SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Koevolucijski algoritmi

Danko Komlen

Voditelj: *Domagoj Jakobović*

Zagreb, svibanj, 2011.

Sadržaj

1. Uvod 1

2. Pregled i razvoj koevolucijskog računanja 2

2.1 Podjela koevolucijskih algoritama 2

2.2 Razvoj i povijesni pregled 2

3. Natjecateljska koevolucija 3

3.1 Određivanje dobrote 3

3.2 Jedno-populacijska natjecateljska koevolucija 3

3.3 Dvo-populacijska natjecateljska koevolucija 4

4. Suradnička koevolucija 7

4.1 N-populacijska suradnička koevolucija 7

5. Ostvarenje podrške za koevolucijske algoritme u ECF-u 9

5.1 Arhitektura općenitog koevolucijskog algoritma 9

5.2 Simbolička regresija korištenjem 2PC CA 9

5.3 Rezultati ispitivanja 10

6. Zaključak 12

7. Literatura 13

8. Sažetak 14

1. Uvod

Koevolucija (eng. *coevolution*) je pojam koji se najčešće koristi u kontekstu biologije te znanstvenih područja koja se bave proučavanjem složenih ekosustava. Radi se o promjeni nekog biološkog objekta koja je uzrokovana promjenom drugog objekta koji je s njim u interakciji. U prirodi postoji mnogo primjera takve paralelne evolucije različith vrsta. Primjerice, pčele i cvijeće koje one oprašuju, koevoluirali su na način da su potrebni jedno drugome za opstanak. Osim suradnje, koevolucija je prisutna i u obliku međusobnog nadmetanja vrsta, poput antilope i geparda čije su fizičke sposobnosti rezultat neprestane borbe za preživljavanje.

Koevolucijski algoritmi (eng. *coevolutionary algorithm*, CA) pripadaju grani računarstva pod nazivom koevolucijsko računanje (eng. *coevolutionary computation*). Radi se o primjeni koncepta koevolucije na metaheurističke optimizacijske metode. Karakteristika takvih algoritama za razliku od uobičajenih evolucijskih algoritama je da omogućuju istovremen razvoj rješenja problema kao i sam problem koji je potrebno riješiti. Time je moguće obuhvatiti širi prostor problema i razviti kompleksnija i robusnija rješenja. Problem koji se javlja u razvoju koevolucijskih algoritama najčešće je nepredvidivo ponašanje sustava uzrokovano pretjeranom složenosti. Po uzoru na biološke sustave algoritmi se dijele na suradničke (eng. *cooperational CA*) i natjecateljske (eng. *competitive CA*), svaki primjenjivi na specifičnu vrstu problema.

Rad je podijeljen u dva osnovna dijela. U prvome je obrađena teoretska pozadina koevolucijskih algoritama i dan pregled osnovnih podjela zajedno sa primjerima njihove primjene na praktičnim problemima. U drugom dijelu prikazano je proširenje Okruženja za evolucijsko računanje[1] sa podrškom za koevolucijske algoritme i rješavanje problema simboličke regresije.

[1] eng. Evolutionary computation framework, ECF (http://gp.zemris.fer.hr/ecf)

2. Pregled i razvoj koevolucijskog računanja

Koevolucijsko računanje može se promatrati kao primjena evolucijskog računanja (eng. *evolutionary computation*) na višeagentsku paradigmu [2]. Rješenja i problemi koji se razvijaju predstavljaju agente i njihovom interakcijom stvaraju se promjene u prostoru problema. Evolucija pojedinog agenta određena je okolinom koju sačinjavaju drugi agenti. Osnovna podjela vrši se na suradničke i natjecateljske algoritme, a njihov je razvoj, motiviran proučavanjem složenih dinamičkih ekosustava, započeo 1957. godine.

2.1 Podjela koevolucijskih algoritama

Kao što je već navedeno, koevolucijski algoritmi dijele se na *suradničke* i *natjecateljske*. Kod natjecateljskih CA dobrota jedinki određuje se na temelju natjecanja sa ostalim jedinkama iz iste populacije. Takvi algoritmi često se koriste za razvoj dobrih igračkih strategija, primjerice za razvoj igrača dame ili pokera. Primjer korištenja natjecateljskog CA zajedno sa klasifikatorskim sustavom je rad [5] u kojem se razvija igrač pokera. Osim za razvoj strategije ponašanja, natjecateljski CA koriste se i za rješavanje problema kod kojih je domena beskonačno velika i odabir dobrih primjera za ispitivanje rješenja je izazovan zadatak. Kod takvih problema natjecanje se odvija između rješenja i primjera za ispitivanje.

Osnovna podjela koevolucijskih algoritama[1]:

- Jedno-populacijska natjecateljska koevolucija (1PC)
- Dvo-populacijska natjecateljska koevolucija (2PC)
- N-populacijska suradnička koevolucija (NPC)

Posljednja vrsta CA je N-populacijska suradnička koevolucija. Kod nje se glavni problem dijeli na N manjih podproblema. Rješenje svakog podproblema traži se pomoću posebne populacije jedinki. Dobrota jedinke iz svake od N populacija određuje se evaluacijom čitavog rješenja, pri čemu se iz preostalih populacija uzimaju odgovarajuća (najčešće najbolja) podrješenja.

2.2 Razvoj i povijesni pregled

Primjena prvih koevolucijskih algoritama započela je 1957. kada je Barricelli motiviran idejom simbiogeneze, razvio apstraktni ekološki model kako bi proučavao evoluciju vrsta koje međusobno surađuju [2]. Conrad je 1970. razvio niz složenih umjetnih ekosustava kako bi proučavao evoluciju i dinamiku populacija. Korištenje evolucijskog računanja za razvoj umjetnih eko-sustava nastavlja se i danas.

3. Natjecateljska koevolucija

Natjecateljska koevolucija je vrsta koevolucije kod koje se rješavanje problema svodi na nadmetanje između istih ili različitih vrsta jedinki. U nastavku je dan pregled jedno-populacijske i dvo-populacijske natjecateljske koevolucije te prikaz problema određivanja dobrote.

3.1 Određivanje dobrote

Kod koevolucijskih algoritama dobrota nije više apsolutna kao kod običnih evolucijskih algoritama. Ona se određuje na temelju ostalih jedinki u populaciji te je potreban nešto drugačiji pristup. Prisutnost relativne dobrote uzrok je dvaju glavnih problema[1]. Prvi je taj što primjena genetičkih operatora poput selekcije ovisi o dobroti jedinki te dinamika njihove primjene može oscilirati ovisno o okolnostima u trenutnoj populaciji. Drugi problem je praćenje napredovanja algoritma kao cjeline. U uobičajenim evolucijskim algoritmima funkcija dobrote stalno raste kako algoritam napreduje, dok je kod CA moguće da jedinke napreduju, a dobrota stagnira jer i protivnička populacija također napreduje. Stoga se uvodi pojam unutarnje (eng. *internal fitness*) i vanjske dobrote (eng. *external fitness*). Unutarnja dobrota koristi se kod primjene genetičkih operatora, a vanjska za praćenje napretka algoritma.

3.2 Jedno-populacijska natjecateljska koevolucija

U pozadini ideje 1PC koevolucije stoji težnja za postepenim razvojem krivulje učenja. Početna populacija puna je loših rješenja iz kojih se međusobnim nadmetanjem mogu postepeno izdvojiti ona bolja. Na taj način populacija predstavlja okolinu koja se korak po korak razvija i postavlja sve teže zahtjeve.

Primjer na kojem bi takav pristup bio dobar je razvoj igrača pokera [5]. Jednostavan način vrednovanja rješenja mogao bi se temeljiti na broju pobjeda ostvarenih protiv dobrog ljudskog igrača. U tom slučaju u početnoj populaciji bolja rješenja ne bi se mogla izdvojiti, jer bi sva bila jednako loša protiv jako dobrog protivnika. Korištenjem koevolucijskog pristupa igrači iz populacije igrali bi jedni protiv drugih i tako postepeno razvijali krivulju učenja.

Općeniti 1PC koevolucijski algoritam

```
1pc {
     inicijaliziraj(P);
     najbolji = null;
     dok (!uvjet_zaustavljanja()){
          unutarnja_evaluacija(P);
          vanjska_evaluacija(P);
          za_svaki(I iz P) {
               ako(najbolji     ==     null     ||     I.vanjska_dobrota     >
```

```
                najboji.vanjska_dobrota)
                      najbolji = I;
          }
     }
     P = gen_operatori(P);
     vrati najbolji;
}
```

Jedno-populacijsku natjecateljsku koevoluciju moguće je ostvariti u *steady-state* obliku, gdje se u svakog generaciji natječu dvije jedinke. Primjer je projekt Ken Stanleya and Risto Miikkulainena pod nazivom rtNEAT u kojem se agenti igraju igru *shoot-em-up* u virtualnoj okolini[6]. Igrači sa najboljim rezultatom ostajali bi u igri dok bi oni sa najlošijim rezultatom bili zamijenjeni sa potomcima najboljih.

3.3 Dvo-populacijska natjecateljska koevolucija

Za razliku od 1PC koevolucije, gdje su sve jedinke predstavljale isti oblik rješenja, u ovom pristupu jedinke imaju posebne uloge ovisno kojoj od dviju populacija pripadaju. Populacija *P* (eng. *primary population*) sadrži jedinke koje predstavljaju rješenja problema, a populacija *Q* (eng. *foil population*) primjere kojima se rješenja testiraju. Jedinka iz populacije *P* ocjenjuje se prema uspješnosti savladavanja test primjera iz populacije *Q*. Primjeri se također podvrgavaju evolucijskom procesu pri čemu njihova dobrota označava koliko se loše s njima nose rješenja iz populacije *P*. Kod evaluacije jedinki iz jedne populacije, druga populacija ima u ulogu konteksta i u tom smislu nazivamo ju suradnička populacija. U konačnici od interesa je samo populacija rješenja.

Općeniti paralelni 2PC koevolucijski algoritam

```
2pc_paralelno {
     inicijaliziraj(P);
     inicijaliziraj(Q);
     najbolji = null;
     dok (!uvjet_zaustavljanja()){
          unutarnja_evaluacija(P,Q);
          unutarnja_evaluacija(Q,P);
          vanjska_evaluacija(P);
          za_svaki(I iz P) {
               ako(najbolji == null || I.vanjska_dobrota >
               najboji.vanjska_dobrota)
                    najbolji = I;
```

```
          }
          P = gen_operatori(P);
          Q = gen_operatori(Q);
     }
     vrati najbolji;
}
```

2PC koevoluciju moguće je izvesti serijski i paralelno (slika 3.1). Slijedna izvedba najprije evaluira jednu populaciju i nad njom izvršava genetičke operatore, a zatim isti postupak ponavlja za drugu populaciju. Nedostaci ove izvedbe su da populacija koja se prva evaluira je uvijek u prednosti jer se njezina suradnička populacija svaki puta evaluira sa uznapredovalim jedinkama. Također, nedostatak je što se evaluacija svake populacije odvija zasebno, što je nepogodno za slučajeve kada se obje populacije mogu evaluirati odjednom. Navedeni nedostaci serijske 2PC implementacije uklonjeni su u paralelnoj izvedbi. Kod nje se obje populacije najprije evaluiraju, a zatim se provode genetički operatori. Kod paralelne izvedbe, populacije je moguće evaluirati sa prethodnim inačicama suradničkih populacija.

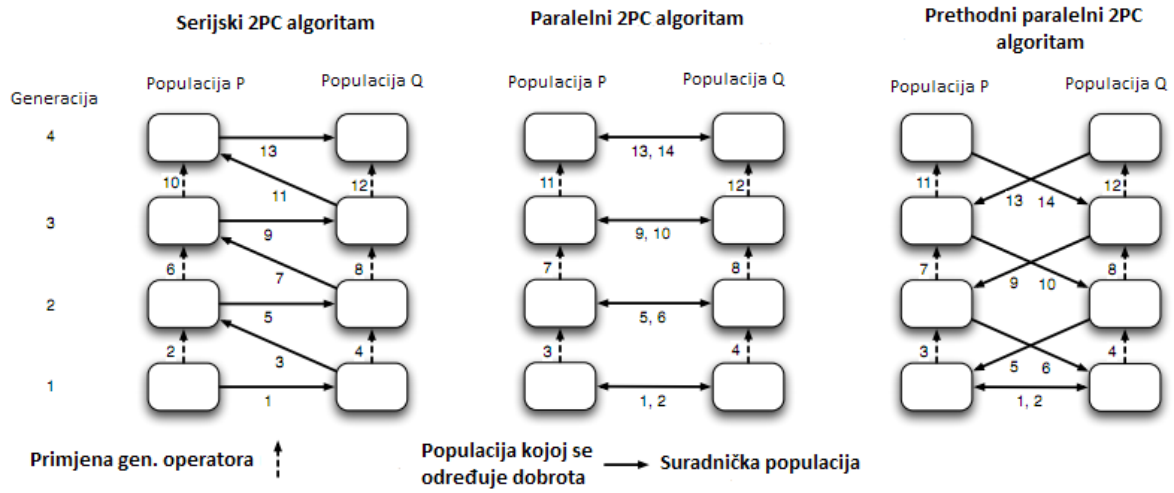
Općeniti prethodni paralelni 2PC koevolucijski algoritam

```
2pc_prethodni {
     inicijaliziraj(P);
     inicijaliziraj(Q);
     najbolji = null;
     unutarnja_evaluacija(P,Q);
     unutarnja_evaluacija(Q,P);
     vanjska_evaluacija(P);
     za_svaki(I iz P) {
          ako(najbolji == null || I.vanjska_dobrota > najboji.vanjska_dobrota)
               najbolji = I;
     }
     dok (!uvjet_zaustavljanja()){
          P' = gen_operatori(P);
          Q' = gen_operatori(Q);

          unutarnja_evaluacija(P',Q);
          unutarnja_evaluacija(Q',P);
          vanjska_evaluacija(P');
          za_svaki(I iz P') {
               ako(najbolji == null || I.vanjska_dobrota > najboji.vanjska_dobrota)
```

```
                    najbolji = I;
            }
            P = P';
            Q = Q';
        }
        vrati najbolji;
}
```

Uobičajena ilustracija 2PC koevolucije je Danny Hillisov pokušaj da pronađe optimalne sortirajuće mreže (eng. *sorting network*) [7]. Sortirajuće mreže predstavljaju opis algoritma za sortiranje pomoću niza zamjena koje je potrebno izvesti između odgovarajućih elemenata u nizu. Optimalna mreža je ona koja za zadanu veličinu niza ima najmanji broj potrebnih zamjena. Proučavanje takvih opisa algoritama za sortiranje korisno je zbog mogućnosti jednostavne sklopovske implementacije. Odabir nasumičnih nizova za ispitivanje mreže nije se pokazao uspješnim, jer je na taj način teško pokriti sve zahtjevnije ulaze. Iz tog razloga Hills se odlučio za 2PC koevoluciju gdje je istovremeno sa mrežama razvijao i nizove za njihovo ispitivanje. Kako su mreže napredovale, tako su i primjeri postajali sve teži.



Slika 3.1 *Načini ostvarenja 2PC algoritma[1]*

Na 2PC koevoluciju može se gledati kao na utrku (eng. *arms-race*) [1]. Jedna populacija razvije prilagodbu kojom natjera drugu da razvije vlastitu prilagodbu kojom će poraziti prethodnu. Ubičajeni rezultat takvog međusobnog nadmetanja prilagodbama je postepeni rast krivulje učenja. Međutim, u praktičnim primjenama to ne mora biti slučaj. Moguće je da jedna populacija značajnije napreduje u odnosu na drugu. Tada će sve jedinke iz te populacije imati istu dobrotu jer nadmašuju sve jedinke iz druge populacije. Također, u suradničkoj populaciji događa se ista pojava jer sve jedinke su jednako loše. Navedna okolnost uzrokuje gubitak gradijenta na krivulji učenja, jer genetički operatori nemaju dobru podlogu u dobroti jedinki da bi ih mogli poboljšati.

4. Suradnička koevolucija

U prethodnim vrstama koevolucijskih algoritama jedinke su se međusobno nadmetale kako bi cjelokupan proces evolucije vodio ka boljim rješenjima. Kod suradničke koevolucije jedinke zajednički stvaraju cjelokupno rješenje i njihova suradnja vodi sustav ka napretku. Primjer suradničke koevolucije je rad Colinsa i Jeffersona [2] koji su koristili GA za razvoj mrava pogonjenih neuronskim mrežama. Mravi su živjeli u kolonijama od 20 jedinki i cilj im je bio razviti efikasno pronalaziti hranu. Reynolds je 1993. koristio GP kako bi koevoluirao grupno ponašanje simuliranih mobilnih agenata. Osim za primjenu na višeagentske sustave koevolucijom je moguće rješavati složene probleme koji se mogu rastaviti na manje, što je prikazano u nastavku poglavlja.

4.1 N-populacijska suradnička koevolucija

NPC koevolucija koristi se kod problema sa velikim prostorom rješenja i kod kojih je moguće napraviti dekompoziciju na manje podprobleme. Svaki od N podproblema rješava se zasebnom populacijom i za razliku od 2PC koevolucije svaka od populacija je od interesa u konačnici. Pojedinoj jedinki dobrota se dodjeljuje prema uspješnosti cijelog rješenja, koje uključuje nju i najbolje jedinke iz svake od ostalih N-1 populacija. Kod ovakvog oblike koevolucije praćenje napretka cijelog algoritma je jednostavnije, jer se u svakom koraku, uzimanjem najboljih jedinki iz svake populacije, može odrediti i evaluirati trenutno najbolje rješenje.

Izvedba NPC algoritma, kao i kod 2PC moguća je serijski i paralelno. Kod serijske izvedbe vrši se više evaluacija i stoga je paralelna nešto praktičnija. Funkcija *evaluacija(P_1,...,P_n)* izvodi redom testove za svaku jedinku iz svake populacije. Međutim, za razliku od serijske implementacije ovdje se pamte rezultati ispitivanja te se mogu iskoristiti za izračunavanje dobrota suradničkih jedinki.

Općeniti paralelni NPC koevolucijski algoritam

```
npc_paralelno {
    inicijaliziraj(P1,...,Pn);
    najbolji = (null,...,null);
    dok (!uvjet_zaustavljanja()){
        evaluacija(P1,...,Pn);
        za_svaki( (I1,...,In), Ii iz Pi) {
            ako(najbolji == (null,...,null) ||
            zajednicka_dobrota(I1,...,In) >
            zajednicka_dobrota(najboji))
                najbolji = (I1,...,In);
        }
```

```
        za (i = 1 do n) {
            P_i = gen_operatori(P_i);
        }
    }
    vrati najbolji;
}
```

5. Ostvarenje podrške za koevolucijske algoritme u ECF-u

Programsko ostvarenje koevolucijskog algoritma izvedeno je u sklopu Okruženja za evolucijsko računanje. Pri tome su bile potrebne prilagodbe samog okruženja te izvedba arhitekture općenitog koevolucijskog algoritma, pomoću kojeg se mogu rješavati različiti problemi. U nastavku je dan pregled navedene arhitekture te primjer njenog korištenja za rješavanje problema simboličke regresije, zajedno sa pripadnim rezultatima.

5.1 Arhitektura općenitog koevolucijskog algoritma

Za ostvarenje podrške za koevolucijske algoritme bile su potrebne promjene u temeljnoj arhitekturi Okruženja za evolucijsko računanje. Tako je uobičajeni način rada ECF–a sa jednom populacijom, proširen na proizvoljan broj populacija. Nad svakom populacijom djeluje algoritam zadužen za nju. Također, jedinke mogu imati različite genotipove ovisno o populaciji kojoj pripadaju. Prilagodbe u okruženju vršene su po uzoru na [4].

Osnovni razred za korištenje koevolucijskih algoritama u sklopu okruženja je *CoevEvalOp*. On je vrsta evaluacijskog operatora prilagođenog za rad sa više populacija. Za svaki problem koji se rješava koevoluvijom potrebno je napisati konfiguracijske datoteke za svaku populaciju. U njima se nalazi opis algoritma koji se koristi te opis genotipa jedinke. Također, za svaku populaciju potrebno je napisati evaluacijski operator koji nasljeđuje razred *CoevEvalOp* i služi za određivanje dobrote jedinke iz dotične populacije.

Evaluacijski operator mora implementirati metode *evaluate*, *makeSet* i *initEval*. Metoda *evaluate* služi za određivanje vrijednosti dobrote jedinke na temelju jedinki iz ostalih populacija. Jedinke iz ostalih populacija mogu se dohvatiti pomoću varijable *indSets_*, koja je zapravo lista skupova jedinki koje svaka populacija priprema pomoću metode *makeSet*. Metoda *initEval* služi za određivanje početne dobrote jedinke, kada još nisu poznati ostali skupovi jedinki potrebni za njenu evaluaciju.

Pozivanje metoda *makeSet* i *initEval* u odgovarajućim trenucima vrši bazni razred *CoevEvalOp*. Metoda makeSet kojom se stvaraju skupovi jedinki potrebni za evaluaciju, poziva se za svaki evaluacijski operator prije početka nove generacije. Tako je moguće ostvariti različite vrste koevolucijskih algoritama u paralelnoj izvedbi.

5.2 Simbolička regresija korištenjem 2PC CA

Algoritmi evolucijskog računanja zahtjevaju odabir primjera za računanje dobrote rješenja. Poseban izazov predstavljaju problemi sa velikom domenom iz koje je potrebno odabrati primjere koji dobro ocjenjuju kvalitetu rješenja. Zbog prevelikog skupa ulaznih vrijednosti nije moguće odrediti optimalnost dobivenog rješenja.

Tehnika genetičkog programiranja koristi se za razvoj aproksimacija određene funkcije (simbolička regresija). Odabir dobrih primjera za ispitivanje aproksimacije je zahtjevan zbog najčešće beskonačno velike domene funkcije. Jedno od mogućih rješenja je korištenje ideje koevolucije kao tehnike za odabir dobrih primjera iz velikog skupa mogućih ulaznih vrijednosti [3]. Primjeri se evoluiraju zajedno sa rješenjima, a dobrota se određuje

prema tome koliko loše rezultate daju trenutna rješenja za pojedini primjer. Želja je dobiti što robusnija rješenja koja dobro generaliziraju izlaze i za još ne viđene primjere.

Funkcija koja se aproksimirira je $f(x) = x^4+x^3+x^2+x$ i to nad domenom realnih brojeva iz intervala [-1 1]. Korisit se dvo-populacijski natjecateljski koevolucijski algoritam (2PC CA). Populacija rješenja *P* sastoji se od matematičkih izraza koji aproksimiraju funkciju f i nad njom djeluje algoritam genetičkog programiranja (GP). Druga populacija (*Q*) sastoji se od realnih vrijednosti iz intervala [-1, 1] koje predstavljaju primjere sa kojima se ispituju rješenja. Nad njom djeluje genetički algoritam (GA) kako bi se stvorili što teži primjeri.

Rješenja su predstavljenja sintaksnim stablima koja određuju matematičke izraze (Tree genotip). Njihova evolucija vrši se pomoću *steady-state* genetičkog algoritma uz veličinu turnira od tri jedinke. Za evaluaciju se koristi razred *GPSymbRegEvalOp* koji rješenje evaluira na temenju najbolje jedinke iz populacije primjera za ispitivanje (izraz 5.1).

Svaka od jedinki iz populacije primjera za ispitivanje sastoji se od 10 realnih brojeva iz intervala [-1, 1]. Nad njima djeluje generacijski genetički algoritam sa proporcionalnom selekcijom. Za evaluaciju se koristi razred *GASymbRegEvalOp* koji svaku jedinku evaluira na temelju najbolje jedinke iz populacije rješenja (izraz 5.2).

Algoritam je ostvaren kao prethodni paralelni 2PC koevolucijski algoritam, jer se kod svake evaluacije koriste najbolje jedinke iz prethodne generacije. Dobrote jedinki računaju se prema sljedećim izrazima:

$$f(r) = \frac{1}{n}\sum_{i=1}^{n}[r(t\_best_i) - f(t\_best_i)]^2 \tag{5.1}$$

$$f(t) = \frac{1}{n}\sum_{i=1}^{n}[r\_best(t_i) - f(t_i)]^2 \tag{5.2}$$

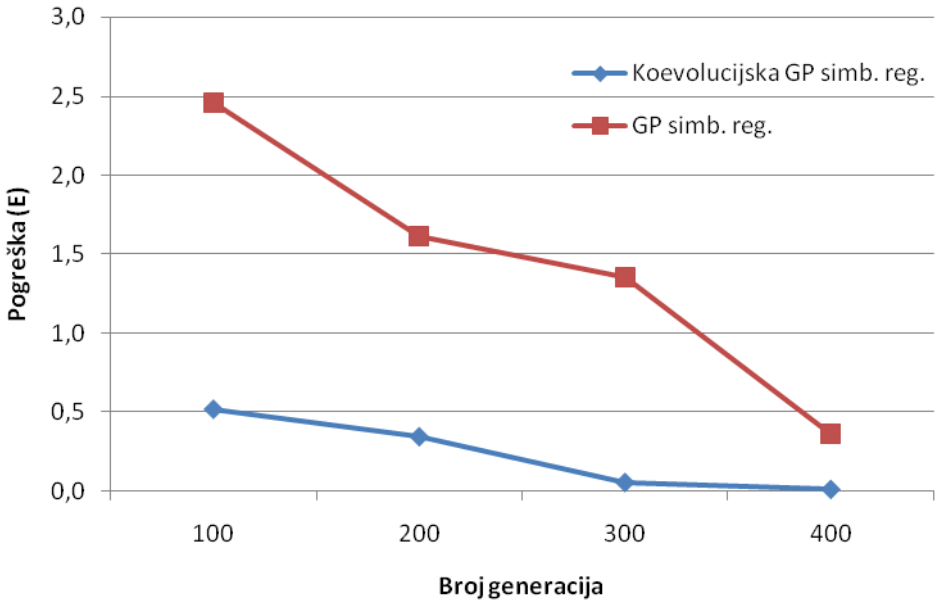
Simbol *r* označava jedinku rješenja, a *t* jedinku primjera za ispitivanje (t_i je i-ti realni broj iz jedinke). *t_best* i *r_best* su najbolje jedinke iz prethodne generacije. Broj primjera koje sadrži jedinka za ispitivanje je *n* i u konkretnom primjeru jednaka je 10, a *f* je funkcija čiji se opis traži. Jedinke iz populacije rješenja su to bolja što im je vrijednost dobrote manje, dok je kod populacije primjera za ispitivanje obrnuto.

5.3 Rezultati ispitivanja

Ispitivanje programskog rješenja koevolucijske simboličke regresije vršeno je zajedno sa ispitivanjem obične GP simboličke regresije sa jednom populacijom, koja se evaluirala nad unaprijed zadanih 10 realnih vrijednosti. Drugi algoritam bio je također *steady-state* GA sa populacijom sintaksnih stabala. Kod GP dijela algoritma korišteni su operatori *sin*, *cos*, +, -, * i /, te konstante X i 1 kao završni simboli. Algoritmi su ispitivani nad zasebnim skupom za validaciju koji se sastojao od 100 nasumičnih realnih brojeva iz intervala [-1, 1].

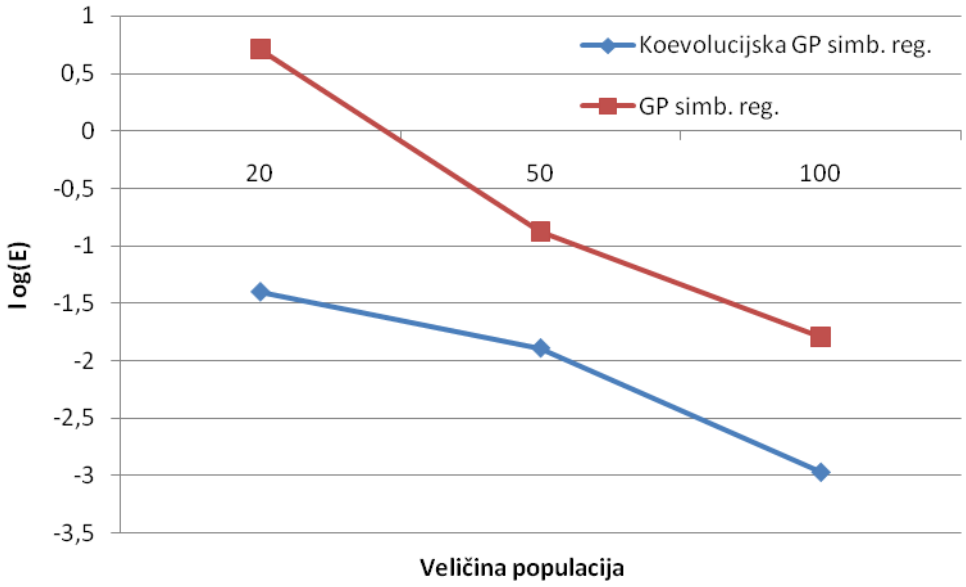
Na slici 5.1 prikazani su rezultati oba algoritma u ovisnositi o broju generacija za veličinu populacije od 20 jedinki. Vidljivo je da oba algoritma daju bolje rezultate za veći broj

iteracija te da koevolucijski algoritam daje rješenja koja bolje generaliziraju od obične GP simboličke regresije.



Slika 5.1 *Ovisnost pogreške regresije o broju generacija*

Na slici 5.2 prikazana je ovisnost pogreške regresije o veličini populacije. Moguće je uočiti da veličina populacije ima veliki utjecaj na konačno rješenje kod oba algoritma te da je i u ovim mjerenjima koevolucijski algoritam dao bolje rezultate. Svi prikazani podaci dobiveni su kao prosjek od pet mjerenja.



Slika 5.2 *Ovisnost pogreške regresije o veličini populacija*

6. Zaključak

Koevolucijski algoritmi obuhvaćaju široko područje u evolucijskom računanju. Problemi kod kojih je teško izraziti ocjenu dobrote posebno su pogodni za primjenu natjecateljskih CA. S druge strane suradnički koevolucijski algoritmi pogodni su za stvaranje rješenja koje se može podijeliti manje dijelove sa složenom interakcijom. U konkretnom primjeru simboličke regresije pokazano je kako implementirani algoritam daje bolje rezultate od uobičajenog genetičkog programiranja sa jednom populacijom. Zbog prilagodbe ulaznih vrijednosti, dobivena rješenja davala su bolje rezultate za još neviđene ulaze. Budući rad na implementaciji mogao bi uključivati primjenu algoritma na probleme suradničke koevolucije.

7. Literatura

[1] Luke S., „Essentials of Metaheuristics“, Lulu, 2009., str. 103.-125.

[2] Bull L., „Coevolutionary Computation: An Introduction“, University of the West of England

[3] Panait L., Luke S., „Methods for Evolving Robust Programs“, GECO 2003., Chicago: Springer-Verlang, 2003.

[4] Open BEAGLE, a versatile EC framework, http://beagle.gel.ulaval.ca/

[5] Kushida J., Taniguchi N., Hoshino Y., Kamei K., „A Coevolutionary System for Development of Strategies in Poker Game“, Proceedings of the Second International Conference on Innovative Computing, Washington: IEEE Computer Society, 2007.

[6] Stanley K.O., Bryant B.D., Karpov I., Miikkulainen R., „Real-Time Evolution of Neural Networks in the NERO Video Game“, Proceedings of the Twenty-First National Conference on Artificial Intelligence, 2006.

[7] D Hillis, „Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure“, Emergent Computation, str. 228.-234.

8. Sažetak

Rad obuhvaća teoretski pregled koevolucijskih algoritama te opis njihove ugradnje u Okruženje za evolucijsko računanje. Koevolucija je biološki proces u kojem interakcija između populacija različitih jedinki dovodi do nadmetanja i neprestanog prilagođavanja ili suradnje u obliku simbioze. Primjenom koevolucije u području evolucijskog računanja razvijeni su koevolucijski algoritmi. U radu se objašnjava razlika između klasičnih populacijskih optimizacijskih algoritama i koevolucijskih algoritama te njihova podjela na suradničke i natjecateljske. Osnova suradničkih algoritama je da jedinke iz svake populacije zajednički stvaraju rješenje problema s ciljem što bolje suradnje, dok se kod natjecateljskih populacije međusobno nadmeću stvarajući postepeno sve bolja rješenja za sve složenije probleme. Također, izneseni su važniji problemi koji se mogu javljati prilikom primjene algoritama, a uzrokovani su složenim dinamičkim ponašanjem koevolucije. Kroz rad se iznose razne primjene koevolucijskih algoritama, poput pronalaženja optimalnih sortirajućih mreža te kod problema gdje je teško eksplicitno odrediti uspješnost rješenja, primjerice kada je cilj razviti dobrog igrača neke igre. Na kraju rada pokazan je primjer korištenja koevolucijskog natjecateljskog algoritma za rješavanje problema simboličke regresije.