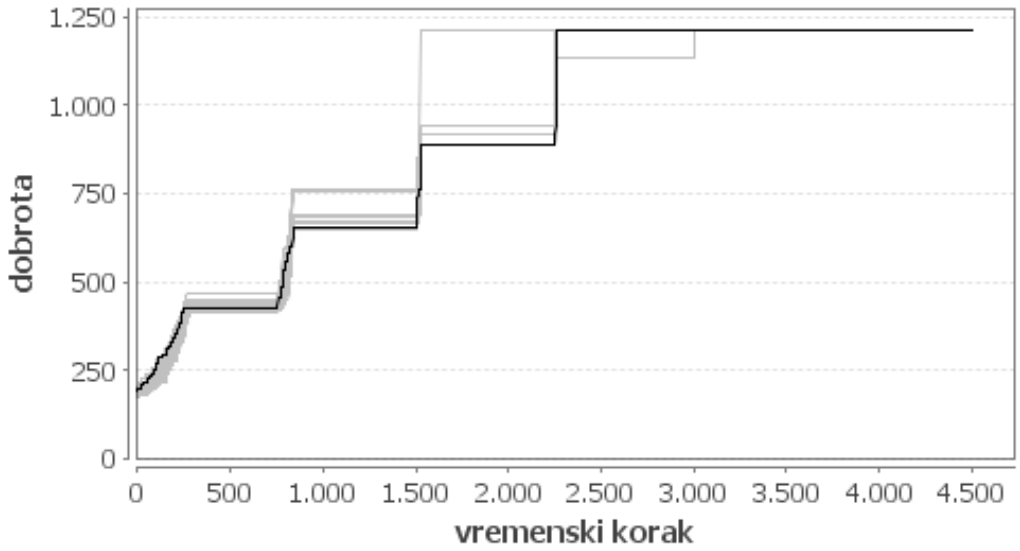
Ukupni hijerarhijski problem je kontroliran parametrom λ, broj razina, i n, i definiran je kao

$$\mathrm{HSBB}(x) = \sum_{i=1}^{n/k} \mathrm{F}(x^i) + \sum_{l=2}^{\lambda} k^{l-1} \sum_{i=1, i+=k}^{n/k^{l-1}} \mathrm{F}\left(\mathrm{Transformiraj}\left(x^{(ik^{l-2})}\right)^{(i)}\right)$$

gdje je $x^i = x[(i-1)k+1] \ldots x[ik]$, i $x^{ik} = x[(i-1)k^2+1] \ldots x[ik^2]$, i tako dalje. Zapis $f(x^{(ik^{l-2})})^{(i)} = f(x^{(ik^{l-2})}) f(x^{((i+1)k^{l-2})}) \ldots f(x^{((i+k-1)k^{l-2})})$ predstavlja konkatenaciju vrijednosti funkcija f, gdje je f u HSBB funkcija $\mathrm{Transformiraj}$. Veličinu bloka definiramo na najnižoj razini, k, kao $k = n^{1/\lambda}$. Doprinos dobrote bloka se povećava s razinom radi održavanja ukupnog doprinosa svake razine.

Ilustrirajmo kako se algoritam Macro-H ponaša na problemu HSBB. Slika 5.4 prikazuje napredovanje dobrote kroz vrijeme. U svrhu ovog grafa morali smo malo promjeniti algoritam. Konačna rješenja lokalne pretrage iz jedne razine se koriste kao početna rješenja iduće razine. Početna faza pretražuje s primitivnim jedinicama problema. Nakon pronalaska nekoliko različitih optimuma, fazom ažuriranja skupa jedinica V, jedinice postaju veće (otprilike za vremenski korak 750). Ponovnom lokalnom pretragom slijedi povećanje dobrote rješenja.

[6] Ovdje u TwoMax funkciji parametar c postavljamo na vrijednost nula te s time ne stvaramo razliku između dva optimuma.

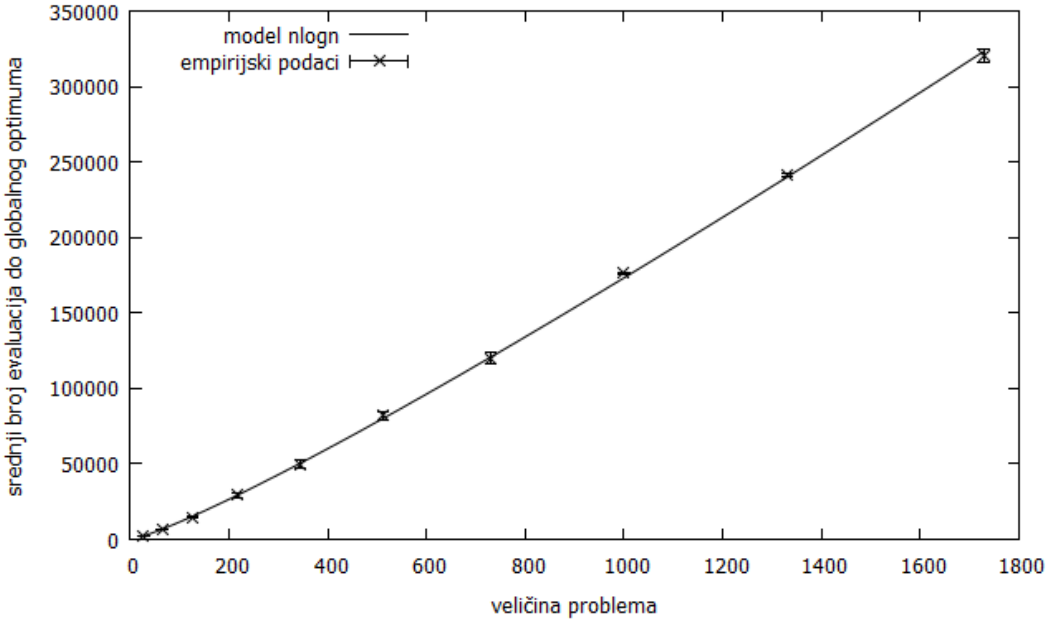


Slika 5.4: Napredovanje dobrote kroz vrijeme.

Za idući eksperiment koristimo $\lambda = 3$ i $n \in [27, 1728]$ (sa $k = n^{1/3} \in [3, 12]$). HSBB problem ima jednak broj blokova na najnižoj razini kao i primjerak funkcije SBB jednake veličine,[7] što znači da Teorem 1 daje dovoljnu vrijednost za parametar d. Ta vrijednost je isto dovoljna za više razine koje imaju manje blokova te zbog toga vrijednost d nije potrebno mijenjati tijekom izvođenja algoritma.

Vidjeli smo u prethodnim potpoglavljima kako Macro-H transformira jednu razinu kako bi riješio SBB problem u polinomijalnoj složenosti. Intuitivno razmišljajući, ako imamo hijerarhijski problem s konstantnim brojem razina problema SBB onda bi Macro-H trebao i taj problem rješavati u polinomijalnoj složenosti. Vrijedi $O(\lambda \cdot n \log(n)) = O(n \log(n))$ jer je λ konstantan broj. Čini se da podaci na slici 5.5 prate ovo razmišljanje čime potvrđujemo našu pretpostavku. Ovim rezultatom pokazujemo sposobnost algoritma Macro-H rekurzivne transformacije prostora problema u prostor manje dimenzije.

[7] Ovo u algebarskom izrazu funkcije HSBB možemo vidjeti kao prvu sumu.



Slika 5.5: Srednji broj evaluacija do globalnog optimuma kod HSBB funkcije

6. Zaključak

Uspješno je implementiran algoritam naveden u radu "Transforming Evolutionary Search into Higher-Level Evolutionary Search by Capturing Problem Structure", IEEE Trans. on Evolutionary Computation, October 2014, Vol 18, No 5. Implementacija algoritma je testirana na niz (m, k) točno odvojivih funkcija te na hijerarhijsku funkciju. Dobiveni rezultati ukazuju na podudarnost s rezultatima iznesenim u radu. Pokazano je kako osnovna intuicija da dobre kombinacije bitova mogu biti pronađene i iskorištene za identificiranje modula, a zatim dobre kombinacije modula mogu se pronaći i iskoristiti kako bi se identificirale rješenja za sljedeću razinu strukture problema.

Višerazinsko pretraživanje se razlikuje od postojećih pristupa zbog načina na koji se jednostavan postupak pretraživanja ponovno pokreće uzastopno na višim razinama uređenosti. To pruža osnovu za proširenja ovog pristupa koja se mogu primijeniti za neke od stvarnih problema.

LITERATURA

Bojana Dalbelo-Bašić. Neizrazito, evolucijsko i neuro računarstvo. *Zagreb: FER*, 2004.

Darko Grundler. Evolucijski algoritmi (i) pobude i načela. *AUTOMATIKA: časopis za automatiku, mjerenje, elektroniku, računarstvo i komunikacije*, 42(1-2):13–22, 2001.

Josef Kallrath i Thomas I Maindl. *Real optimization with SAP® APO*. Springer Science & Business Media, 2006.

Rob Mills, Thomas Jansen, i Richard A Watson. Transforming evolutionary search into higher-level evolutionary search by capturing problem structure. *IEEE Transactions on Evolutionary Computation*, 18(5):628–642, 2014.

Martin Pelikan, Mark W Hauschild, i Fernando G Lobo. Introduction to estimation of distribution algorithms. *MEDAL Report*, (2012003), 2012.

L Darrell Whitley. *Foundations of genetic algorithms 2*, svezak 2. morgan Kaufmann, 1993.

Optimizacija funkcije evolucijskim algoritmom uz analizu strukture rješenja

Sažetak

Problem optimizacije sveprisutan je u području računarske znanosti kao i u mnogim drugim područjima. Optimizacijski problem može biti definiran nad diskretnom domenom, kontinuiranom domenom ili mješavinom obaju domena. Ovisno o domeni problema, razvijeno je mnoštvo različitih pristupa za njihovo rješavanje. Dimenzionalnost domene pri tome predstavlja ograničavajući faktor, pri čemu probleme s visokodimenzijskim domenama često nije moguće rješavati učinkovito. Jedan od pristupa koji bi kod određenih vrsta problema mogao rezultirati povećanjem učinkovitosti pronalaska kvalitetnih rješenja jest dinamičko smanjenje dimenzionalnosti problema na način da se pojedine komponente domene grupiraju. Primjer takvog pristupa opisan je u radu "Transforming Evolutionary Search into Higher-Level Evolutionary Search by Capturing Problem Structure", IEEE Trans. on Evolutionary Computation, October 2014, Vol 18, No 5.

U okviru završnog rada proučen je navedeni rad te je ostvarena programska implementacija predloženog pristupa. Provedeno je vrednovanje i usporedba dobivenih rezultata s onima objavljenim u navedenom radu.

Ključne riječi: metaheuristika, algoritmi izgradnje modela, višerazinsko pretraživanje, Macro-H

Evolutionary Algorithm Based Function Optimization with Solution Structure Analysis

Abstract

The optimization problem is ubiquitous in the field of computer science as well as in many other areas. The optimization problem can be defined over a discrete domain, a continuous domain, or a mixture of both domain. Depending on the domain of the problem, a multitude of different approaches have been developed for solving them. Domain dimensionality represents a limiting factor and problems with high-dimensional domains can often not be effectively addressed. One approach that could result in certain types of problems increasing the efficiency of finding quality solutions is dynamic reducing the dimensionality of the problem in a way that individual components domains are grouped. An example of such approach is described in "Transforming evolutionary Search in Higher-Level Evolutionary Search by Capturing problem Structure ", IEEE Trans. On Evolutionary Computation, October 2014, Vol 18, No 5.

Within Bachelor thesis the mentioned work was studied and the program implementation of proposed approach was realized. Evaluation and comparison of the results obtained with those published in the mentioned work were carried out.

Keywords: metaheuristics, model building algorithms, multi-scale search, Macro-H