SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHINKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3855

PRILAGODLJIVI MEMETIČKI ALGORITMI

Marko Lukić

Zagreb, lipanj 2015.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 8. ožujka 2015.

ZAVRŠNI ZADATAK br. 3855

Pristupnik: **Marko Lukić (0036469568)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Prilagodljivi memetički algoritmi**

Opis zadatka:

Opisati način rada i motivaciju za korištenje memetičih algoritama. Pronaći, odabrati i opisati genetski algoritam te algoritme lokalne pretrage koji će se koristiti. Opisati načine prilagodbe memetičkih algoritama i ostvariti odabrane algoritme u programskom obliku. Primijeniti algoritme na ispitnim primjerima i zabilježiti rezultate. Usporediti učinkovitost primijenjenih algoritama te usporediti s rezultatima postojećih optimizacijskih algoritama. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 13. ožujka 2015.
Rok za predaju rada: 12. lipnja 2015.

Mentor:

Izv. prof. dr. sc. Domagoj Jakobović

Predsjednik odbora za završni rad modula:

Prof. dr. sc. Siniša Srbljić

Dielovođa:

Doc. dr. sc. Tomislav Hrkać

Zahvaljujem mentoru prof. dr. sc. Domagoju Jakoboviću
na vođenju te zajedno s obitelji i prijateljima
na podršci, pomoći i motivaciji tokom rada.

Sadržaj

1. Uvod

Tema ovog rada, prilagodljivi memetički algoritmi, razrađuje se u tri dijela.

Prvi dio govori o tome što su, kako je došlo do njih i zašto su potrebni. Pobliže se opisuju dva sastavna dijela algoritama: algoritmi lokalne pretrage i načini na koji se algoritmi prilagođuju. Za svaki od dijelova, osim onog što je programski implementirano u ovom radu spominju se i druge potencijalne opcije.

Drugi dio govori o programskoj implementaciji prilagodljivih memetičkih algoritama. Prvo se opisuje okruženje u kojem se nalazi – ECF. Zatim se na apstraktan način predstavlja struktura cijele implementacije: koje se sve komponente koriste, čemu služe i na koji način su međusobno povezane. Slijedeće, prikazane su i konkretne implementacije pojedinih komponenti te je objašnjen njihov način rada. Konačno, spominje se kako će izgledati buduće korištenje implementacije i kako ona može biti proširena uz savjete o ispravnom rukovanju.

Treći dio govori o ispitivanju programske implementacije i rezultatima tog ispitivanja. Opisuje se način ispitivanja tj. koji se ispitni primjeri koriste, s kojim se drugim algoritmima uspoređuje i što se više, a što manje gleda kod rezultata. Svi rezultati su predstavljeni slikovno ili tablično uz objašnjenje sadržaja prikaza. Rezultati se naposljetku raspravljaju te se navode mogući zaključci koji se iz njih mogu izvući.

Rad završava kratkim zaključkom u kojem se navode mišljenja autora o prilagodljivim memetičkim algoritmima kao i o cjelokupnom radu.

2. Prilagodljivi memetički algoritmi

2.1. Optimizacija i kako je došlo do prilagodljivih memetičkih algoritama

Mnoštvo današnjih problema moguće je predstaviti u obliku funkcije. Rješenje takvih problema bi onda bio upravo globalni optimum (minimum ili maksimum) takve funkcije. Za posao pronalaska optimuma zaduženi su optimizacijski algoritmi implementirani na računalu. Tradicionalno, takvi algoritmi su determinističke heuristike i specijalizirani su za učinkovit pronalazak lokalnog optimuma s tim da je svaki algoritam nešto bolji za određenu vrstu funkcije. Ako funkcija ima samo jedan optimum, problem je riješen, no za funkcije s više optimuma takvi algoritmi često zapinju u lokalnim optimumima i potrebni su algoritmi sposobni za pronalazak globalnog optimuma.

Genetski algoritmi su moderni prirodom inspirirani stohastički algoritmi koji služe za pronalazak globalnog optimuma sa relativno dobrom uspješnošću. Koriste populaciju nasumično odabranih točaka (jedinki) unutar prostora funkcije i pokušavaju ih genetskim operatorima što više približiti optimumu. No, za razliku od algoritama lokalne pretrage oni ne koriste nikakvo znanje o lokalnom izgledu krajolika funkcije pa se to mora nadoknaditi podešavanjem parametara algoritma ili korištenjem specijaliziranih operatora. Unatoč tome, iz tog razloga genetski algoritmi i dalje sporo konvergiraju.

Slijedeći korak u optimizaciji je kombiniranje algoritama lokalne pretrage i genetskih algoritama – memetički algoritmi. Na jedinke populacije primjenjuje se algoritam lokalne pretrage pa tako genetski algoritam radi s lokalnim optimumima umjesto nasumičnih točaka. Na taj način je pretraga usmjerena i ujedno učinkovitija, konvergira uz manje provjera vrijednosti funkcije. Problem sad predstavlja odabir algoritma lokalne pretrage. Svaki je dobar u svojoj domeni i potrebno je od strane korisnika prepoznati u koju domenu spada funkcija koja se pretražuje. Ako je to nemoguće, bilo bi potrebno isprobavati sve dostupne algoritme što ponovno oduzima učinkovitost.

Cilj prilagodljivih memetičkih algoritama je riješiti oba problema: umanjiti posao koji korisnik mora raditi te učinkovito isprobati i odabrati algoritam lokalne pretrage.

Kod njih se za vrijeme rada algoritma koriste svi dostupni algoritmi lokalne pretrage, a posebna komponenta je zadužena za odluku koji će biti odabrani i kako će se koristiti.

2.2. Algoritmi lokalne pretrage

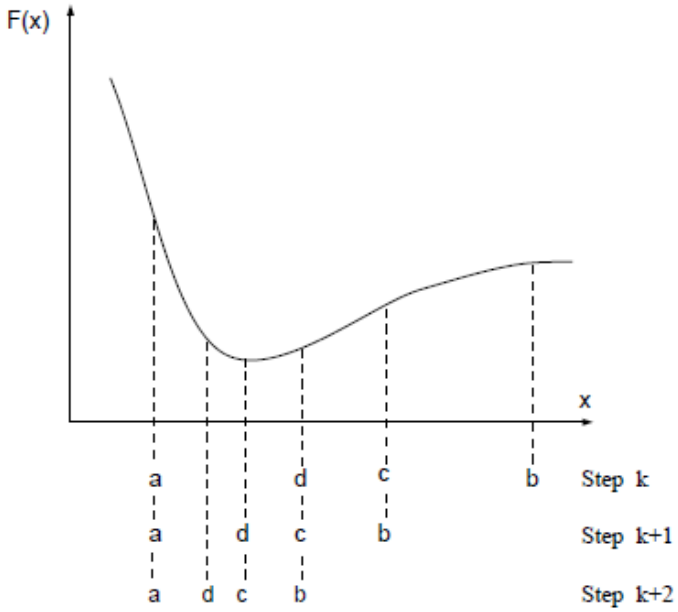
Algoritme lokalne pretrage možemo dijeliti na jednodimenzijske i višedimenzijske. Jednodimenzijski vrše optimizaciju nad samo jednom varijablom dok višedimenzijski pokušavaju optimizirati sve varijable zajedno. Primjena jednodimenzijskih algoritama je najbolja kod funkcija koje imaju samo jedan parametar ili kad su parametri funkcije međusobno neovisni. Kako su takvi algoritmi iznimno učinkoviti u tome što rade, mogu se upotrijebiti i u ostalim slučajevima kada nam je potrebna samo brza konvergencija, a ne i maksimalna točnost konačnog rješenja. Takav slučaj su između ostalima i memetički algoritmi.

Jednodimenzijski algoritmi se dalje mogu dijeliti na istovremene i sekvencijske metode [2]. Istovremene metode jednostavno unutar nekog intervala ispituju vrijednosti funkcije za točke koje su međusobno jednako udaljene. Svako ispitivanje je neovisno od ostalih pa se mogu izvesti istovremeno. Povećanjem broja točaka tj. smanjivanjem razmaka između dvije točke povećava se preciznost rezultata. Budući da su smanjenje broja ispitivanja i točnost rješenja proturječni ciljevi, potrebno je napraviti i optimizirani kompromis. Sekvencijske metode su kompleksnije jer mogu zadržati znanje skupljeno o prethodno ispitanim točkama za donošenje odluke o točki koja će se slijedeća ispitati. Također, pretraživanje korak po korak je prikladnije za današnja računala od paralelnog.

Jedna skupina sekvencijskih metoda specijalizirana je za ograđivanje optimuma ili, drugim riječima, pronalazak intervala u kojem se optimum nalazi. Takvi algoritmi počinju od neka točke i kreću se u smjeru u kojem ima napretka korakom eksponencijalno rastuće duljine. Kad napretka više nema, optimum je ograđen krajevima posljednja dva koraka.

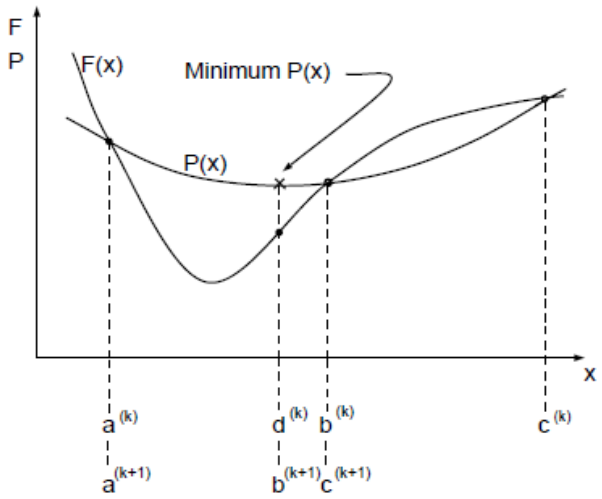
Druga skupina se može nadovezati na prvu jer traže optimum rekurzivnom podjelom intervala. Takvi algoritmi završavaju rad kad je duljina dobivenog

intervala jednaka zadanoj preciznosti. Primjeri algoritama su Fibonaccijeva metoda i Zlatni rez.



Slika 1. Fibonaccijeva metoda [2]

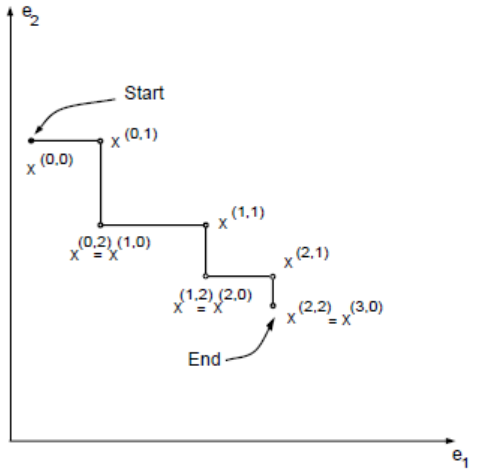
Treća skupina su interpolacijske metode. One koriste točno određene točke unutar intervala, pripadajuće vrijednosti funkcije i/ili vrijednosti derivacija funkcije za interpolaciju točke optimuma. Neke od interpolacija mogu se i iterativno ponavljati. O dostupnim informacijama o funkciji i njenim derivacijama ovisi koja se interpolacija može koristiti, najčešće su to linearna, kvadratna i kubna. Primjeri algoritama su Regula Falsi iteracija, Newton-Raphson iteracija, Lagrangeova interpolacija i Hermiteova interpolacija.



Slika 2. Lagrangeova kvadratna interpolacija [2]

Višedimenzijski algoritmi dijele se na direktne, gradijentne i Newtonove [2]. Direktni funkcioniraju ispitivanjem vrijednosti funkcije u određenim točkama kako bi se pomoću heuristika odredio smjer optimuma, gradijentni koriste prve parcijalne derivacije, a Newtonove i druge parcijalne derivacije. Za potrebe ovog rada razmatraju se samo direktni algoritmi.

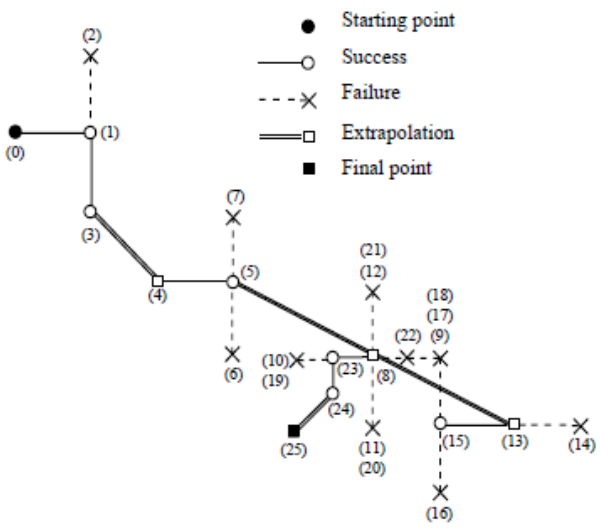
Najjednostavniji primjer je koordinatna strategija [2]. Ona koristi neki jednodimenzijski algoritam i ponavlja ga za svaku varijablu tj. argument funkcije. Do problema dolazi kad se treba kretati u smjeru koji nije kolinearan s koordinatnim osima pa napredak bude stepeničasti. U takvim slučajevima koordinatna strategija jako sporo konvergira.



Numbering	Iteration index	Direction index	Variable values	
	k	i	x_1	x_2
(0)	0	0	0	9
(1)	0	1	3	9
(2)	0	2	3	5
		carried over		
(2)	1	0	3	5
(3)	1	1	7	5
(4)	1	2	7	3
		carried over		
(4)	2	0	7	3
(5)	2	1	9	3
(6)	2	2	9	2
		carried over		
(6)	3	0	9	2

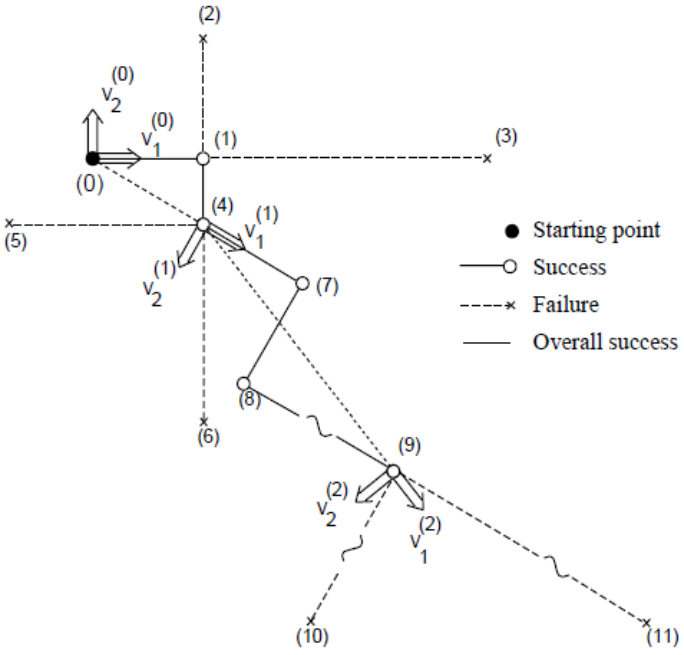
Slika 3. Koordinatna strategija [2]

Hooke-Jeeves strategija [2] proširuje koordinatnu tako što se može kretati i u smjeru napretka. Ovaj algoritam za svaku varijablu ispituje korak u oba smjera i premješta se u najbolju između trenutne i dvije nove točke. Nakon obilaska svih varijabli radi novi korak u smjeru od početne točke prema novoj, poboljšanoj točki. Na taj je način omogućeno dijagonalno kretanje. Dok god je novi smjer dobar algoritam će se pobrinuti da se njime kreće korakom eksponencijalno rastuće duljine. Preciznost algoritam postiže smanjivanjem koraka po osima kad s trenutnim korakom više nema napretka.



Slika 4. Hooke-Jeeves strategija [2]

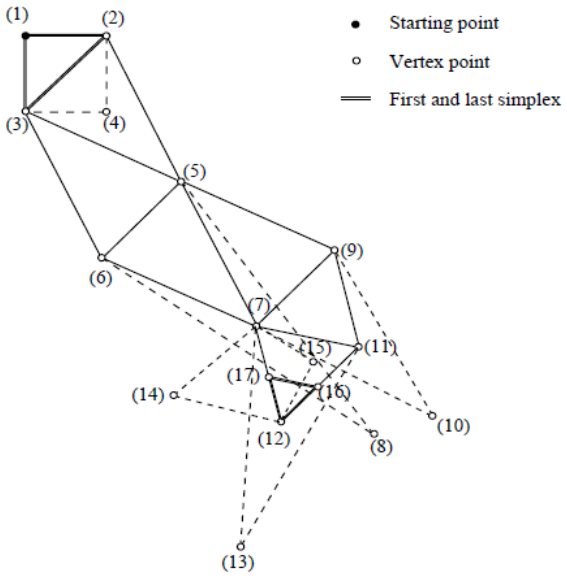
Rosenbrockova strategija [2] radi jedan korak više i uz korak u smjeru napretka ortogonalizacijom rotira koordinatne osi sustava tako da je prva os kolinearna s tim smjerom. U slijedećoj se iteraciji tako prvo ispituje najbolji smjer.



Slika 5. Rosenbrockova strategija [2]

Davies-Swan-Campey strategija (DSC) [2] je kombinacija prethodne dvije. Umjesto da rotira sve osi, ona primjenjuje ortogonalizaciju samo na smjerove u kojima je nakon izvođenja jednodimenzijske pretrage napredak bio veći od nekog minimalnog. Da se ortogonalizacija primijenila na sve smjerove, oni u kojima nije bilo napretka bi bili izgubljeni i tako neiskoristivi u svim budućim iteracijama.

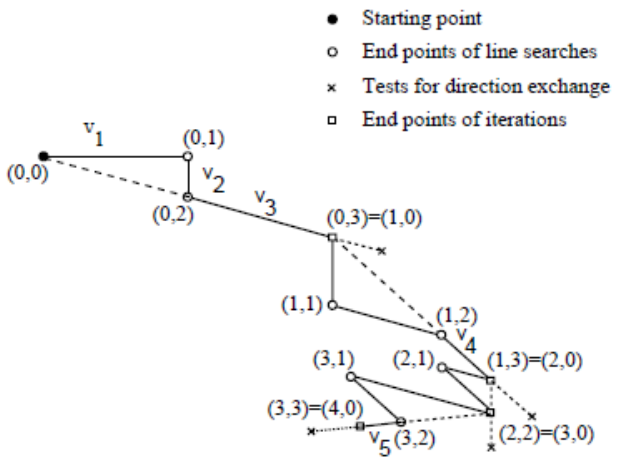
Potpuno drugačiji pristup predstavlja Nelder-Mead simpleks strategija [2]. Ne koristi nikakvu koordinatnu strategiju nego gradi poliedar s brojem vrhova za jedan većim od broja dimenzija. Pomoću njega može istovremeno vidjeti razlike u vrijednosti funkcije u svim smjerovima bez dodatnih ispitivanja. Dalje koristi zrcaljenje najlošije točke u odnosu na sve ostale kako bi se cijeli poliedar korak po korak kretao u smjeru napretka. Da bi se kretanje prilagodilo veličini napretka, moguće je da se nova točka nakon zrcaljenja dalje proširi ili skupi u smjeru kretanja. Ako nema nikakvog napretka cijeli poliedar se skuplja prema najboljem vrhu do neke granice.



Slika 6. Simpleks [2]

Boxova strategija, zvana kompleks [2], je proširenje simpleks strategije time što uzima u obzir granice prostora kojeg pretražuje.

Powellova strategija [4] je više nalik prethodnim algoritmima. Naime, ona se početno ponaša kao koordinatna strategija koristeći smjerove kolinearne koordinatnim osima, a zatim poput ostalih strategija računa smjer najboljeg napretka. Nakon pretrage u tom smjeru, jedinični vektor smjera dodaje kao novi smjer koji se pretražuje, a uklanja prvi smjer koji se trenutno pretraživao. Cijeli proces može se zamisliti kao red smjerova gdje se u svakoj iteraciji novi smjer dodaje na kraj i jedan stari miče s početka.



Slika 7. Powellova strategija [2]

2.3. Načini prilagodbe algoritama

Načini prilagodbe mogu se dijeliti na više različitih pristupa. Tip prilagodbe može biti statički ili adaptivni [1]. Statičke prilagodbe su one koje ne koriste nikakvu povratnu informaciju tokom izvođenja programa. Adaptivne se dalje mogu dijeliti na kvalitativne i kvantitativne [1]. Kvalitativne će dopustiti algoritmu lokalne pretrage da nastavi s radom dok god ima poboljšanja, bez obzira na veličinu poboljšanja. S druge strane, kvantitativne gledaju koliko je točno velik napredak koji su napravili svi algoritmi i odabiru one koji su najbolji.

Druga moguća podjela je prema razini prilagodbe: vanjska, lokalna i globalna [1]. Vanjska razina je ekvivalentna statičkom tipu prilagodbe. Na lokalnoj razini, za odabir algoritma lokalne pretrage koristi se neko trenutno znanje dok se na globalnoj koristi svo znanje skupljeno od početka izvođenja. Lokalne prilagodbe mogu se proširiti na globalne pamćenjem znanja kojeg koriste.

Treća podjela, koja će se koristiti i u nastavku teksta, je prema vrsti hiperheuristike. Hiperheuristika je općenito heuristika s višim stupnjem apstrakcije od metaheuristika koje se primjenjuju direktno na problem. Ona određuje koje od njih će se u kojem trenutku koristiti. Ovdje konkretno je to algoritam koji na neki način bira koji algoritam lokalne pretrage će se primijeniti i na koji način. Hiperheuristika može biti nasumična, pohlepna (eng. *greedy*) ili koristiti funkciju odabira [1]. Nasumične hiperheuristike se većinom oslanjaju na stohastički odabir

algoritma lokalne pretrage. Pohlepne kod svakog odabira isprobavaju sve dostupne algoritme i odabiru onaj koji je zabilježio najveći napredak. Funkcija odabira ocjenjuje rad algoritama tokom izvođenja i tako stvara bazu znanja koje hiperheuristika može koristiti da bi donijela odluku.

Najjednostavnija nasumična hiperheuristika je *simplerandom* [3] koja u svakoj iteraciji odabere nasumični algoritam iz skupa svih dostupnih. *Randomdescent* [3] slično odabire nasumični algoritam, ali ga nastavlja koristiti dok god ima napretka. Treća varijanta je algoritam *randompermdescent* [3] koji na početku izvođenja programa stvara nasumičnu permutaciju algoritama i nakon toga ih samo odabire nastalim redoslijedom s cikličkim ponavljanjem.

Greedy hiperheuristike [3] su međusobno vrlo slične i mogu se svesti na samo jednu s manjim preinakama. U svakoj iteraciji odabire algoritam koji je ispitivanjem pokazan kao najuspješnijim. Dalje može odabrati hoće li primijeniti isti algoritam dok god ima napretka ili ne te hoće li uvijek napraviti ispitivanje na svakoj jedinci genetskog algoritma ili na jednom pa primijeniti rezultat na sve ostale kako bi se uvelike smanjila složenost.

Hiperheuristike koje koriste funkciju odabira imaju najveći broj mogućnosti pa tako i najveću raznovrsnost. Za početak, nije definirano kakva se funkcija mora koristiti pa se mogu koristiti sve od vrlo jednostavnih do vrlo složenih. Zatim, nakon skupljanja podataka o algoritmima moguće je izvršiti odabir na način sličan operatoru selekcije kod genetskih algoritama.

Često se za funkciju odabira koristi više manjih funkcija od kojih svaka vodi računa o različitom podatku o algoritmu. Npr. jedna funkcija koja prati promjenu kvalitete jedinke na koju je primijenjen algoritam, druga koja prati koliko je resursa potrošeno na izvođenje algoritma mjerenjem vremena, broja ispitivanja vrijednosti funkcije koja se optimizira i/ili broja provjera granica prostora koji se pretražuje i treća koja potiče raznovrsnost odabira praćenjem frekvencije odabira pojedinih algoritama.

Straightchoice [3] algoritam jednostavno svaki put odabire algoritam kojem je funkcija dala najveću ukupnu ocjenu. *Rankedchoice* [3] odabire najboljih nekoliko i zatim nad smanjenim izborom izvodi *greedy* algoritam. Slično, *decompchoice* [3] odabire jedan algoritam s najboljom ukupnom ucjenom i po jedan najbolji prema ocjeni svake od pod-funkcija koja se koriste te na njih delegira *greedy* algoritmu.

Roulettechoice [3] kod odabira generira nasumičan broj na intervalu od nula do ukupne ocjene svih algoritama, a svaki algoritam je predstavljen jednim podintervalom širine koja odgovara pripadajućoj ocjeni. Tako svaki algoritam ima odgovarajuću vjerojatnost da bude odabran.

Tablica 1. Popis hiperheuristika i kratki opisi

NAZIV HIPERHEURISTIKE	NAČIN ODABIRA LOKALNE PRETRAGE
Simplerandom	Nasumično u svakoj iteraciji
Randomdescent	Nasumično kad trenutna više ne napreduje
Randompermdescent	Ciklično iz nasumično permutiranog popisa
Greedy	Odabir najbolje ispitivanjem jednog koraka za svaku
Straightchoice	Odabir najbolje ocijenjene
Rankedchoice	*Greedy* odabir između nekoliko nabolje ocjenjenih
Decompchoice	*Greedy* odabir između najbolje ocjenjenih u svakoj kategoriji
Roulettechoice	*Roulette wheel* odabir prema udjelu ocjena

3. O programskoj implementaciji

3.1. O ECF-u

ECF (*Evolutionary Computation Framework)* je programski okvir pisan u jeziku C++ specijaliziran za optimizaciju pomoću evolucijskog računanja. Ima implementirane brojne evolucijske algoritme uključujući determinističke algoritme lokalne pretrage, stohastičke genetske algoritme, memetičke algoritme, genetsko programiranje itd.

U ECF je jednostavno dodati novi algoritam, potrebno je samo stvoriti razred koji nasljeđuje „*Algorithm*“ i u njemu opisati željeno ponašanje. Unutar *main* funkcije potrebno je javiti sustavu da postoji novi algoritam i u konfiguracijskoj datoteci koja sadrži sve parametre izvođenja programa upisati njegov naziv kako bi mogao biti korišten.

Na taj način je izvedena i programska implementacija prilagodljivih memetičkih algoritama u ovom radu, kao jedan algoritam koji u sebi sadrži više manjih od kojih svaki ima svoj dio odgovornosti tj. predstavlja jednu od komponenti.

3.2. Apstraktna struktura implementacije

3.2.1. Baza implementacije

Algoritam u ECF-u se nužno sastoji od dva osnovna dijela: inicijalizacija i glavna petlja. Sustav ih jednog za drugim pokreće i na taj način algoritam može funkcionirati.

Inicijalizacija se vrši kroz konstruktor i metodu *initialize.* U konstruktoru se definira ime algoritma, konkretno *AdaptiveMemetic,* kako ga sustav može lako prepoznati i odabrati te varijable koje su neovisne o stanju sustava. Ovisni dijelovi dobivaju informacije iz stanja u metodi *initialize*. U njoj se postavljaju vrijednosti poput granica prostora koji se pretražuje i preciznosti koja se očekuje od rezultata te provjerava vrsta genotipa koji se optimizira, u ovom slučaju to mora biti vektor

realnih brojeva. Svaki algoritam može i imati svoje parametre koji se zadaju u parametarskoj datoteci, oni se također čitaju i postavljaju ovom koraku. Posebni parametri koje se koriste biti će navedeni u nastavku teksta uz komponente na koje se odnose.

Gotovo svaki optimizacijski algoritam funkcionira iterativno što se implementira njegovom glavnom petljom. Jedna iteracija opisana je u metodi *advanceAlgorithm* i nju sustav poziva dok god nije ispunjen neki kriterij zaustavljanja. U ovom algoritmu ona se sastoji od nekoliko dijelova: za svaku jedinku izvršava se odabir algoritma lokalne pretrage, primjena jedne njegove iteracije i jedne iteracije genetskog algoritma na tu jedinku.

Klasične jedinke genetskog algoritma koje se koriste u ECF-u za potrebe prilagodljivih memetičkih algoritama zamotane su u razred koji predstavlja proširenu jedinku. Takva jedinka sadrži dodatne podatke potrebne za rad, prvenstveno njeno stanje i pridruženi algoritam lokalne pretrage. Tri stanja u kojima ona može biti su početno, stanje optimizacije i konvergirano. Početno stanje sprječava da se jedinka koristi prije nego što joj je prvi put dodijeljen algoritam lokalne pretrage i primijenjena barem jedna njegova iteracija. Ako je algoritmu lokalne pretrage dozvoljeno da radi kroz više iteracija glavnog algoritma jedinka je u stanju optimizacije. To znači da se ona ne smije mijenjati niti joj se dodijeliti novi algoritam, ali se može koristiti kao izvor genetskog materijala. Kada je završena lokalna pretraga tada je jedinka u konvergiranom stanju i dostupna da ju genetski algoritam mijenja. Čak i nakon konvergencije, jedinka pamti koji joj je algoritam posljednje dodijeljen.

3.2.2. Algoritmi lokalne pretrage

Algoritmi lokalne pretrage, kao i glavni algoritam, raspoznaju se po svom imenu i izvode jednu iteraciju svoje glavne petlje pomoću metode *advance*. Za procjenu dobrote jedinke koriste metodu *evaluate* roditeljskog (glavnog) algoritma koja se određuje na razini sustava. Kada je rad algoritma završen i daljnja iteracija nema koristi, zastavica *finished* postavlja se u istinito stanje. Tokom rada algoritam piše izvješće imenovano imenom algoritma. Ono sadrži količinu napretka u dobroti jedinke, broj procjena i broj iteracija od početka izvođenja do sadašnjeg trenutka, a

završava kad je postignuta konvergencija. Budući da se algoritam može istovremeno koristiti za više jedinki i više puta tokom izvođenja programa, on mora znati točno do kud je došao i s kojom jedinom. Kako ne bi morao sve to pamtiti on ima mogućnost stvaranja svoje kopije koja se nalazi u početnom stanju i ta kopija se zatim pridruži jedinki. Svi dostupni algoritmi lokalne pretrage čuvaju se u vektoru ispunjenom tokom inicijalizacije glavnog algoritma i otamo se mogu birati.

3.2.3. Selektor

Komponenta, zvana *selektor*, koja je zadužena za odabir i dodjelu algoritama lokalne pretrage ujedno i određuje način prilagodbe memetičkog algoritma. Selektor ima jednu glavnu metodu kojom to izvodi i u njoj su implementirani svi koraci i odluke koje treba izvršiti. Kod inicijalizacije osim dostupnih algoritama lokalne pretraga dobiva i bazu podataka u kojoj su zapisane trenutne ocjene svih tih algoritama. Ona služi kod odabira nekih od selektora i opisivati će se u nastavku teksta. Budući da postoji više različitih implementacija selektora, tokom inicijalizacije glavnog algoritma odlučuje se koji od njih će se koristiti u glavnoj petlji.

3.2.4. Genetski algoritam

Genetski algoritam koji se koristi je vrlo jednostavan. On samo mora paziti da ne izmijeni trenutno najbolju jedinku i da su jedinke s kojima radi u odgovarajućem stanju. Radi prvenstveno s onima u konvergiranom stanju i njih vidi kao receptore genetskog materijala, dok sve ostale koje su u stanju optimizacije predstavljaju potencijalne donatore. Receptor se križa s nasumično odabranim donatorom i zatim mutira. Receptor tako postaje novo dobivena jedinka, a donator ostaje u izvornom stanju. Uz odgovarajuće postavke parametara glavnog algoritma moguće je korištenje i nasljeđivanja algoritma lokalne pretrage. Tada jedinka koja nastaje kao rezultat križanja nasljeđuje algoritam pridijeljen roditelju koji je u tom trenutku bolji, ili nasumičan ako su jednaki. Ovako se proces odabira algoritma

lokalne pretrage neće izvoditi više od jednom jer jedinke nikad ne ostaju u konvergiranom stanju.

3.2.5. Ocjenjivanje algoritama lokalne pretrage

Proces ocjenjivanja algoritama lokalne pretrage omogućuju dvije komponente: povijest izvođenja i funkcija ocjenjivanja. Naime, kad svaka proširena jedinka nastaje, ona dobiva svoje mjesto za unos podataka u bazu koja predstavlja zapis povijesti izvođenja. U trenutku kad jedinka konvergira ona od algoritma lokalne pretrage preuzima stvoreno izvješće i predaje ga bazi podataka. Potreba za odvojenim unosima za svaku jedinku proizlazi iz zahtjeva da se prate uzastopne primjene algoritama na pojedinoj jedinci. Osim navedenih izvješća, jedinke javljaju bazi trenutak kad im je dodijeljen neki algoritam. Pomoću toga u bazi se može evidentirati broj generacija od posljednjeg odabira nekog algoritma koji se namjesti na nulu svaki put kad stigne jedna takva dojava.

Funkcija ocjenjivanja direktno je povezana s povijesti izvođenja. U svakoj iteraciji glavnog algoritma baza podataka o izvođenju šalje sve svoje podatke funkciji ocjenjivanja. Ona odvaja nove podatke od prethodno obrađenih i na osnovu njih u vlastitoj bazi osvježava ocjene algoritama. Prethodno postignuti napredak algoritma umanjuje se skaliranjem i dodaje se novi. Slično funkcionira i ocjenjivanje ukupnog napretka dvaju uzastopno primijenjenih algoritama. Na taj se način stavlja naglasak na nedavne rezultate korištenja algoritma, ali bez gubitka podataka o globalnom korištenju. Kako je moguće da se na ovaj način uvijek preferiraju isti algoritmi, a drugi zanemaruju, treći kriterij ocjenjivanja je broj generacija od posljednjeg odabira koji se direktno preslikava iz baze o izvođenju. Ukupna ocjena je zbroj triju kriterija uz to da je svaki pomnožen faktorom važnosti. Tako je nedavni zasebni uspjeh znatno važniji od uzastopnog, dok se broj generacija od posljednjeg odabira mora množiti najvećim faktorom jer vrijednosti druga dva kriterija znaju biti nekoliko redova veće.

Selektori koji koriste funkciju ocjenjivanja čitaju iz njene baze podataka dohvaćanjem ukupne ocjene ili njenih dijelova koristeći ime algoritma kao ključ. Može se vidjeti kako je cijeli proces kružni baziran na povratnoj informaciji.

Selektor početno nasumce odabire algoritme lokalne pretrage i pridružuje ih jedinkama. Algoritmi stvaraju izvješća koja jedinke upisuju u bazu podataka o povijesti izvođenja. Funkcija ocjenjivanja formira ocjena prema sakupljenim podacima i nakon određenog broja iteracija glavnog algoritma selektori ih mogu koristiti kako bi donijeli informirane odluke o odabiru.

3.3. Konkretne implementacije komponenti

Od algoritama navedenih u prethodnom poglavlju za potrebe prilagodljivih memetičkih algoritama implementirani su zlatni rez i Lagrangeova kvadratna interpolacija koja ne koristi derivacije od algoritama koji rade s jednom dimenzijom te Hooke-Jeeves, Davies-Swan-Campey, simpleks i Powellova strategija od višedimenzijskih. Zlatni rez i interpolacija koriste se u koordinatnoj strategiji, a mogu se koristiti i kao dio nekog drugog algoritma.

Tablica 2. Popis implementiranih algoritama lokalne pretrage

NAZIV ALGORITMA
Koordinatna strategija koja koristi zlatni rez
Koordinatna strategija koja koristi Lagrangeovu kvadratnu interpolaciju
Hooke-Jeeves strategija
Nelder-Mead simpleks strategija
Davies-Swan-Campey strategija
Powellova strategija

Svaki selektor ima mogućnost da radi na tri različita načina, ovisno o zadanim parametrima. Prvi parametar određuje hoće li algoritmu lokalne pretrage biti dopušteno da nastavi s radom dok god ima napretka ili će se samo koristiti jedna njegova iteracija. U slučaju da se koristi druga varijanta genetski algoritam će u svakoj glavnoj iteraciji moći raditi sa svim jedinkama, a inače mora čekati da algoritam završni no jedinka će biti kvalitetnija. Drugi parametar određuje hoće li

se selekcija izvoditi za svaku jedinku posebno ili jednom za cijelu populaciju. Prva varijanta donosi točniju odluku, ali uz cijenu veće složenosti. Ako je preciznost odabira manje bitna ili je odluka manje ovisna o stanju pojedine jedinke, jednostavnije je svesti odabir na cijelu populaciju.

Pomoću parametra koji određuje hoće li se isti algoritam lokalne pretrage koristiti dok god ima napretka ili u samo jednoj iteraciji, sad su *simplerandom* i *randomdescent* jedan selektor. *Randompermdescent* je opće nazvan *randomperm* jer je određen samo time što koristi iteraciju nad nasumičnom permutacijom. *Greedy* selektor i dalje ispituje samo prvi korak kod usporedbe i onda ovisno o parametru može pustiti da algoritam dalje radi. Moglo bi se reći da je ovo relativno netočna procjena rada cijelog algoritma jer je nekima potrebno više iteracija da ispune svoju specifičnu svrhu. Budući da je ovo računski najsloženiji selektor može se koristiti za odabrane podskupe algoritama lokalne pretrage ili primijeniti jedan rezultat na cijelu populaciju.

Svi selektori koji koriste funkciju ocjenjivanja u fazi prikupljanja podataka prepuštaju odluku *simplerandom*-u. On je namješten da dopušta samo jednu iteraciju algoritama i bira poseban algoritam za svaku jedinku kako bi se povećala raznovrsnost uzastopnog korištenja. Nakon što su podaci prikupljeni koriste se *straightchoice* i *roulettechoice* kao samostalni algoritmi te *rankedchoice* i *decompchoice* koji odabiru podskup algoritama lokalne pretrage i konačnu odluku ostavljaju *greedy* selektoru.

Tablice 2. i 3. Popis implementiranih selektora i popis parametara selektora

NAZIV SELEKTORA
Simplerandom
Randomperm
Greedy
Straightchoice
Rankedchoice
Decompchoice
Roulettechoice

PARAMETAR	VRIJEDNOST
Koji selektor	od 1 do 7
Koristiti nasljeđivanje	true ili false
Koristiti spust	true ili false
Izbor za svaku jedinku	true ili false

3.4. Buduće korištenje i proširivost

Na isti način na koji su implementirani svi složeni selektori mogu se za izradu novih selektora koristiti već postojeći u kombinaciji jednih s drugima. Najjednostavniji primjer bi bio hibridni *radnom* algoritam koji ima dvije *simplerandom* instance od kojih jedna radi sa svakom jedinkom i po jednu iteraciju i druga sa cijelom populacijom i pušta da algoritam radi dok god ima napretka. Takav algoritam bi mogao prvo koristiti prvi način rada dok se prostor istražuje velikim nepreciznim koracima, a zatim drugi da se istraženi prostor maksimalno precizno iskoristi.

Kao i sa selektorima, moguće je eksperimentirati s algoritmima lokalne pretrage tako da se više puta koristi isti algoritam koristeći različite parametre. Prije pokretanja petlje glavnog algoritma metodom *addLocalSearchAlg* mogu se dodati vlastiti algoritmi sa željenim argumentima. Postoji i parametar glavnog algoritma koji se postavlja u datoteci i sprječava automatsko ispunjavanje vektora s algoritmima lokalne pretrage tako da se koriste samo naknadno dodani algoritmi. Ako korisnik ne zna koji su parametri nekog algoritma najbolji za primjenu na trenutnoj funkciji i ne želi ih sve posebno isprobavati, može tako dodati svoj algoritam više puta i posao prepustiti funkciji ocjenjivanja i selektoru. Naravno cijena automatskog odabira je višak posla koji se obavlja u fazi prikupljanja podataka te nesigurnost točnosti rezultata budući da je i sam proces prilagodbe stohastički.

Ako se pak želi koristiti samo jedan algoritam lokalne pretrage zajedno s genetskim algoritmom kao kod običnih memetičkih algoritama, potrebno je taj algoritam jedini dodati i koristiti *simplerandom* kao najjednostavniji selektor uz opciju da dopušta cijeli rad algoritma i izvodi selekciju za svaku jedinku. Spremanje podataka u bazu i računanje ocjena je zanemariv udio ukupnog vremena pa je tako postavljan algoritam gotovo jednak običnom memetičkom algoritmu. Nadalje, ako se ne doda niti jedan algoritam lokalne pretrage, glavni algoritam će se ponašati kao jednostavni genetski algoritam.

Algoritmi lokalne pretrage implementirani su tako da rade s osnovnim jedinkama ECF-a što znači da nisu ovisni o okolišu adaptivnih memetičkih algoritama. Zbog toga je svaki od njih moguće ponovno iskoristiti kao komponentu lokalne pretrage u nekom drugom algoritmu ili jednostavno zamotati u razred koji

nasljeđuje *Algorithm* i primijeniti na populaciju. Posljednji slučaj je dobar ako nam su nam dovoljni samo lokalni optimumi, a višestrukim izvođenjem može se gledati i kao *random restart hill climbing* algoritam.

Za korištenje mehanizma nasljeđivanja algoritama lokalne pretrage trebalo bi napomenuti da nije preporučeno korištenje uz selektor koji jedan odabir primjenjuje na cijelu populaciju. Tada bi se zbog nedostatka raznolikosti isti algoritam nastavio koristiti za sve jedinke do kraja izvođenja programa. Općenito je za prvu dodjelu algoritma lokalne pretrage u ovom slučaju nabolje koristiti selektor koji jednoliko raspoređuje i potiče raznovrsnost, npr. *simplerandom* ili *randomperm*.

Kao što se može vidjeti, implementacija prilagodljivih memtičkih algoritama može imati raznolike uloge ovisno o načinu uporabe. Osnovna uloga bi bila skup algoritama lokalne pretrage koji se koristi zajedno s genetskim algoritmom, a čiji odabir i korištenje ovisi o selektoru. Taj skup je moguće i proširivati dodavanjem novih algoritama koji će se dalje ravnopravno koristiti. Druga moguća uloga je kao mehanizam za automatski odabir parametara algoritma za lokalnu pretragu dodavanjem više različitih verzija istog algoritma u prazan skup. Zatim ako skup sadrži samo jedan algoritam, uloga postaje ta običnog memetičkog algoritma. I naposljetku ako je skup prazan, ono što preostaje je jednostavan genetski algoritam.

4. Ispitivanje i rezultati

4.1. Što i kako se ispituje

Optimizacijski algoritmi koji rade s kontinuiranim funkcijama obično se ispituju pomoću skupa raznolikih problema koji zahtijevaju različite kvalitete algoritma. Primjerice, osnovna funkcionalnost se ispituje jednostavnim glatkim funkcijama s jednim optimumom. Gleda se može li algoritam pronaći taj optimum, koliko mu vremena ili evaluacija treba da pronađe taj optimum i koliko je precizan krajnji rezultat. Dalje se krajolik funkcije može proširiti tako da ima više lokalnih optimuma, platoe, doline i slično. Ako algoritam ima ikakvu stohastičku komponentu potrebno ga je više puta pokrenuti i vidjeti koliko rezultati međusobno odstupaju, koliko je često optimum pronađen te koja je vrijednost medijana rezultata.

Jedan takav opće prihvaćeni skup ispitnih funkcija su funkcije COCO (*COmparing Continuous Optimisers*) platforme tj. *Black-Box Optimization Benchmarking* (BBOB) funkcije [5]. Sadrži 24 funkcije bez i 30 funkcija sa šumom. Implementacije svih funkcija već postoje unutar ECF-a i jednostavno ih je iskoristiti za ispitivanje algoritama koji su također u njemu. Sve funkcije su namještene da im se optimum nalazi na intervalu od -5 do 5 s vrijednosti u optimuma 0.

U nastavku teksta bit će prikazani rezultati izvođenja algoritma kroz funkcije bez šuma uz ponavljanje tako da će se koristiti različiti selektori ili parametri poput veličine populacije i maksimalnog broja evaluacija jedinki. Rezultati će biti uspoređeni međusobno kao i s rezultatima drugih algoritama unutar ECF-a.

4.2. Rezultati ispitivanja

4.2.1. Ispitivanje s ponavljanjem kroz sve bešumne funkcije

Slijedeće ispitivanje izvršeno je izvođenjem algoritama kroz sve 24 bešumne funkcije, s 10 ponavljanja za svaku funkciju. Veličina populacije jedinki je 100, broj

dimenzija 10 i ograničenjima izvođenja maksimalno 1000000 (milijun) evaluacija ili pronalazak točnog minimuma (0). Vrijednosti uz algoritme su medijani rezultata za odgovarajuću funkciju te rang algoritma u ispitivanju (1-5).

Tablica 4.1. Rezultati ispitivanja na bešumnim COCO funkcijama

F	***Randomdescent***	**R**	***Greedydescent***	**R**	***Roulettechoice***	**R**
1	**0**	**1**	**0**	**1**	**0**	**1**
2	**0**	**1**	1.58309e-11	4	5.414335e-12	3
3	5.96975	5	4.477415	4	**5.5769585e-12**	**1**
4	6.96471	5	4.47732	4	0.4974795	2
5	**0**	**1**	**0**	**1**	**0**	**1**
6	1.0884435e-10	3	2.261585e-10	4	1.78492e-13	2
7	0.0451597	2	**0.03381225**	**1**	0.04790605	3
8	**5.68435e-14**	**1**	2.00917e-7	4	9.288235e-11	3
9	6.9207e-12	2	0.00810537	4	8.87894e-11	3
10	**0.10945245**	**1**	0.49275	3	6.7002	4
11	0.131108	3	**0.011146525**	**1**	0.0422008	2
12	**5.032643e-13**	**1**	5.34896e-5	2	0.0007142215	3
13	0.0001882566	2	0.00085113	4	0.000192871	3
14	1.919305e-6	2	**3.88052e-7**	**1**	2.277515e-6	4
15	31.3411	5	24.8748	3	**11.442**	**1**
16	0.5431715	4	**0.1167205**	**1**	0.161071	2
17	2.951415	5	1.975305	3	**0.0311117**	**1**
18	7.492265	4	7.46743	3	**0.1894205**	**1**
19	0.4703665	5	0.272882	3	**0.159965**	**1**
20	0.473065	3	0.621807	5	**0.236877**	**1**
21	**0**	**1**	**0**	**1**	1.5880635e-12	2
22	**0**	**1**	8.753895e-12	3	1.955	4
23	0.041764	2	0.06198395	3	0.0639608	4
24	32.83785	5	29.9431	4	**9.926245**	**1**

Tablica 4.2. Rezultati ispitivanja na bešumnim COCO funkcijama

Funkcija	ECF_SST	Rang	ECF_GHJ	Rang
1	4.21295e-10	3	2.84217e-14	2
2	1.0294375e-8	5	2.84217e-14	2
3	1.197555e-7	2	4.24217	3
4	**6.15991e-7**	**1**	2.98488	3
5	**0**	**1**	**0**	**1**
6	0.00039576	5	**1.24898e-13**	**1**
7	2.117185	5	0.3511765	4
8	0.160909	5	1.705305e-13	2
9	4.69924	5	**9.947605e-14**	**1**
10	323.7805	5	0.2194545	2
11	0.3737095	4	6.773525	5
12	1.4607805	5	0.00446698	4
13	3.585965	5	**9.249795e-5**	**1**
14	0.000310397	5	1.95625e-6	3
15	27.8678	4	22.38655	2
16	3.4833	5	0.2840575	3
17	0.3234745	2	2.796005	4
18	1.382235	2	11.90895	5
19	0.295105	4	0.201656	2
20	0.454017	2	0.58094	4
21	2.25581	3	**0**	**1**
22	1.955	4	5.68435e-14	2
23	0.556624	5	**0.04121965**	**1**
24	19.8479	2	24.25735	3

Algoritmi koji su ispitani su redom: prilagodljivi memetički uz *simplerandom* selektor koji dopušta spust (*descent*), *greedy* selektor koji dopušta spust i *rouletteselect* koji dopušta spust te *Steady state tournamet* i memetički algoritam koji koristi Hooke-Jeeves strategiju kao lokalnu pretragu iz ECF-a. Treba napomenuti da svi prilagodljivi memetički algoritmi koriste dopuštanje spusta kako bi se bolje usporedili selektori. Svaki od korištenih selektora predstavlja različitu od tri skupine (nasumični, *greedy,* oni koji koriste ocjene funkcije odabira).

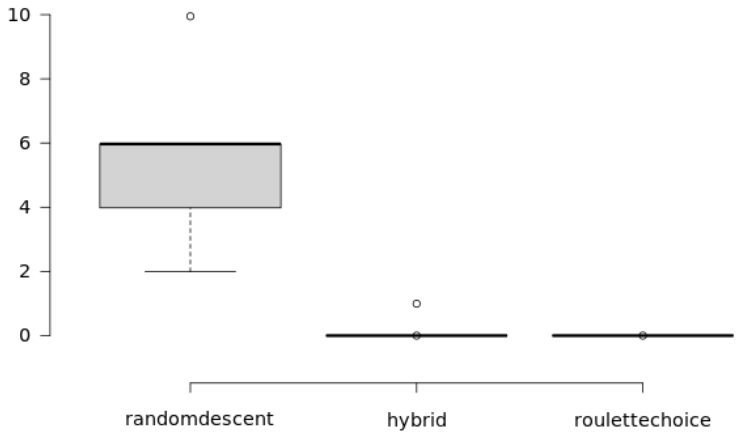
Za bolji prikaz uspješnosti algoritama prikazuju se i njihovi prosječni rangovi.

Tablica 5. Prosječni rangovi algoritama

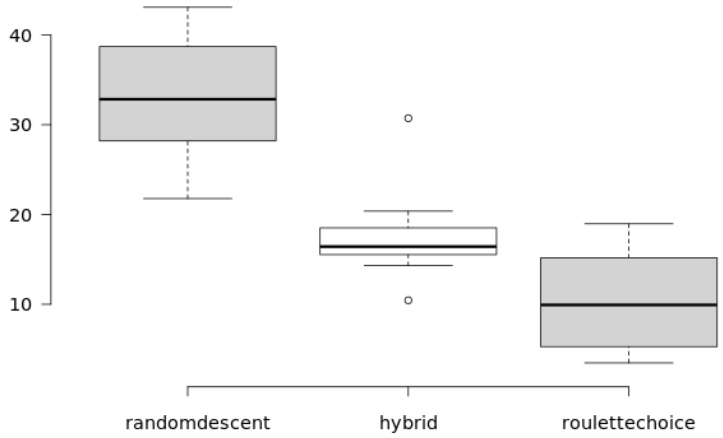
	Randomdesc	**Greedydesc**	**Roulettech**	**ECF_SST**	**ECF_GHJ**
Prosječni rang	2.6667	2.8333	**2.2803**	3.7083	2.5417

4.2.2. Ispitivanje utjecaja perioda bez spusta

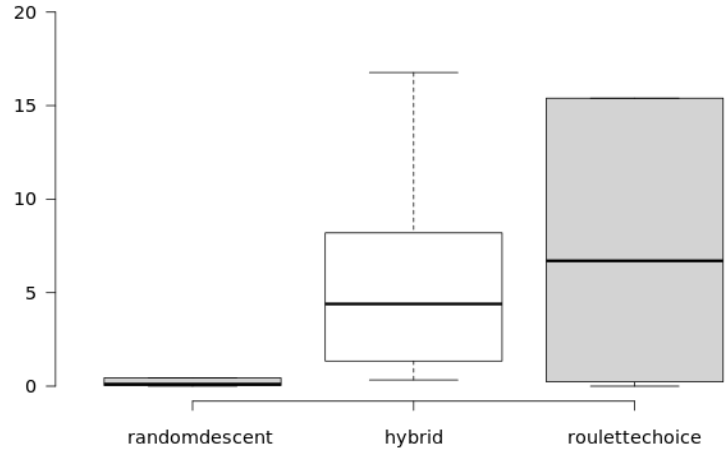
Budući da se osim načina odabira *simplerandom* i *roulettechoice* koji dopuštaju spust razlikuju po tome što *roulettechoice* prvih 100 generacija radi samo prvi korak algoritama lokalne pretrage, slijedeće se prikazuju rezultati usporedbe tih dvaju algoritama s novim selektorom koji uklanja tu razliku. On kombinira dva *simplerandom* selektora: prvi koji 100 generacija ne dopušta spust i drugi koji nakon 100 generacija to dopušta. Tako se može vidjeti koliki utjecaj ima samo korištenje perioda bez spusta u odnosu na proširenje korištenjem kompleksnijeg načina odabira. Ispitivanje se izvršava na funkcijama 3, 10 i 24 zbog velikih razlika u rezultatima *simpleradnom* i *roulettechoice* selektora.



Slika 8. Rezultat ispitivanja funkcijom 3



Slika 9. Rezultat ispitivanja funkcijom 10

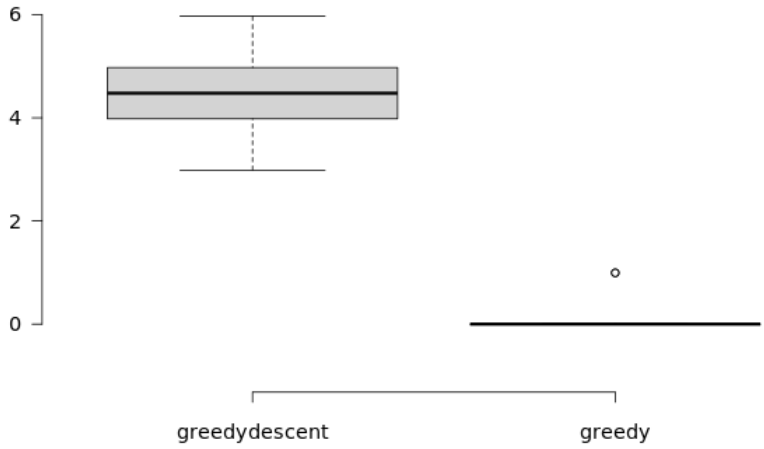


Slika 10. Rezultat ispitivanja funkcijom 24

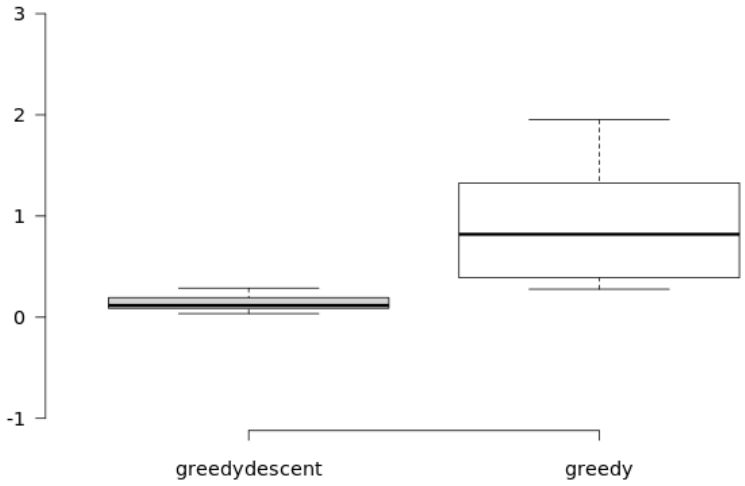
x-os na slikama prikazuje pojedine algoritma, a y-os vrijednosti retultata kroz svih 10 izvođenja algoritma za funkciju. Istaknuti su medijan, minimalna i maksimalna vrijednost te krajevima pravokutnika prvi i treći kvartil.

4.2.3. Ispitivanje utjecaja parametra spusta kod greedy *selektora*

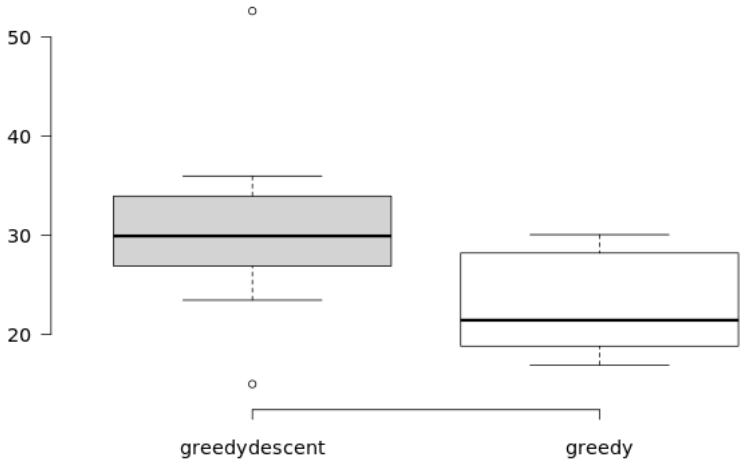
Slično se ispituje *greedy* selektor. On donosi odluke ispitujući kvalitetu prvog koraka algoritma lokalne pretrage i zatim, ovisno o parametru, izvodi taj korak ili pušta algoritmu da radi dok ima napretka. Kao što je prethodno u tekstu navedeno, upitno je koliko je ta procjena točna u drugom slučaju. Zato se sad ispituje hoće li i kako ta razlika utjecati na kvalitetu rezultata na nekim od funkcija gdje je *greedy* selektor imao najbolji ili gotovo najlošiji rezultat.



Slika 11. Rezultat ispitivanja funkcijom 3



Slika 12. Rezultat ispitivanja funkcijom 16

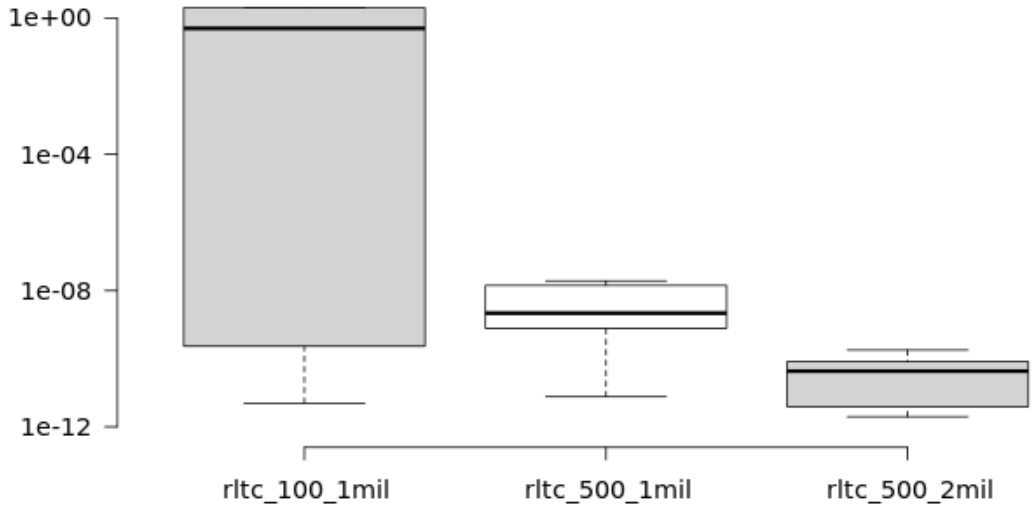


Slika 13. Rezultat ispitivanja funkcijom 24

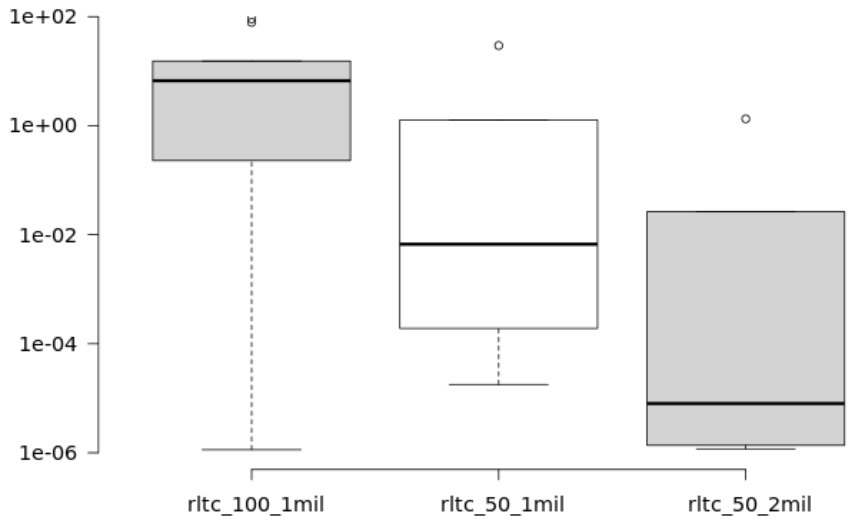
Ostale usporedbe za funkcije u kojima je *greedydescent* bio najbolji izgledaju slično kao za funkciju 16 samo što je razlika u redovima rezultata veća pa bi vizualizacija bila lošija poput one za funkciju 3.

4.2.4. Ispitivanje utjecaja veličine populacije i broja evaluacija

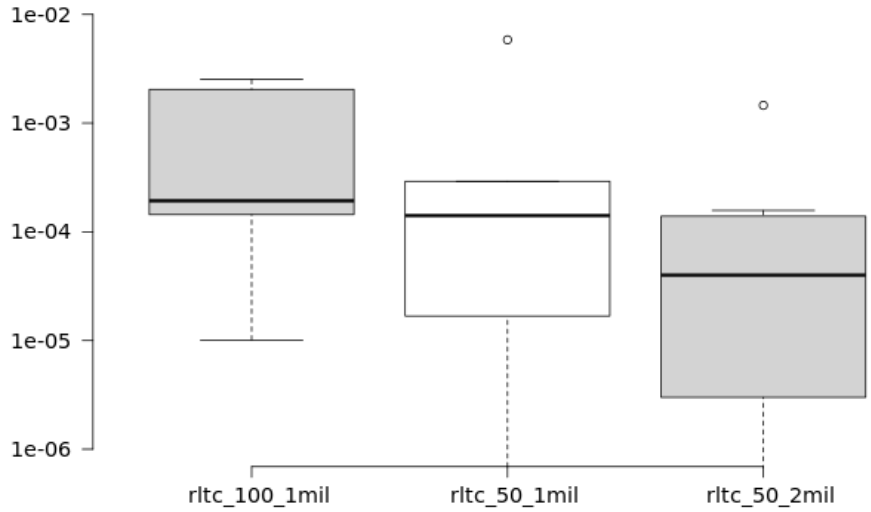
Kako je period skupljanja podataka kod selektora koji koriste ocjene funkcije odabira ograničen brojem generacija, količina podataka koja se skuplja tokom perioda bez spusta i broj lokalnih pretraga bez spusta može se jedino mijenjati pomoću veličine populacije. Povećanjem količine podataka koji se skupljaju uzrokuje točnije donošenje odluka kasnije dok smanjenje potiče veću raznovrsnost budući da je manja razlika u ocjenama algoritama. S većom populacijom broj evaluacija koji se troši u periodu skupljanja podataka je puno veći pa je potrebna promjena ograničenja kako bi se drugi period mogao dovoljno dugo izvoditi. Slijedeći ispitni primjer pokazuju moguća poboljšanja rezultata algoritma koji koristi *roulettechoice* selektor prilagodbom veličine populacije i ograničenja maksimalnog dozvoljenog broja evaluacija



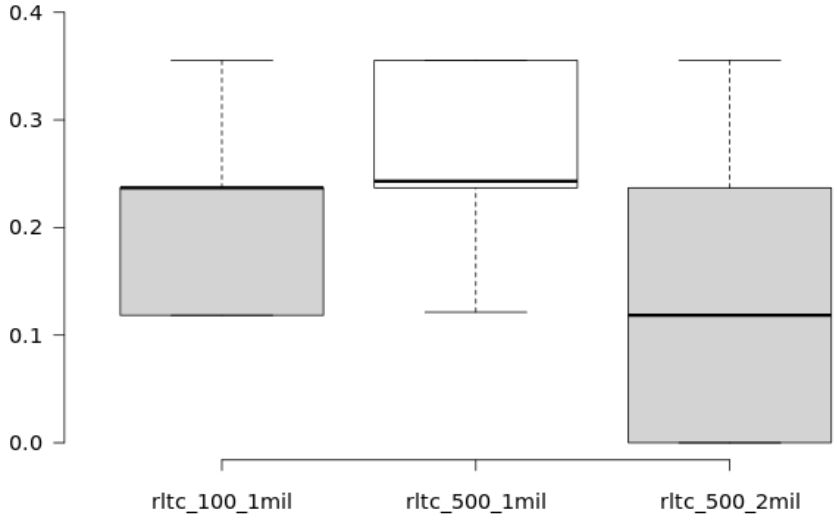
Slika 14. Rezultat ispitivanja funkcijom 4



Slika 15. Rezultat ispitivanja funkcijom 10



Slika 16. Rezultat ispitivanja funkcijom 13



Slika 17. Rezultat ispitivanja funkcijom 20

Na svakoj slici na prvom mjestu stoji *roulettechoice* sa 100 jedinki i ograničenjem od 1000000 (milijun) evaluacija, na drugom s populacijom smanjenom na 50 ili povećanom na 500 i na trećem s ograničenjem povećanim na 2000000 (dva milijuna) evaluacija.

4.3. Diskusija o rezultatima

U prvom dijelu ispitivanja može se vidjeti kako memetički algoritmi daju u prosjeku nešto preciznije rezultate od čistog genetskog algoritma jer se manje oslanjaju na slučajno pogađanje optimuma. Korištenjem deterministčkih algoritama lokalne pretrage pozicije jedinki su finije namještene istraživanjem njihovih okoliša prije nego što su prepuštene genetskim operatorima.

Također, može se vidjeti da bez korištenja kvalitetnog algoritma za prilagodbu memetički algoritmi koji koriste više različitih lokalnih pretraga u prosjeku zaostaju za specijaliziranim memetičkim algoritmom koji koristi samo jedan. Jedan od mogućih načina rješavanja tog problema bi moglo biti da se nakon što slabiji prilagodljivi memetički algoritam završi rad, vidjeti koja od lokalnih pretraga ima najbolji uspjeh te u slijedećem izvođenju koristiti samo nju, po izboru i s različitim parametrima. S druge strane, vidi se kako kvalitetan selektor može nadmašiti specijalizirane memetičke algoritme ako su problemi s kojima radi dovoljno raznoliki da se isplate svi različiti pristupi optimizaciji koje koristi.

Iz drugog dijela je očito da je korištenje samo prvog koraka algoritama lokanih pretraga kroz određeni broj generacija, a zatim prelazak na izvođenje dok ima napretka, dio razlike između potpuno nasumičnog odabira i odabira osnovanog na prikupljenim podacima. Rezultati pokazuju kako je u slučajevima gdje je takvo ponašanje poželjno ipak bolje imati mehanizam koji uči iz njega, a ne da nastavlja s nasumičnim korištenjem. No, kad je to nepoželjno, moguće je učenjem iz podataka doći do pogrešnih zaključaka i izgubiti raznolikost odabira. Kod potpuno nasumičnih selektora zastupljenost algoritama ostaje ista pa pogrešan napredak može biti ispravljen.

Kod *greedy* selektora je nemoguće znati je li bolje koristiti spust ili ne jer to posve ovisi o problemu koji se optimizira. On sam već koristi više evaluacija nego

drugi algoritmi i isprobavanje obje inačice bi bila skupa operacija. Čak i da je poznato koju inačicu treba koristiti, u prosjeku je prikazano da je ovaj pristup lošiji od jednostavnog nasumičnog koji je i puno brži u svom izvođenju.

Naposljetku se može vidjeti kako prilagodljivi memetički algoritmi i dalje imaju vrlo važne vlastite parametre koje korisnik mora ručno namještati. Kao što je kod ispitivanja objašnjeno, veličina populacije i maksimalan broj evaluacija izravno utječu na skupljanje podataka i udio evaluacija koji se koristi u pojedinom periodu algoritma. Kod običnih memetičkih algoritama veličina populacije samo omogućuje više lokalne pretrage unutar jedne generacija i veći izbor genetskom algoritmu, ali kod prilagodljivih memetičkih algoritama može potpuno promijeniti rad algoritma. Za različite probleme poboljšanje kvalitete rezultata većinom se kreće u jednom smjeru: ili povećanja ili smanjenja veličine populacije kako bi izvođenje više odgovaralo upravo tom problemu. No može se utvrditi kako će dodavanje broja evaluacija koje algoritam može koristiti do određene mjere pouzdano dati bolji rezultat. To je pogotovo vidljivo na slici 17. gdje su algoritmu s većom populacijom potrebne dodatne evaluacije kako bi se izašlo iz perioda skupljanja podataka. Na ostalim slikama algoritmu je dozvoljena dulja optimizacija pa na taj način i veća vjerojatnost da će u slučaju pogrešnog odabira prije ili kasnije biti odabran prikladniji algoritam lokalne pretrage.

Suprotno tome, *steady state tournament* često pronalazi neki optimum (ne nužno globalni) nakon relativno malog broja generacija i evaluacija te ne nastavlja dalje znatno napredovati. Slično, specijalizirani memetički algoritmi imaju kvalitetu rezultata ograničenu zadanom preciznošću algoritma lokalne pretrage dok prilagodljivi mogu promijeniti metodu optimizacije kad vide da više nema napretka.

5. Zaključak

Prilagodljivi memetički algoritmi ispunili su svoju ulogu kao dobar način istovremenog korištenja više algoritama lokalne pretrage. Pokazali su se sposobnim za rješavanje širokog skupa problema koji se može dalje proširiti dodavanjem novih algoritama, kako za lokalnu pretragu, tako i za selekciju ili ocjenjivanje. Omogućuju korisniku da prepušta neke odluke samom algoritmu, ali uz cijenu povećanog broja evaluacija kvalitete jedinki.

No algoritam i dalje nije potpuno autonoman jer i dalje ovisi o vlastitim postavkama. Koji selektor koristiti, koje i koliko algoritama lokalne pretrage dodati u izbor, koristiti samo prvi ili sve korake tih algoritama, vršiti odabir za svaku jedinku ili za cijelu generaciju i dalje zahtijevaju od korisnika znanje o problemu i umijeće u korištenju algoritma. Čak i nakon tog odabira, veličina populacije treba biti optimizirana tako da ne zahtjeva veći broj evaluacija nego što se može priuštiti, a da se i dalje proizvode kvalitetni rezultati.

Kako bi se ovakvi problemi riješili, bilo bi potrebno ići korak dalje od prilagodljivih memetičkih algoritama uvođenjem još kompleksnije i bolje prilagodbe koja bi vodila računa i o tim parametrima, uz to da ne povećava nerazumno kompleksnost i vrijeme izvođenja. Trenutno bolje rješenje predstavlja usavršavanje specijaliziranih algoritama i poznavanje za koji problem odgovaraju iako to zahtjeva stručnost korisnika, ali prilagodljivi memetički algoritmi mogu se gledati kao dobar korak u drugačijem smjeru.

6. Literatura

[1] Y. S. Ong, M. H. Lim, N. Zhu, K. W. Wong, „Classification of Adaptive Memetic Algorithms: A Comparative Study“, 2006. https://dr.ntu.edu.sg/bitstream/handle/10220/4653/Classification%20of%20Adaptive%20Memetic%20Algorithms-%20A%20Comparative%20Study.pdf

[2] H. P. Schwefel, „Evolution and Optimum Seeking“, 2001. http://ls11-www.cs.uni-dortmund.de/lehre/wiley/

[3] P. Cowling, G. Kendall, E. Soubeiga, „A Hyperheuristic Approach to Scheduling a Sales Summit“, 2000. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.7949&rep=rep1&type=pdf

[4] V. Roth, “Optimization without Gradients: Powell's method”, 2013. http://informatik.unibas.ch/fileadmin/Lectures/HS2013/CS253/PowellAndDP1.pdf

[5] COmparing Continuous Optimisers, 2015. http://coco.gforge.inria.fr/

7. Sažetak

U ovom radu opisuju se prilagodljivi memetički algoritmi kao ideja i implementacija. Oni se oni se koriste kao svestrani optimizacijski algoritmi kojima na raspolaganju stoje razni alati. Pobliže se prikazuju komponente od kojih se sastoje te na koji su način one implementirane. Naposljetku algoritmi su ispitani skupom problema, uspoređeni s drugima te je komentiran njihov postignuti uspjeh.

Ključne riječi: prilagodljivi memetički algoritmi, optimizacija, algoritam lokalne pretrage, genetski algoritam, hiperheuristika, Evolutionary Computation Framework, ECF

8. Summary

This thesis describes adaptive memetic algorithms as an idea and implementation. They are used as versatile optimization algorithms which have various tools at disposal. Components from which they are built from and their respective implementations are thoroughly presented. In the end, the algorithms are tested on a set of problems, compared to others and have comments on their acchieved success.

Keywords: adaptive memetic algorithms, optimization, local search algorithms, genetic algorithm, Evolutionary Computation Framework, ECF