SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 417

Modeliranje genske regulacijske mreže pomoću hibridnog koevolucijskog algoritma

Danko Komlen

Zagreb, lipanj 2012.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Zahvaljujem mentoru prof. dr. sc. Domagoju Jakoboviću, na podršci, izvrsnim savjetima i pruženoj pomoći prilikom izrade diplomskog rada i izvođenja ispitivanja.

SADRŽAJ

1. Uvod

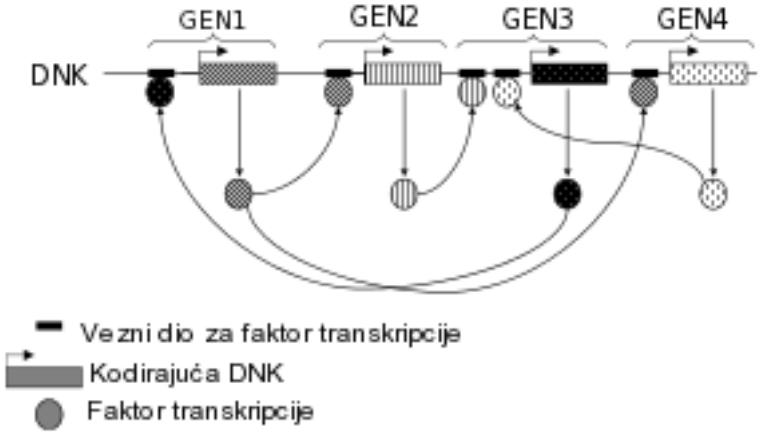
Jedan od aktivnih problema u bioinformatici i sistemskoj biologiji je određivanje regulacijskih interakcija između pojedinih gena. Genska regulacijska mreža (eng. *gene regulatory network*, GRN) je skup gena uz opis njihovih međusobnih interakcija. Za njezin opis i proučavanje koriste se različiti modeli, opisani određenim brojem parametara. Stoga je za uspješno modeliranje mreže potrebno odrediti optimalne parametre, koji opisuju ponašanje dobiveno laboratorijskim mjerenjima. Kod modela mreža s većim brojem gena prostor pretraživanja parametara je također velik, te je jedno od mogućih rješenja korištenje evolucijskih algoritama za postupak optimizacije, što je tema ovog rada. Kao poboljšanje postojećih algoritama upotrijebljena je suradnička koevolucija te su uspoređeni dobiveni rezultati.

U prvom poglavlju dan je detaljniji opis GRN mreža i mogućih modela. Zatim je u iduća dva poglavlja iznesen pregled algoritama evolucijskog računanja i koevolucijskih algoritama. U četvrtom poglavlju objašnjena je primjena algoritama na problem modeliranja mreža, dan je pregled srodnih radova te su navedeni korišteni podatkovni skupovi. U posljednja dva poglavlja opisano je programsko rješenje te su prikazani dobiveni rezultati ispitivanja.

2. Genske regulacijske mreže

2.1. Uvod

Sve stanice u organizmu imaju isti genom[1], a za njihovu diferencijaciju i funkcionalnost zaslužni su složeni mehanizmi na razini molekula koji te informacije tumače. Ključnu ulogu u takvim mehanizmima imaju faktori transkripcije (eng. *transcription factor*, TF) koji se vežu za dijelove DNK (Pal et al., 2006). Nakon vezanja, oni mogu aktivirati ili potisnuti djelovanje određenog gena. Složeni sustavi se javljaju jer su faktori transkripcije također proizvodi gena i mogu biti regulirani drugim faktorom transkripcije (slika 2.1). Također geni mogu biti kontrolirani od strane više faktora transkripcije, a moguće su i povratne veze.



Slika 2.1: Prikaz jednostavne mreže transkripcijskih faktora

Za istraživanje takvih složenih sustava potrebno je moći izmjeriti aktivnost pojedinih gena unutar stanice. Takav postupak naziva se profiliranje izražajnosti gena[2] (eng. *gene expression profiling*) i trenutno postoji više tehnologija koje ga omogućuju. Jedna

[1] nasljedni podaci kodirani u DNK

[2] stvaranje proteina iz odgovarajućeg gena

od često korištenih je DNA *microarray* tehnologija (Baldi et al., 2002), gdje se istovremeno prati kretanje količine mRNA [3] molekula za više gena. Konkretni *microarray* podatkovni skupovi (eng. *microarray data*, MAD) opisani su u poglavlju 5.3.

Genska regulacijska mreža (eng. *gene regulatory network*, GRN) je apstrakcija koja omogućava razumijevanje tih složenih dinamičkih sustava i daje obrazloženje za izražajnost gena koji se promatraju. Prema radu (D'haeseleer et al., 2000) postoje dva osnovna principa na kojima se takva apstrakcija treba temeljiti (D'haeseleer et al., 2000).

1. Tok genske informacije (eng. *genetic information flow*): Iz genskog zapisa i podataka o izražajnosti gena, potrebno je moći objasniti način na koji se statičan genski zapis prevodi u složenu funkcionalnost stanice.

2. Složeni dinamički sustavi (eng. *complex dynamic systems*): GRN mreža opisuje složeni dinamički sustav na molekularnoj razini. Takav sustav ima svoju trajektoriju koju je moguće pratiti profiliranjem izražajnosti gena te postoji atraktor koji označava stabilno stanje u kojem će stanica završiti (npr. diferencirana, zdrava ili bolesna stanica). GRN mreža kao apstrakcija treba omogućiti da se predvidi atraktor u kojem će sustav završiti te pružiti potrebno znanje da se sustav usmjeri u željeni atraktor.

2.2. Vrste modela GRN mreža

GRN mreža je apstrakcija koja služi za opisivanje složenih dinamičkih sustava. Stvaranje takve apstrakcije vrsta je obrnutog inženjerstva (eng. *reverse engineering*) gdje postoji više pristupa rješavanju problema. U nastavku će prema (Schlitt i Brazma, 2007b) biti opisano nekoliko različitih vrsta modela koji se najčešće koriste.

2.2.1. Liste dijelova

Liste dijelova (eng. *parts lists*) predstavljaju skup, opis i sistematizaciju osnovnih dijelova genske mreže određenog organizma, poput faktora transkripcije, promotora i veznih dijelova za faktore transkripcije (eng. *transcription factor binding sites*). Stvaranje takvih modela prvi je korak kod izrade ostalih složenijih modela.

[3] glasnička RNA (eng. *messenger RNA*, mRNA) je RNA koja nastaje prevođenjem/transkripcijom DNA tijekom izražavanja gena i služi za stvaranje proteina

2.2.2. Topološki modeli

Topološki modeli (eng. *topology models*) opisuju strukturu GRN mreže pomoću grafova. Čvorovi grafa odgovaraju genima, a lukovi predstavljaju postojanje interakcije između dva gena. Kod definiranja topološkog modela odnose između gena moguće je odrediti na više načina. Primjerice luk između gena A i B može određivati da je A transkripcijski faktor koji se veže na promotora od ciljnog gena B. Druga mogućnost je da veza između gena označava da poremećaj nad genom A utječe na izražajnost gena B. Preduvjet za stvaranje toploških modela je model liste dijelova.

2.2.3. Modeli upravljačke logike

Prethodno opisani topološki modeli opisuju strukturu GRN mreže, a modeli upravljačke logike (eng. *control logics models*) služe za opisivanje pravila međudjelovanja osnovnih elemenata mreže. Primjerice, ako se više transkripcijskih faktora veže za promotor potrebno je odrediti kako se njihovo djelovanje kombinira. Interakciju je moguće opisati Booleovim funkcijama, linearnim funkcijama ili stablima odluke.

2.2.4. Dinamički modeli

Poznavanje prethodno navedenih modela omogućuje izradu dinamičkih modela (eng. *dynamic models*) koji služe za opis i simulaciju GRN mreže. Tako je moguće promatrati ponašanje modela na različite promjene u okolini i donositi zaključke o stvarnom dinamičkom sustavu koji je opisan modelom. Neki od modela koji se najčešće koriste u radovima su: Booleove mreže, Petrijeve mreže, modeli diferencijalnih jednadžbi te stohastičke mreže. Modele možemo podijeliti na diskretne i kontinuirane ovisno o tome da li se vrijeme gleda kao kontinuirana ili diskretna varijabla.

Uspješnost modeliranja GRN mreže pomoću dinamičkih modela određuje se prema tome koliko se ponašanje modela podudara s profilom izražajnosti gena. U sljedeća dva poglavlja detaljnije su opisana dva dinamička modela koji se koriste u radu.

2.3. S-sustavi

GRN mreže opisuju biomolekularna međudjelovanja koja su nelinearna i mogu se opisati općenitim sustavom diferencijalnih jednadžbi:

$$\frac{dx_i(t)}{dt} = f_i[x_1(t), ..., x_N(t)], \tag{2.1}$$

za i = 1,...,N, pri čemu je N broj gena, x_i izražajnost gena i, a f_i funkcija koja opisuje dinamički utjecaj svih gena na gen i. Tako primjerice ako gen j aktivira gen i, tada f_i raste s porastom x_j i obratno ako gen j inhibira gen i.

Određivanje funkcija f_i koje bi definirale uspješan model predstavlja loše postavljen problem. Iz tog razloga koriste se različite aproksimacije navedenih funkcija.

S-sustavi (eng. *S-systems*) su posebna vrsta sustava diferencijalnih jednadžbi gdje se funkcija f_i aproksimira sljedećim izrazom (rad (Kikuchi et al., 2003)):

$$\frac{dx_i(t)}{dt} = \alpha_i \prod_{j=1}^{N} {x_j}^{g_{i,j}} - \beta_i \prod_{j=1}^{N} {x_j}^{h_{i,j}} \tag{2.2}$$

Model se sastoji od ukupno $2N^2 + 2N$ parametara.

2.4. Linearan vremenski promjenjiv model

U radu (Kim et al., 2008), predstavljen je linearan vremenski promjenjiv model (eng. *linear time-variant model*, LTV) gdje se koristi sljedeća aproksimacija funkcija f_i:

$$\frac{dx_i(t)}{dt} = \sum_{j=1}^{N} W_{i,j}(t) x_j(t), \tag{2.3}$$

pri čemu je $W_{i,j}(t)$ matrica međudjelovanja gena (regulacijska matrica). Njezine vrijednosti ovise o trenutku t, što omogućuje opisivanje nelinearnosti u sustavu. Vrijednosti matrice računate su kao suma prva dva člana Fourierovog reda:

$$W_{i,j}(t) = \alpha_{i,j} sin(\omega_i t + \phi_{i,j}) + \beta_{i,j}. \tag{2.4}$$

Koeficijent $W_{i,j}$ određuje jakost utjecaja gena j na regulaciju gena i. Pozitivna vrijednost označava da gen j aktivira gen i, negativna da ga inhibira, a nula označava da gen j ne utječe na transkripciju gena i.

Ovaj model korišten je i u radu (Kabir et al., 2010) uz nešto drukčiji pristup. Autori su koristili diskretan oblik modela te su vrijednosti izražajnosti gena (x_i) računate izravno primjenom sigmoidalne funkcije na regulacijski ulaz:

$$x_i(t+1) = \frac{1}{1 + e^{(-Z_i(t))}}. \tag{2.5}$$

Regulacijski ulaz $Z_i(t)$ označava utjecaj svih gena na gen i, a definiran je kao:

$$Z_i(t) = \sum_{j=1}^{N} W_{i,j}(t) x_j(t). \tag{2.6}$$

3. Algoritmi evolucijskog računanja

3.1. Uvod

Evolucijsko računanje (eng. *evolutionary computation*, EC) grana je umjetne inteligencije (eng. *artificial intelligence*, AI) koja se bavi rješavanjem optimizacijskih problema primjenom koncepta evolucije. EC algoritmi rade sa populacijom rješenja koja se iterativno poboljšava dok se ne zadovolji određeni uvjet zaustavljanja poput broja iteracija ili se ne postigne zadovoljavajuće rješenje. Početnu populaciju svih algoritama moguće je odrediti nasumično ili nekim drugim optimizacijskim procesom.

U nastavku su detaljnije opisani genetski algoritam, algoritam diferencijske evolucije, algoritam roja čestica ta hibridni Hook-Jeeves GA.

3.2. Genetski algoritam

Genetski algoritam (eng. *genetic algorithm*, GA) heuristička je metoda optimiranja koja imitira prirodni evolucijski proces, robustan proces pretraživanja prostora rješenja.

Algoritam radi sa populacijom potencijalnih rješenja koje nazivamo jedinkama (eng. *individual*). Mjera uspješnosti jedinke naziva se dobrota (eng. *fitness*) i ovisi o problemu koji se rješava. Po uzoru na genetski materijal u živim bićima, jedinke GA opisuju se genotipom iz kojeg je prevođenjem moguće dobiti fenotip, odnosno rješenje koje možemo evaluirati. Neke od mogućih vrsta genotipa su:

- binarni zapis,
- zapis s pomičnom točkom,
- zapis pomoću stabla.

Glavne faze izvođenja GA nazivaju se generacije. Prilikom prijelaza iz jedne generacije u drugu, postupkom selekcije izdvajaju se bolje jedinke koje dalje postupkom reprodukcije stvaraju jedinke sljedeće generacije. Neke od češće korištenih selekcija su:

– jednostavna selekcija (eng. *roulette wheel selection*),

– turnirska selekcija (eng. *tournament selection*) i

– eliminacijska selekcija (eng. *steady-state selection*).

Kod postupka reprodukcije koriste se genetski operatori križanja (eng. *crossover*) i mutacije (eng. *mutation*), koje je potrebno potrebno definirati za svaku vrstu genotipa. Križanjem se iz dvije jedinke stvara nova koja nasljeđuje određena svojstva roditelja, a mutacijom se rade nasumične promjene nad genotipom što omogućuje slučajno pretraživanje prostora rješenja.

Algoritam 1 Genetski algoritam, GA

function GA(n, m, p_c, p_m)
  $P \leftarrow stvori\_pop(n)$
  while $!uvjet\_zaustavljanja()$ **do**
    $Q \leftarrow najbolja(0.1 \cdot velicina(P), P)$
    while $velicina(Q) < n \cdot m$ **do**
      $p \leftarrow prop\_sel(P)$
      $q \leftarrow prop\_sel(P)$
      $novi \leftarrow krizanje(p, q, p_c)$
      $mutacija(novi, p_m)$
      $Q \leftarrow Q \cup novi$
    end while
    $P \leftarrow najbolja(n, Q)$
  end while
  return $najbolji(P)$
end function

U radu je korištena modificirana generacijska inačica GA (eng. *generational GA*) s elitizmom iz rada (Luke, 2009b) uz jednostavnu selekciju (algoritam 1). Iz prethodne generacije uzima se 10% najboljih jedinki, a zatim se postupkom reprodukcije stvaraju potomci dok veličina populacije ne dosegne $m \cdot n$ jedinki. Pri tome je m faktor napuhavanja kojim se određuje selekcijski pritisak. Na kraju se uzima n najboljih jedinki za sljedeću generaciju te se postupak ponavlja.

Za genotip je korišten zapis s pomičnom točkom uz težinsko križanje. Potomak x se iz roditelja p i q s vjerojatnošću p_c određuje sljedeći način:

$$x_i = \frac{dobrota(p)p_i + dobrota(q)q_i}{dobrota(p) + dobrota(q)}, \tag{3.1}$$

a s vjerojatnošću $1 - p_c$ se za potomka uzima roditelj p. Mutacija se vrši da se na svaku vrijednost genotipa s vjerojatnošću p_m dodaje slučajna varijabla s Gaussovom razdiobom[1]. Kod križanja i mutacije vrijednost se postavlja na rubnu vrijednost ukoliko izlazi iz dozvoljenog intervala.

3.3. Diferencijska evolucija

Diferencijska evolucija (eng. *differential evolution*, DE) heuristička je metoda optimiranja prikladna za nelinearne i nediferencijabilne kontinuirane prostore rješenja. Rješenja su predstavljena populacijom vektora, a nova generacija se stvara zamjenom lošijih vektora boljima koji su dobiveni operacijama zbrajanja, oduzimanja i skaliranja nad vektorima trenutne populacije. Postoji više inačica algoritma, a u nastavku je opisana DE1 inačica iz rada (Storn i Price, 1995).

Algoritam 2 Algoritam diferencijske evolucije, DE

function DE(n, F, p_c)
  $P \leftarrow stvori\_pop(n)$
  $\vec{v}_{best} \leftarrow P_0$
  while $!uvjet\_zaustavljanja()$ **do**
    for $i = 1$ do $velicina(P)$ **do**
      $\vec{v}_{target} \leftarrow P_i$
      $\vec{v}_{mutant} \leftarrow diferencijacija(P, \vec{v}_{target}, F)$
      $\vec{v}_{trial} \leftarrow krizanje(\vec{v}_{target}, \vec{v}_{mutant}, p_c)$
      if $dobrota(\vec{v}_{trial}) \geq dobrota(\vec{v}_{target})$ **then**
        $P_i \leftarrow \vec{v}_{trial}$
      end if
      if $dobrota(\vec{v}_{trial}) > dobrota(\vec{v}_{best})$ **then**
        $\vec{v}_{best} \leftarrow \vec{v}_{trial}$
      end if
    end for
  end while
  return $\vec{v}_{best}$
end function

U algoritmu 2 koriste se tri vrste vektora: ciljni ($\vec{v}_{target}$), mutirani ($\vec{v}_{mutant}$) i trial vektor ($\vec{v}_{trial}$).

[1] slučajna varijabla ima razdiobu $G(\mu = 0, \sigma^2 = 1)$

Operatori diferencijacije

Stvaranje vektora mutanta operatorom diferencijacije moguće je ostvariti na više načina (metoda *diferencijacija*() u algoritmu 2). Općeniti izraz za postupak je sljedeći:

$$\vec{v}_{mutant} = \alpha + F\beta, \tag{3.2}$$

pri čemu je α bazni vektor, F faktor skaliranja i β vektor razlike. Operatori se razlikuju prema odabiru baznog vektora i načinu računanja vektora razlike. Tri načina diferencijacije iz rada (Price, 1999) prikazani su u tablici 3.1. Vektor $\vec{v}_{best}$ označava najbolje rješenje trenutne populacije, a vektori $\vec{v}_i$ označavaju nasumične vektore, koji su međusobno različiti i različiti od ciljnog vektora.

Tablica 3.1: Operatori diferencijacije DE algoritma

Naziv operatora	**Izraz za dobivanje vektora mutanta**
DE/rand/1	$\vec{v}_{mutant} = \vec{v}_1 + F(\vec{v}_2 - \vec{v}_3)$
DE/rand/2	$\vec{v}_{mutant} = \vec{v}_1 + F(\vec{v}_2 + \vec{v}_3 - \vec{v}_4 - \vec{v}_5)$
DE/best/1	$\vec{v}_{mutant} = \vec{v}_{best} + F(\vec{v}_1 - \vec{v}_2)$

Operatori križanja

Metoda *krizanje*() u DE algoritmu stvara novi vektor na temelju vektora $\vec{x}$ i mutiranog vektora $\vec{y}$. Operator uniformnog križanja (procedura 3) za svaki element potomka bira roditelja od kojega će ga preuzeti uz vjerojatnost odabira mutiranog vektora p_c.

Operator eksponencijalnog križanja (procedura 4) od mutiranog vektora preuzima slijedno elemente od nasumično odabrane pozicije. Preuzimanje se može završiti sa vjerojatnošću $1 - p_c$ u svakom trenutku. Ostali elementi preuzeti su od vektora x.

3.4. Algoritam roja čestica

Algoritam roja čestica (eng. *particle swarm optimization*, PSO) heuristička je metoda optimiranja koja radi s populacijom čestica P. Za svaku česticu $p \in P$ poznati su vektori pozicije ($p.\vec{x}$) i brzine ($p.\vec{y}$) u prostoru rješenja te skup susjednih čestica koje su određene na početku algoritma. Korištena je prstenasta struktura susjedstva.

Procedura 3 Operator uniformnog križanja

function KRIZANJEUNIF($\vec{x}, \vec{y}, p_c$)
  $j \leftarrow rand(1, D)$
  $\vec{z} \leftarrow \vec{x}$
  for $i = 1$ do D **do**
    if $i == j || rand(0, 1) < p_c$ **then**
      $\vec{z}_i \leftarrow \vec{y}_i$
    end if
  end for
  return $\vec{z}$
end function

Procedura 4 Operator eksponencijalnog križanja

function KRIZANJEEKSP($\vec{x}, \vec{y}, p_c$)
  $\vec{z} \leftarrow \vec{x}$
  $i \leftarrow rand(1, D)$
  $k \leftarrow 0$
  repeat
    $\vec{z}_i \leftarrow \vec{y}_i$
    $i \leftarrow (i + 1)\%D$
    $k \leftarrow k + 1$
  until $rand(0, 1) > p_c \,||\, k == D$
  return $\vec{z}$
end function

Metoda *azuriraj_lokalno(p)* osvježava najbolju poziciju na kojoj se čestica p nalazila ($p.\vec{x}_{lok}$), a metoda *azuriraj_globalno(p)* osvježava najbolje rješenje od čestica iz skupa susjeda.

Algoritam 5 Agoritam roja čestica, PSO

function PSO(n, k, C_1, C_2)
  $P \leftarrow stvori\_pop(n, k)$
  while !$uvjet\_zaustavljanja()$ **do**
    for $i = 1$ do n **do**
      $evaluiraj(P_i)$
      $azuriraj\_lokalno(P_i)$
    end for
    for $i = 1$ do n **do**
      $azuriraj\_globalno(P_i)$
    end for
    for $i = 1$ do n **do**
      $p \leftarrow P_i$
      $p.\vec{v} = w{\cdot}p.\vec{v}{+}C_1{\cdot}rand(0,1){\cdot}(p.\vec{x}_{lok}{-}p.\vec{x}){+}C_2{\cdot}rand(0,1){\cdot}(p.\vec{x}_{glob}{-}p.\vec{x})$
      $p.\vec{x} \leftarrow p.\vec{x} + p.\vec{v}$
      $w \leftarrow azuriraj\_w()$
    end for
  end while
  return $najbolji(P)$
end function

Parametar algoritma k određuje broj čestica u skupu susjedstva, parametar C_1 utjecaj vlastitog najboljeg rješenja, a parametar C_2 utjecaj najboljeg rješenja iz susjedstva na pomak čestice. U algoritmu 5 parametar w označava faktor inercije koji određuje jakost utjecaja prethodne brzine, a osvježava se prema sljedećem izrazu (metoda *azuriraj_w*()):

$$w = \begin{cases} \frac{perc\_comp}{0.8}(W_{max} - W_{min}) + W_{max}, & perc\_comp \leq 0.8 \\ W_{min}, & \text{inače,} \end{cases} \tag{3.3}$$

pri čemu $perc\_comp$ postotak završenosti algoritma, a W_{min} i W_{max} konstante koje određuju najmanju i najveću inerciju.

Opis i usporedba više različitih inačica PSO algoritma te načini stvaranja susjedstva dani su u radu (Oca et al., 2009).

3.5. Hibridni Hook-Jeeves GA

Posljednji korišteni algoritam kombinacija je determinističkog optimizacijskog algoritma Hook-Jeeves (Hooke i Jeeves, 1961) i ranije opisanog genetskog algoritma.

Algoritam radi s populacijom rješenja na način da izvodi po jednu iteraciju Hook-Jeeves algoritma redom nad svakim rješenjem, a u trenutku kada rješenje više ne može napredovati primjenjuju se GA operatori. Iteracija započinje lokalnim pretraživanjem (metoda *istrazi*() u algoritmu 6), gdje se ispituju susjedna rješenja redom po svim koordinatama uz pomak Δx. Ukoliko je pronađeno rješenje bolje od baznog (x^b, pamte se u posebnoj populaciji), bazno rješenje se prebacuje preko pronađenog i umanjuje se pomak.

Kada se pomak dovoljno smanji ($\Delta x < \varepsilon$) rješenje se zamjenjuje potomkom od dva rješenja odabrana operatorom selekcije, a pomak se vraća na početnu vrijednost. Metode *krizanje*() i *mutacija*() rade na isti način kao i kod GA algoritma.

Algoritam 6 Hibridni Hook-Jeeves genetski algoritam, HIB

function GAHOOKJEEVES($n, \Delta x_0, \varepsilon, p_c, p_m$)
  $P \leftarrow stvori\_pop(n)$
  $P^b \leftarrow P$ ▷ populacija baznih rješenja
  $\Delta x \leftarrow [\Delta x_0, ..., \Delta x_0]$
  while !$uvjet\_zaustavljanja()$ **do**
    for $i = 1$ do $velicina(P)$ **do**
      $x \leftarrow P_i$
      $x^b \leftarrow {P^b}_i$
      $x \leftarrow istrazi(x, \Delta x_i)$
      if $dobrota(x) > dobrota(x^b)$ **then**
        $y \leftarrow x$
        for $j = 1$ do $velicina(x)$ **do**
          $x_j \leftarrow 2x_j - {x^b}_j$
        end for
        $x^b \leftarrow y$
      else
        $\Delta x_i \leftarrow \Delta x_i/2$
        $x \leftarrow x^b$
        if $\Delta x_i < \varepsilon$ && $dobrota(P.najbolji) \geq dobrota(x)$ **then**
          $a \leftarrow selekcija(P)$
          $b \leftarrow selekcija(P)$
          $x \leftarrow krizanje(a, b, p_c)$
          $x \leftarrow mutacija(x, p_m)$
          $\Delta x_i \leftarrow \Delta x_0$
          $x^b \leftarrow x$
        end if
      end if
      $P_i \leftarrow x$
      ${P^b}_i \leftarrow x^b$
    end for
  end while
  return $najbolji(P)$
end function

4. Koevolucijski algoritmi

4.1. Uvod

Koevolucija (eng. *coevolution*) pojam je 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 istovremeni 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.

Koevolucijsko računanje može se promatrati kao primjena evolucijskog računanja na višeagentsku paradigmu (Bull, 2008). 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.

4.2. 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. Takvi algoritmi često se koriste za razvoj igračkih strategija, primjerice za razvoj igrača dame ili pokera. Osim za razvoj strategije ponašanja, natjecateljski CA koriste se i za rješavanje problema kod kojih je domena 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 prema radu (Luke, 2009a):

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

Kod NPC koevolucije glavni problem se 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.

4.3. Natjecateljska koevolucija

Natjecateljska koevolucija je vrsta koevolucije kod koje se rješavanje problema svodi na nadmetanje između istih ili različitih vrsta jedinki. Dobrota nije više apsolutna kao kod običnih evolucijskih algoritama. Ona se određuje na temelju ostalih jedinki u populaciji te je potreban nešto drugačiji pristup. Uvodi se 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.

U pozadini ideje 1PC koevolucije stoji težnja za postupnim 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.

Za razliku od 1PC koevolucije, gdje su sve jedinke predstavljale isti oblik rješenja, kod 2PC koevolucije 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.

4.4. Suradnička koevolucija

Za razliku od natjecateljske koevolucije, kod suradničke jedinke zajednički stvaraju cjelokupno rješenje i njihova suradnja vodi sustav ka napretku. Njezina najčešća primjena je u višeagentskim sustavima te kod rješavanja složenih problema koji se mogu rastaviti na manje.

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 jedinci dobrota se dodjeljuje prema uspješnosti cijelog rješenja, koje uključuje nju i najbolje jedinke iz svake od ostalih N-1 populacija. Kod ovakvog oblika 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.

Algoritme je moguće izvesti slijedno (algoritam 7) i paralelno (algoritam 8). Za algoritme koji djeluju nad pojedinom populacijom moguće je odabrati različite optimizacijske algoritme (metoda *EC_algoritam*() u algoritmima).

Algoritam 7 Općeniti slijedni NPC koevolucijski algoritam

$inicijalizacija(P_1, ..., P_n)$
$najbolji \leftarrow null$
while !$uvjet\_zaustavljanja()$ **do**
  for i = 1 do n **do**
    for j = 1 do velicina(P_i) **do**
      $y \leftarrow rjesenje(x_j \in P_i, P_1, ..., P_n)$ ▷ stvaranje globalnog rješenja
      $evaluacija(x_j, y)$ ▷ određivanje dobrote jedinci x_j
      if $najbolji$ = $null$ || $dobrota(najbolji) < dobrota(y)$ **then**
        $najbolji \leftarrow y$
      end if
    end for
    $P_i \leftarrow EC\_algoritam(P_i)$
  end for
end while

Algoritam 8 Općeniti paralelni NPC koevolucijski algoritam

$inicijalizacija(P_1, ..., P_n)$
$najbolji \leftarrow null$
while !$uvjet\_zaustavljanja()$ **do**
  for i = 1 do n **do**
    for j = 1 do velicina(P_i) **do**
      $y \leftarrow rjesenje(x_j \in P_i, P_1, ..., P_n)$ ▷ stvaranje globalnog rješenja
      $evaluacija(x_j, y)$ ▷ određivanje dobrote jedinci x_j
      if $najbolji$ = $null$ || $dobrota(najbolji) < dobrota(y)$ **then**
        $najbolji \leftarrow y$;
      end if
    end for
  end for
  for i = 1 do n **do**
    $P_i \leftarrow EC\_algoritam(P_i)$
  end for
end while

5. Problem modeliranja GRN mreža

Svaki model GRN mreže sadrži skup parametara koji ga definiraju. Nakon odabira vrste modela, sljedeći korak je određivanje vrijednosti parametara. Kod dinamičkih modela, radi se o optimizacijskom problemu gdje je cilj minimizirati pogrešku između podataka dobivenih simuliranjem modela i eksperimentalnih podataka (MAD mjerenja).

5.1. Opis problema

Za modeliranje GRN mreža korišten je linearan vremenski promjenjiv model opisan u radu (Kabir et al., 2010), gdje se za izračunavanje izražajnosti gena koristi izraz 2.5. Skup parametara modela za mrežu od N gena sastoji se od sljedećih $3N^2 + N$ parametara:

$$\{\alpha_{i,j}, \beta_{i,j}, \phi_{i,j}, \omega_i | i, j \in \{1, ..., N\}\}. \tag{5.1}$$

Za funkciju dobrote korišten je izraz:

$$f = -\sum_{k=1}^{M} \sum_{t=1}^{T_k} \sum_{i=1}^{N} \left(\frac{X_{k,i,mod}(t) - X_{k,i,mje}(t)}{X_{k,i,mje}(t)} \right)^2, \tag{5.2}$$

pri čemu je M broj podatkovnih skupova, T_k broj mjerenja u k-tom skupu, a $X_{k,i,mod}(t)$ i $X_{k,i,mje}(t)$ izražajnosti i-tog gena u trenutku t dobivene modelom odnosno eksperimentalnim mjerenjem.

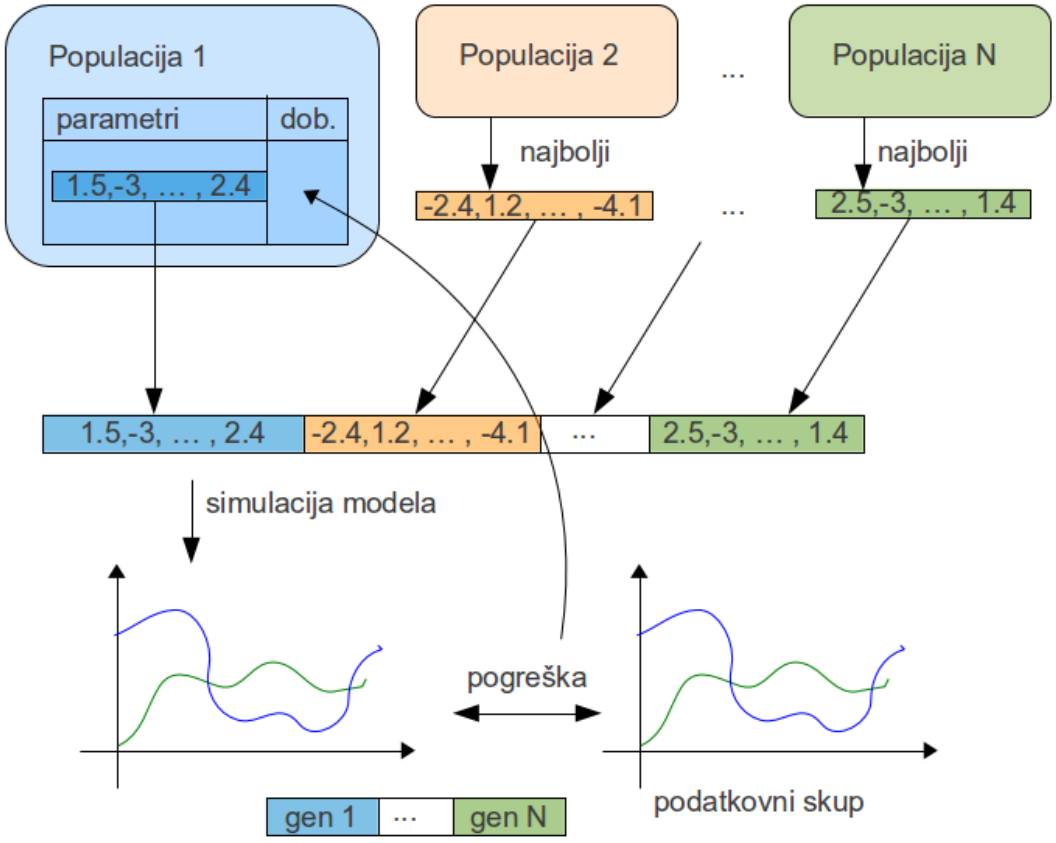
Svaka jedinka EC algoritama sadrži konkretne vrijednosti parametara modela. Također, za svaku vrstu parametara unaprijed su određene rubne vrijednosti.

5.2. Primjena koevolucijskog algoritma

Broj parametara linearnog vremenski promjenjivog modela raste kvadratno u ovisnosti o broju gena u podatkovnom skupu. Iz tog razloga evolucijski algoritmi pretražuju

veliki prostor rješenja, a i brzina izvođenja im se smanjuje. Kao što je objašnjeno u poglavlju 4.4, ovakav oblik problema prikladan je za suradničku koevoluciju uz uvjet da se može definirati podjela problema na manje potprobleme. U nastavku će biti prikazana dva načina podjele koje su korištene.

Prvi način podjele problema je podjela na N populacija, gdje jedinka iz populacije i sadrži $3N + 1$ parametara: $\{\alpha_{i,j}, \beta_{i,j}, \phi_{i,j}, \omega_i | j \in \{1, ..., N\}\}$. Dakle, svaka populacija optimira parametre za određeni gen. Prilikom evaluacije uzimaju se najbolje jedinke iz ostalih populacija i stvara se ukupno rješenje sa svim parametrima koje se ocjenjuje nad zadanim podatkovnim skupom (slika 5.1).

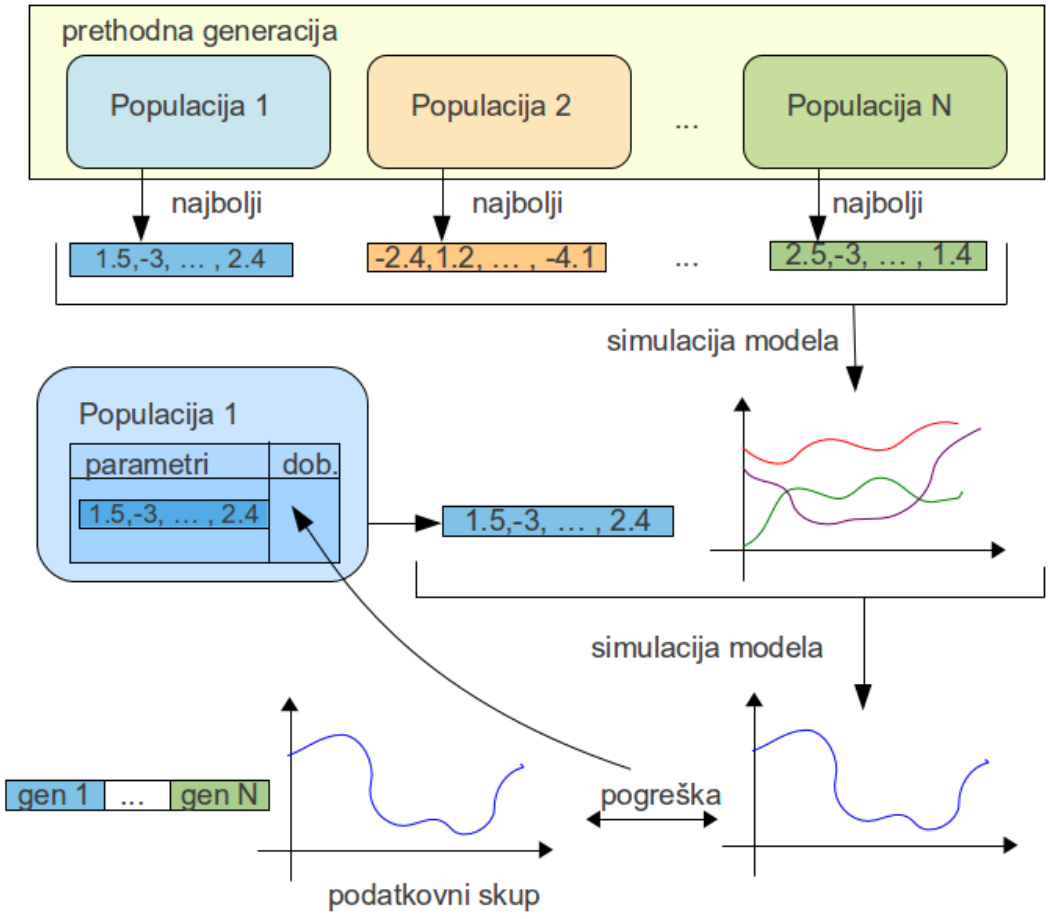


Slika 5.1: Shematski prikaz prvog pristupa podjeli problema kod NPC koevolucije

Drugi način podjele korišten u radu (Kimura et al., 2005) na jednak način dijeli parametre modela na N populacija. Međutim kod ocjenjivanja parametara za određeni gen koristi se sljedeći modificirani izraz za regulacijski ulaz:

$$Z_i(t) = \sum_{j=1}^{N} W_{i,j}(t) Y_j(t)$$
$$Y_j(t) = \begin{cases} x_j(t), & j = i \\ \hat{x}_j(t), & \text{inače} \end{cases} \tag{5.3}$$

pri čemu je $\hat{x}_j(t)$ izražajnost j-tog gena u trenutku t dobivena simulacijom modela za čije su parametre uzete najbolje jedinke iz prethodne generacije. Dakle, na kraju svake iteracije radi se simulacija modela korištenjem najboljih jediniki i tako se stvara $\hat{x}_i(t)$ za svaki gen, koji se onda koristi u izračunu regulacijskog ulaza u sljedećoj generaciji (slika 5.2).



Slika 5.2: Shematski prikaz drugog pristupa podjeli problema kod NPC koevolucije

5.3. Ispitni skupovi

Ispitni skup označava skup od jednog ili više podatkovnih skupova koji se koriste kod ispitivanja algoritama. Njihov pregled dan je u tablici 5.1. Za podatkovne skupove koriste se MAD podaci spomenuti u poglavlju 2.1.

Tablica 5.1: Pregled ispitnih skupova

Naziv	**Broj gena (N)**	**Broj mjerenja (T)**	**Broj pod. skupova (K)**	**Vrsta pod. skupova**
IS-param	5	20	1	Tominaga5
IS-t2	2	11	1	Tominaga2
IS-t5	5	11	10	Tominaga5
IS-yeast6	6	18	1	Spellman p.s.

Podatkovni skupovi Tominaga2 i Tominaga5 su umjetno dobiveni korištenjem S-sustava čiji su parametri dani u tablicama A.1 i A.2. Ovi sustavi predstavljeni su u radu (Tominaga et al., 1999) i često se koriste za mjerenje uspješnosti algoritama. Skupovi su prikazani na slikama 5.3 i 5.4. Apscisa grafova označava vrijeme pri čemu vremenski razmaci između dva mjerenja mogu biti različiti, a za LTV model je bitan samo njihov broj (T). Ordinata označava izražajnost gena i sadrži vrijednosti između 0 i 1.

Ispitni skup Is-yeast6 sadrži eksperimentalna mjerenja dva stanična ciklusa organizma *Saccharomyces Cerevisiae*, a podaci su preuzeti iz Spellmanovog podatkovnog skupa[1]. Skup je prikazan na slici 5.5.

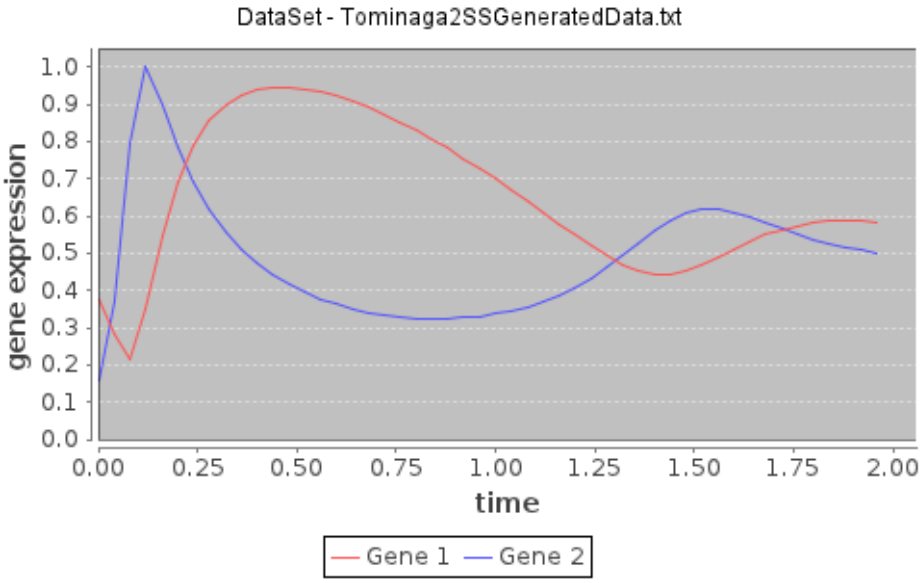
Ispitni skup IS-param korišten je za optimizaciju parametara algoritama, dok su ostali korišteni za usporedbu uspješnosti.

5.4. Pregled srodnih radova

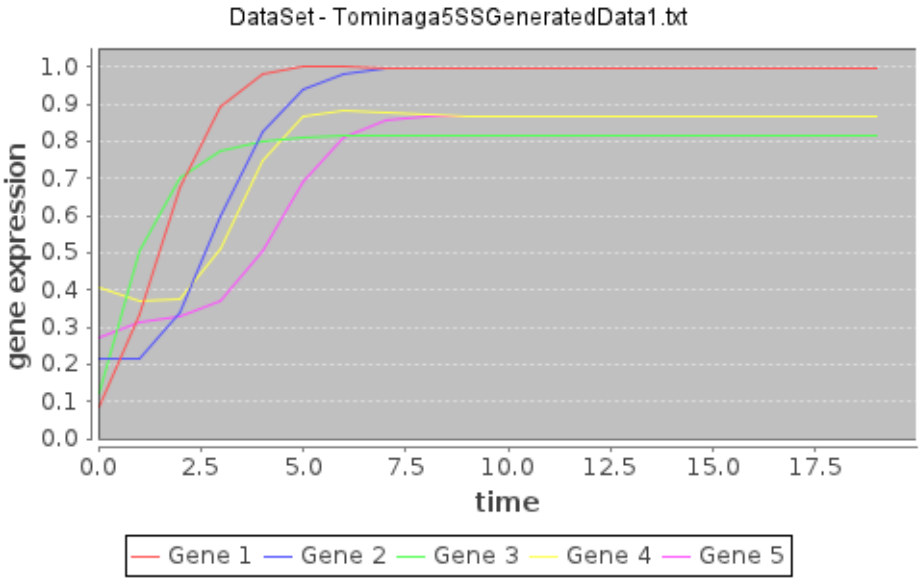
Problem modeliranja GRN mreže jedan je od trenutno aktualnijih problema iz područja sustavne biologije (eng. *system biology*). Primjena evolucijskog računanja u svrhu optimizacije parametara modela samo je jedan od mogućih pristupa.

U tablici 5.2 dan je prikaz korištenih algoritama, modela, funkcija dobrote i podatkovnih skupova iz više različitih znanstvenih radova.

[1] dostupan u KEGG bazi podataka (http://www.genome.jp/kegg/)

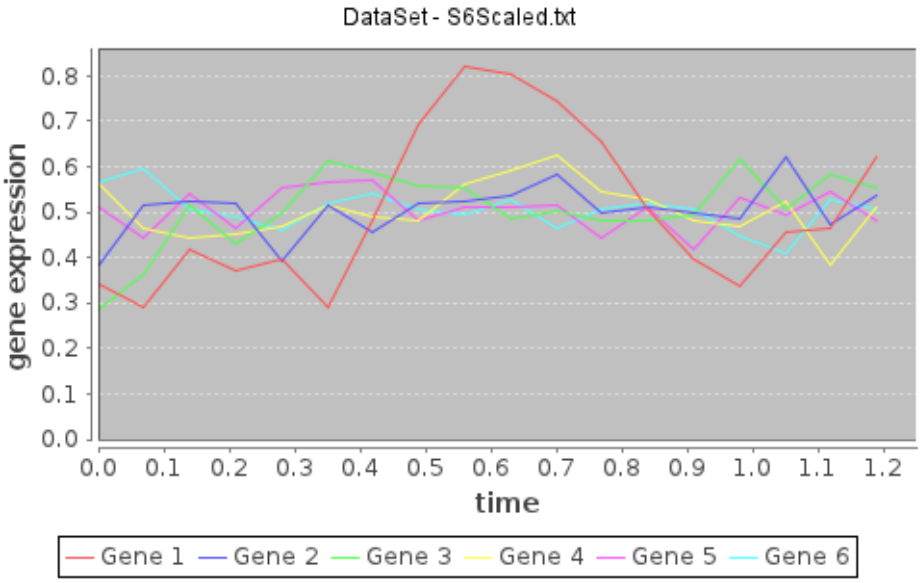


Slika 5.3: Podatkovni skup Tominaga2



Slika 5.4: Podatkovni skup Tominaga5

U radu (Sirbu et al., 2010) načinjena je usporedba više različitih algoritama evolucijskog računanja pri modeliranju GRN mreže S-sustavom. Korišteni algoritmi dio su EvA2 (Kronfeld et al., 2010) okruženja za evolucijsko računanje. Uspoređeno je sedam algoritama: GA, MOGA (višekriterijski GA), GA+ES, GA+ANN (GA s umjetnim neuronskim mrežama), PEACE1 (Kikuchi et al., 2003), GLSDC (Kimura i Konagaya, 2003) i DE. GA+ANN i DE su se pokazali kao najbolji na stvarnim mrežama.



Slika 5.5: Spellman podatkovni skup

Za usporedbu algoritama korišteno je:

- dobrota najbolje jedinke,
- broj evaluacija potrebnih do najbolje jedinke i
- robusnost na šum.

Ispitivanja su vršena na umjetno generiranim mrežama i stvarnim DNA microarray podacima (Spellman podatkovni skup).

Tablica 5.2: Pregled radova sa primjenom evolucijskog računanja na problem modeliranja GRN mreža

Naziv rada	**Evolucijski algoritam**	**Vrsta modela**	**Funkcija dobrote**	**Podatkovni skupovi (veličina)**
(Kabir et al., 2010)	SA-DE	LTV	pogreška	Generirani (5), E. Coli (6)
(Kimura et al., 2005)	coev-GLSDC	SS	pogreška	Generirani (5,30), Theromophilus (25)
(Sirbu et al., 2010)	GA, DE, GA+ANN, MOGA, PACE1, GLSDC, GA+ES	SS	pogreška	Generirani (10 - 50), Spellman p.s.
(Koduru et al., 2004)	GA-simplex	dif. jed.	višekriterijska	Rice (1)

U radu (Kabir et al., 2010) korišten je linearan vremenski promjenjiv model te algoritam diferencijske evolucije sa samopodešavajućim parametrima. Ispitivanja su vršena na Tominaga5, E.Coli SOS i cAMP podatkovnim skupovima. Također vršena su i ispitivanja uz 5% i 10% šuma. Algoritam je postizao zadovoljavajuće rezultate nakon relativno kratkog vremena izvođenja.

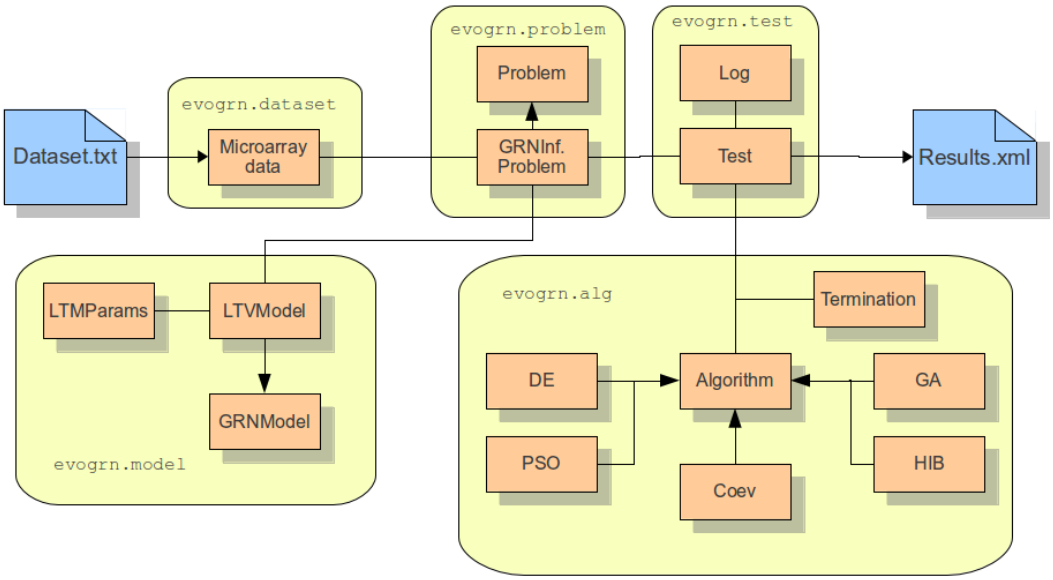
Rad (Kimura et al., 2005) koristi suradničku koevoluciju u kombinaciji s GLSDC algoritmom nad Tominaga5, S-sustavom od 30 gena uz 10% šuma i *Thermus thermophilus* HB8 MAD podatkovnim skupovima. Pristup se pokazao uspješnim u savladavanju dimenzionalnosti problema kod većeg broja gena. Dobiveni rezultati za skup Tominaga5 su otprilike $2 \cdot 10^{-3}$, a pristup se pokazao uspješnijim od uobičajene dekompozicije problema.

Hibridni genetski algoritam u kombinaciji sa simplex metodom korišten je u radu (Koduru et al., 2004). Algoritam se pokazao uspješnijim od uobičajenog višekriterijskog optimizacijskog pristupa SPEA. Nedostatak je što se ispitivanje vršilo za samo jedan gen.

6. Programsko rješenje

Opisani problem modeliranja GRN mreža pomoću evolucijskih algoritama zahtjeva implementaciju četiri različita algoritma te mogućnost korištenja suradničke koevolucije nad svakim od njih. Također, potrebno je moći pokretati algoritme nad različitim podatkovnim skupovima uz zadane parametre. Za navedene potrebe u programskom jeziku Java ostvareno je okruženje pod nazivom *EvoGRN*. Okruženje omogućuje jednostavan odabir i kombinaciju algoritama, podatkovnih skupova te pokretanje testova. Dodatno, nad okruženjem je izgrađeno grafičko sučelje radi lakšeg zadavanja algoritama te rada s rezultatima testova i njihove vizualizacije.

U nastavku poglavlja zasebno su opisane navedene komponente, a shematski prikaz sustava dan je na slici 6.1



Slika 6.1: Shematski prikaz *EvoGRN* okruženja

6.1. *EvoGRN* okruženje

Programsko okruženje podijeljeno je u više paketa:

evogrn.alg: implementacije EC algoritma

evogrn.dataset: rad s MAD podatkovnim skupovima

evogrn.model: modeli GRN mreža

evogrn.problem: opis optimizacijskog problema

evogrn.test: podrška za testiranje algoritama

evogrn.xml: pomoćni paket za lakše zapisivanje rezultata u XML format

Paket evogrn.alg

U korijenu `evogrn.alg` paketa nalaze se apstraktni razredi *Algorithm* i *Individual* pomoću kojih je moguće definirati proizvoljan EC algoritam za rješavanje problema. Uz njih tu su i razredi *Population* i *Termination* koji se koriste u radu algoritama.

Razred *Algorithm* (kod 6.1) je općeniti razred (eng. *generic class*) te je kod definiranja novog algoritma kao parametar potrebno predati razred čijeg su tipa jedinke tog algoritma. Ovaj pristup je korišten da bi se omogućio rad s populacijom općenitih jedinki, a istovremeno specifičan algoritam radi s vlastitom vrstom jedinki, bez potrebe za pretvaranjem tipova.

Algoritam se pokreće metodom *run*(), kojoj se predaje uvjet zaustavljanja (razred *Termination*) i problem koji se rješava (razred *Problem*). Ovdje je vidljivo da se okruženje može iskoristiti za optimiranje proizvoljnog problema.

Kod 6.1: Razred *Algorithm*

```
public abstract class Algorithm<T extends Individual> implements XmlSerializable {

    protected Population<T> population;
    protected T prototype;
    protected Problem problem;

    ...
```

```
        protected abstract boolean runIteration(Termination term, Problem problem);

        public Individual run(Termination term, Problem problem) {

                do {
                        // self−termination of algorithm
                        if (!runIteration(term, problem)) break;
                        ...

                } while (!term.isOver(this, problem));
                return getBestInd();
        }

        public Algorithm(int popSize, T prototype) {
                population = new Population<T>(popSize);
                this.prototype = prototype;
        }

        public void init(Problem problem) {
                this.problem = problem;
                population.init(prototype, problem);
        }

        ...
}
```

Metodi *runIteration*() predaju se i uvjeti zaustavljanja jer izvođenje nekih algoritma može se mijenjati ovisno o u kojoj se vremenskoj fazi nalazi.

Razred *Population* pomoćni je razred za pohranu jedinki. Interno se jedinke čuvaju u strukturi stabla, sortirane prema dobroti. Time je omogućeno brzo dohvaćanje najboljih jedinki, ali je slijedno obilaženje nešto sporije. Iz tog razloga koristi se i red kao dodatna struktura za pohranu. Prebacivanje iz stabla u red vrši se samo ako je došlo do promjene (dodavanje ili uklanjanje jedinki). Ovaj razred služi kao pomoć za ostvarenje algoritama, jer nudi gotove metode za dohvaćanje nasumične potpopulacije, najbolje potpopulacije i slično. Međutim, algoritam može koristiti i vlastiti način pohrane populacije.

Razred *Termination* služi za provjeru uvjeta zaustavljanja. Omogućeni su sljedeći načini zaustavljanja algoritma:

- maksimalan broj iteracija,
- maksimalan broj evaluacija,

– vremensko ograničenje,

– stagnacija najboljeg rješenja i

– zaustavljanje od strane algoritma.

Uvjete je moguće zadati u proizvoljnim kombinacijama, a algoritam će se zaustaviti kada se ispuni barem jedan od njih. Razred također omogućuje provjeru napretka algoritma metodom *percCompleted*().

Implementacija koevolucijskog algoritma

Paket `evogrn.alg.coev` sadrži slijednu implementaciju NPC suradničkog koevolucijskog algoritma (algoritam 7). Nasljeđivanjem razreda *CoevSubProblem* moguće je ostvariti različite pristupe podjeli problema. Razredi *SCPSimple* i *CSPGrnInference* implementiraju prvi i drugi način opisan u poglavlju 5.2. Svakoj populaciji moguće je dodijeliti proizvoljan evolucijski algoritam i zadati mu broj iteracija za svako pokretanje. Najvažniji dijelovi algoritma prikazani su u kodu 6.2.

Kod 6.2: Razred *CoevAlgorithm*

```
public class CoevAlgorithm extends Algorithm<CoevIndividual> {

        ArrayList<Algorithm<?>> algs;
        ArrayList<Individual> bestInds;
        ArrayList<CoevSubProblem> subProblems;

        CoevIndividual best;
        ...

        public CoevAlgorithm(ArrayList<Algorithm<?>> algs, int[] iters, CoevSubProblem
            cspPrototip) {
                ...
        }

        public void init(Problem problem) {

                int id = 0;
                for (Algorithm<?> alg : algs) {
                        CoevSubProblem csp = cspPrototip.getInstance(id++, algs.size(), problem);
                        subProblems.add(csp);
                        alg.init(csp);
                        bestInds.add(alg.getRandomInd());
                }
```

```
                best = new CoevIndividual(popCount, problem, bestInds, subProblems);
                best.evaluate(problem);

                id = 0;
                for (CoevSubProblem csp : subProblems) {
                 csp.setParams(best);
                 algs.get(id++).evaluate(csp);
                }

        }
        ...
        protected boolean runIteration(Termination term,Problem problem) {
         int i = 0;
         CoevIndividual newBest = null;

         for (Algorithm<?> alg : algs) {

                 subProblems.get(i).setParams(best);
                 bestInds.set(i,alg.run(new Termination(iters[i], -1, -1, false) , subProblems.get(i
                     ++)));

                 //stvaranje globalnog rjesenja od najboljih jedinki iz svake podpopulacije
                 newBest = new CoevIndividual(popCount, problem, bestInds, subProblems);
                 newBest.evaluate(problem);

                 if (newBest.isBetter(best)){
                         best = newBest;
                 }
         }

         return true;
        }
        ...
}
```

Paket evogrn.dataset

Paket `evogrn.dataset` sadrži razrede za rad s MAD podatkovnim skupovima. Za pohranu podataka korišten je format sličan onom iz rada (Sirbu et al., 2010), a primjer datoteke se nalazi na slici 6.2. Sa simbolom # započinju linijski komentari. U prvom retku nalaze se parametri N - broj gena i T - broj vremenskih trenutaka. Sljedeći

redak sadrži vremenske trenutke u kojima su dobivena mjerenja ($t_1, ..., t_T$), a potom je u idućih T redaka zapisana izražajnost svakog od gena za dani trenutak. Ukoliko su vremenski trenuci jednoliko razmaknuti moguć je sažeti zapis da se u prvoj liniji kao treći parametar zada razmak između trenutaka (Δt).

```
# Fri May 11 19:00:51 CEST 2012
5 5
0.0 0.05 0.1 0.15 0.2
0.364 0.279 0.923 0.111 0.152
0.854 0.087 0.995 0.998 0.999
0.904 0.820 0.999 0.994 0.999
0.982 0.998 0.999 0.999 0.999
0.998 0.999 0.999 0.999 0.999
```

Slika 6.2: Primjer sadržaja datoteke sa MAD podacima

Razred *MicroArrayData* služi za manipulaciju podacima o izražajnosti gena prilikom ispitivanja algoritama. Ukoliko neka od vrijednosti u skupu prelazi 1, radi se normalizacija skupa, a sve vrijednosti 0 zamjenjuju se vrlo malim vrijednostima 10^{-4}, zbog izraza 5.2.

Paket evogrn.test

Paket `evogrn.test` sadrži razrede *Test* i *Log* koji olakšavaju ispitivanje algoritama. Razred *Log* služi za zapisivanje dnevnika prilikom izvođenja algoritma. Ugrađeno je zapisivanje dobrote najbolje jedinke svakih određen broj iteracija, ali svaki algoritam po potrebi može dodatno pristupati dnevniku.

Razred *Test* omogućuje ispitivanje algoritma s odgovarajućim parametrima za određeni broj pokretanja, zadane podatkovne skupove, parametre modela [1] i uvjete zaustavljanja. Rezultati ispitivanja pohranjuju se u XML formatu. Primjer datoteke prikazan je na slici 6.3. Rezultati pokretanja zapisani su unutar *run* tagova koji se nalaze u tagu *results*. Za svako pokretanje pohranjuje se sadržaj dnevnika te sljedeći statistički podaci populacije iz posljednje generacije:

fit_best: dobrota najbolje jedinke

[1] pod parametre se misli na ograničenja parametara koji se traže algoritmom

fit_worst: dobrota najgore jedinke

fit_avg: prosječna dobrota populacije prema izrazu

$$fit\_avg = \frac{\sum_{i=1}^{N} dobrota(x_i)}{N}$$

fit_dev: standardna devijacija populacije prema izrazu

$$fit\_avg = \sqrt{\frac{\sum_{i=1}^{N}(fit\_avg - dobrota(x_i))^2}{N-1}}$$

Za ukupno testiranje računaju se isti podaci pri čemu se gledaju najbolje jedinke iz svakog pokretanja. Tako primjerice atributi *fit_best* i *fit_worst* unutar taga results označavaju najbolje i najgore rješenje među najboljima iz svakog pokretanja.

Kod 6.3: Razred *Primjer sadržaja datoteke sa rezultatima ispitivanja*

```
<?xml version="1.0" encoding="UTF-8"?>
<test>
  <desc>param-test-2-20 (2 * 300 iterations)</desc>
  <algorithm name="DEAlgorithm">
    <params>
      <param name="pop_size" val="150" />
      <param name="f" val="0.3" />
      <param name="pc" val="0.9" />
      <param name="diff" val="DeBest1" />
      <param name="cross" val="Uniform" />
    </params>
  </algorithm>
  <results iter="2" duration="1.69" fit_best="-0.0761004875283439" fit_worst="
      -0.08582757212024236" fit_avg="-0.08096402982429313" fit_dev="
      0.006878087476106581">
    <run duration="0.895" fit_best="-0.0761004875283439" fit_worst="-0.07635242208789995"
        fit_avg="-0.07611627403742999" fit_dev="2.63413885422544E-5">
      <termination max_iter="300" />
      <log>100: -0,08572031860273788
200: -0,0788709102017673
300: -0,0761004875283439</log>
    </run>
    <run duration="0.737" fit_best="-0.08582757212024236" fit_worst="-0.08598625889259767"
        fit_avg="-0.08589459498016429" fit_dev="4.7347593603111685E-5">
      <termination max_iter="300" />
      <log>100: -0,09020800920541223
200: -0,08975883922423412
300: -0,08582757212024236</log>
    </run>
    <individual size="14" fitness="-0.0761004875283439">9.115070868448944
```

```
−4.175640206775649
0.13548865481796274
−0.36837008661882853
0.04587617785510367
0.6040867547364933
−1.3901012295691435
5.282992407937039
−1.7529869675817638
−1.3568628802816236
1.4057917504416029
−0.9416186229413456
1.5707935768501036
1.5706374329358248</individual>
  </results>
  <problem>
    <LTV−model gene_count="2">
      <params>
        <alphaMin>−20.0</alphaMin>
        <alphaMax>20.0</alphaMax>
        <betaMin>−5.0</betaMin>
        <betaMax>5.0</betaMax>
        <omegaMin>−1.5707963267948966</omegaMin>
        <omegaMax>1.5707963267948966</omegaMax>
        <phiMin>−1.5707963267948966</phiMin>
        <phiMax>1.5707963267948966</phiMax>
      </params>
    </LTV−model>
    <dataset location="../../DataSets/Generated/param−test−2−20.txt" />
  </problem>
</test>
```

Paket evogrn.problem

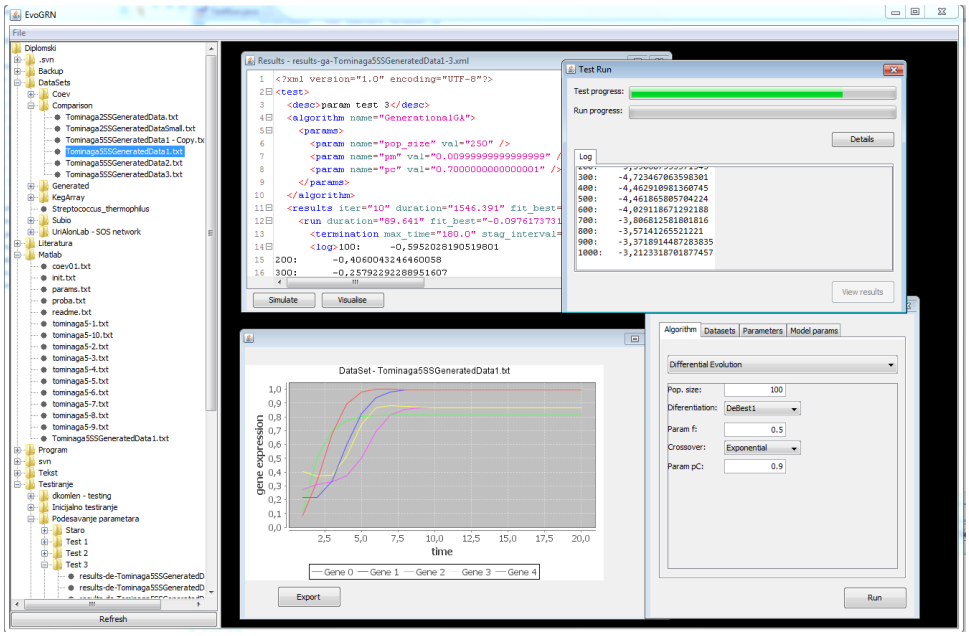
Paket `evogrn.problem` sadrži razred *Problem* koji služi za opisivanje općenitog optimizacijskog problema. Metoda *evaluate*() koristi se za određivanje dobrote jedinkama evolucijskih algoritama. U paketu se nalazi i razred *GRNInferenceProblem* koji opisuje konkretan problem modeliranja GRN mreža te u sebi sadrži podatkovne skupove i GRN model. Kod određivanja dobrote jedinkama, njihovi parametri se predaju modelu, on se simulira i zatim se računa pogreška u odnosu na zadane podatkovne skupove prema izrazu 5.2.

Paket evogrn.model

Paket `evogrn.model` obuhvaća razrede za opisivanje modela GRN mreže. Kao što je ranije spomenuto, on je zajedno s podatkovnim skupovima potreban pri zadavanju problema. Za opisivanje modela koristi se sučelje *GRNModel* čije su osnovne metode *evaluate*() i *simulate*(). Prva služi za određivanje pogreške modela u odnosu na zadane podatkovne skupove, a druga stvara podatkovni skup za zadane početne uvjete. Korišteni linearni vremenski promjenjiv model opisan je razredima *LinearTVModel* i *LTMParams* (sadrži ograničenja za parametre). Razred *LinearTVSubModel* korišten je kod koevolucijskog algoritma.

6.2. *EvoGRN* grafičko sučelje

Glavni motiv za izradu grafičkog sučelja je korištenje okruženja za ispitivanje više algoritama nad više ispitnih skupova, kada se količina podataka počinje gomilati.



Slika 6.3: *EvoGRN* grafičko sučelje

Osnovne funkcionalnosti sučelja su:

- paralelno pokretanje više ispitivanja,
- grafički prikaz podatkovnih skupova,
- prikaz XML datoteka s rezultatima,

– simuliranje najboljeg rješenja iz rezultata,

– grafički prikaz kretanja dobrote po pokretanjima i

– usporedba rezultata.

Grafičko sučelje, izgrađeno radi lakšeg korištenja okruženja, prikazano je na slici 6.3.

Svi alati dostupni su kroz jedini izbornik *File*. U glavnom prozoru sa lijeve strane nalazi se stablo direktorija koje obuhvaća sve direktorije projekta. Dvostrukim klikom na datoteke sa MAD podacima i XML datoteke sa rezultatima pokreće se njihov prikaz unutar programa. Gumb *Refresh* osvježava strukturu direktorija.

Prilikom pokretanja ispitivanja otvara se zasebni prozor u kojem se grafički prikazuje napredak algoritma i ispisuje sadržaj dnevnika. Dodavanje podatkovnih skupova moguće je jednostavnim povlačenjem iz stabla direktorija.

Dobivene rezultate moguće je usporediti na tri načina:

– kretanje dobrote najboljih rješenja,

– grafički prikaz prosječnih dobrota (fit_avg) te

– prikaz izražajnosti odabranog gena.

Kod prikaza prosječnih dobrota rezultate je moguće grupirati u više kategorija, a kod svake usporedbe potrebno je dodijeliti nazive za svaku XML datoteku. Datoteke sa rezutatima također je moguće dodavati povlačenjem.

7. Ispitivanje algoritama

Za svako ispitivanje algoritma potrebni su:

1. parametri algoritma,
2. parametri modela,
3. ispitni skup,
4. uvjeti zaustavljanja i
5. broj ponavljanja.

Korišteni ispitni skupovi opisani su u poglavlju 5.3, a uvjeti zaustavljanja, broj ponavljanja, kao i parametri modela navedeni su posebno za svako od ispitivanja. Parametri algoritama korišteni kod vrednovanja, određeni su nad skupom IS-param, što je objašnjeno u sljedećem poglavlju. Sva mjerenja su vršena na procesorima Pentium 4, 2.6 GHz.

7.1. Određivanje parametara algoritama

Svaki od četiri implementirana algoritma ima vlastiti skup parametara koji određuju njegovo ponašanje za dani ispitni skup. Optimalni parametri određeni su nad ispitnim skupom IS-param uz stagnaciju i vremensko ograničenje kao uvjet zaustavljanja te 10 ponavljanja za svaku vrijednost pojedinog parametra. Algoritam bi se zaustavio ukoliko dobrota najbolje jedinke u 100 iteracija ne poraste najmanje za 0.01 ili dostigne vremensko ograničenje od 3 minute. Za optimalne parametre uzimana je ona vrijednost čije bi ispitivanje dalo najbolju prosječnu vrijednost dobrote najboljih jedinki iz svakog ponavljanja (atribut *fit_avg* u XML datoteci). Prilikom ispitivanja pretpostavljena je nezavisnost parametara te su parametri određivani redom jedan za drugim. Kod svakog ispitivanja korištene su optimalne vrijednosti prethodno ispitanih parametara. Početne

i konačne optimalne vrijednosti parametara dane su u tablicama (7.1 - 7.4). Parametri su ispitivani redom s lijeva na desno kako su navedeni u tablicama.

Prema radu (Kabir et al., 2010) korišteni su sljedeći intervali za parametre modela: $\alpha_{i,j} \in [-20, 20], \beta_{i,j} \in [-5, 5], \omega_i \in [-\frac{\pi}{2}, \frac{\pi}{2}], \varphi_{i,j} \in [-\frac{\pi}{2}, \frac{\pi}{2}]$.

Za veličinu populacije (n) ispitivane su vrijednosti: 50, 100, 150, 200 i 250. Vjerojatnosti križanja (p_c) kod genetskog algoritma i hibridnog Hook-Jeeves GA uzimane su redom: 0.5, 0.6, 0.7, 0.8 i 0.9, a za vjerojatnost mutacije (p_m): 0.02, 0.04, 0.06, 0.08, 0.1 i 0.12. Kod diferencijske evolucije ispitane tri vrste diferencijacije (tablica 3.1) te uniformno i eksponencijalno križanje. Za vrijednosti faktora F korištene su vrijednosti: 0.5, 1, 1.5, 2 i 2.5, za vjerojatnost križanja iste vrijednosti kao kod GA i HIB. Kod algoritma roja čestica za veličinu susjedstva uzimane su vrijednosti: 2, 4, 8, 16 i 32, a za parametre C_1 i C_2 vrijednosti: 0.5, 1, 1.5, 2, 2.5 i 3.

Kod genetskog algoritma za faktor napuhavanja (m) uzeta je vrijednost 2, a kod HIB algoritma $\Delta x_0 = 1$ te $\varepsilon = 10^{-6}$.

Kod koevolucijskog algoritma ispitivana su dva načina podjele problema te broj iteracija osnovnog algoritma koliko se izvršava nad svakom populacijom. Drugi način podjele pokazao se boljim (tablica 7.5), a rezultati za broj iteracija prikazani su u tablici 7.6.

Tablica 7.1: Parametri algoritma GA

	n	p_c	p_m
početni	200	0.8	0.05
optimalni	250	0.8	0.06

Tablica 7.2: Parametri algoritma DE

	n	diferencijacija	križanje	F	p_c
početni	200	DE/best/1	uniformno	0.5	0.9
optimalni	200	DE/best/1	uniformno	0.5	0.7

Tablica 7.3: Parametri algoritma PSO

	n	k	C_1	C_2
početni	200	2	2	2
optimalni	250	8	0.5	1

Tablica 7.4: Parametri algoritma HIB

	n	p_c	p_m	Δx_0	ε
početni	50	0.8	0.05	1	10^{-6}
optimalni	50	0.7	0.1	-	-

Tablica 7.5: Usporedba načina podjele problema kod koevolucijskog algoritma (fit_avg)

	DE	GA	HIB	PSO
prvi način	-1.1254	-0.076895	-4.1002	-6.0824
drugi način	-0.099778	-0.025158	-0.21125	-0.023291

Tablica 7.6: Broj iteracija kod koevolucijskog algoritma

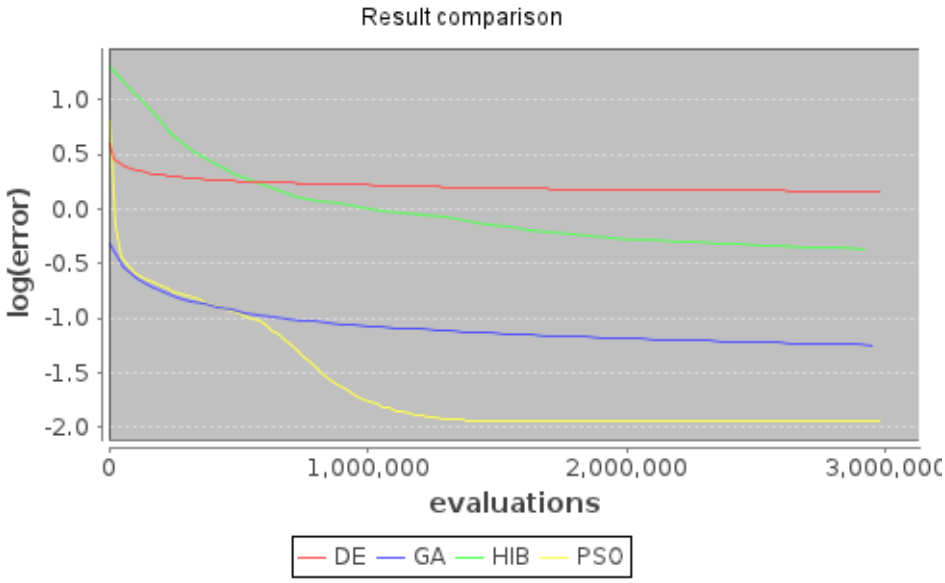
	DE	GA	HIB	PSO
početan	5	5	5	5
optimalan	2	10	2	5

7.2. Rezultati ispitivanja

U svim narednim ispitivanjima korišten je uvjet zaustavljanja od maksimalnih $3 \cdot 10^6$ evaluacija jedinki i vremensko ograničenje od 10 minuta. Svako ispitivanje ponavljano je 50 puta. Rezultati su prikazani grafički, a brojčane vrijednosti su dane u dodatku B.

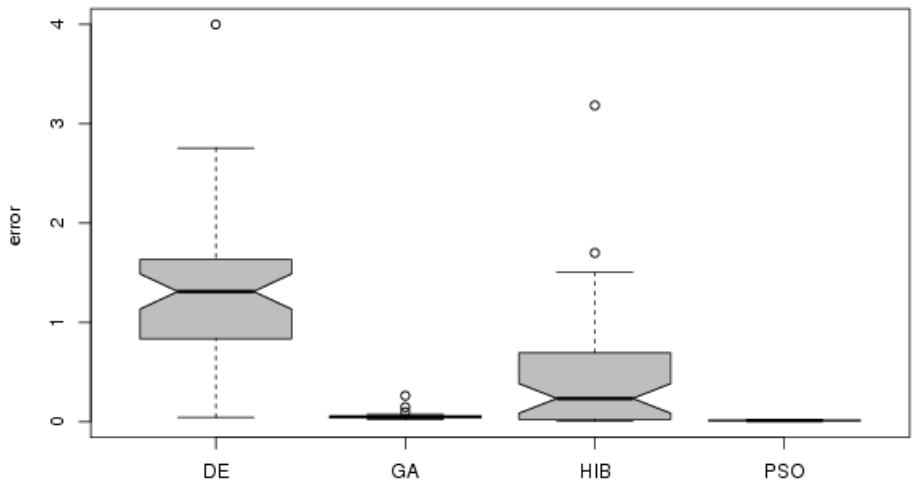
7.2.1. Ispitni skup IS-param

Na slici 7.1 prikazana je prosječna vrijednost najboljih rješenja od svih 50 ponavljanja u ovisnosti o broju evaluacija za podatkovni skup IS-param. Algoritam PSO daje u



Slika 7.1: Kretanje prosječne pogreške za ispitni skup IS-param

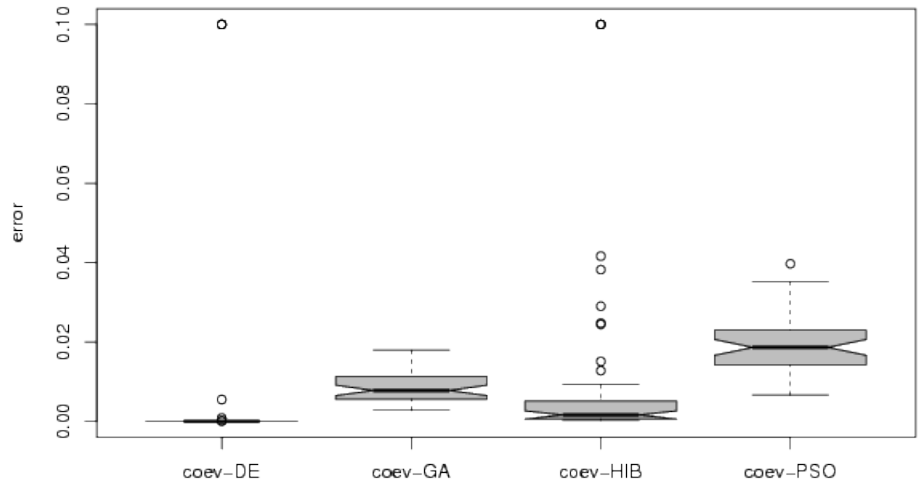
prosjeku najbolja rješenja, a hibridni algoritam je pronašao ukupno najbolje rješenje uz pogrešku 0.0022 (tablica B.1). Za izradu grafa korišteno je *EvoGRN* grafičko sučelje.



Slika 7.2: Usporedba rezultata za ispitni skup IS-param bez koevolucije

Za daljnju usporedbu rezultata korištena je *boxplot* vrsta grafa (Schlitt i Brazma, 2007a). Za izradu je upotrijebljen besplatni alat (Wessa, 2012). Na grafu su označeni medijan, gornji i donji kvartil te minimalna i maksimalna vrijednost koja je unutar in-

tervala veličine $1.5IQR$ od donjeg odnosno gornjeg kvartila. IQR (eng. *interquartile range*) označava razliku gornjeg i donjeg kvartila. S kružićima su označena veća odstupanja (eng. *outlier*). Zbog preglednijeg prikazivanja na nekim grafovima su odbačena odstupanja koja prelaze određenu granicu.



Slika 7.3: Usporedba rezultata za ispitni skup IS-param uz koevoluciju

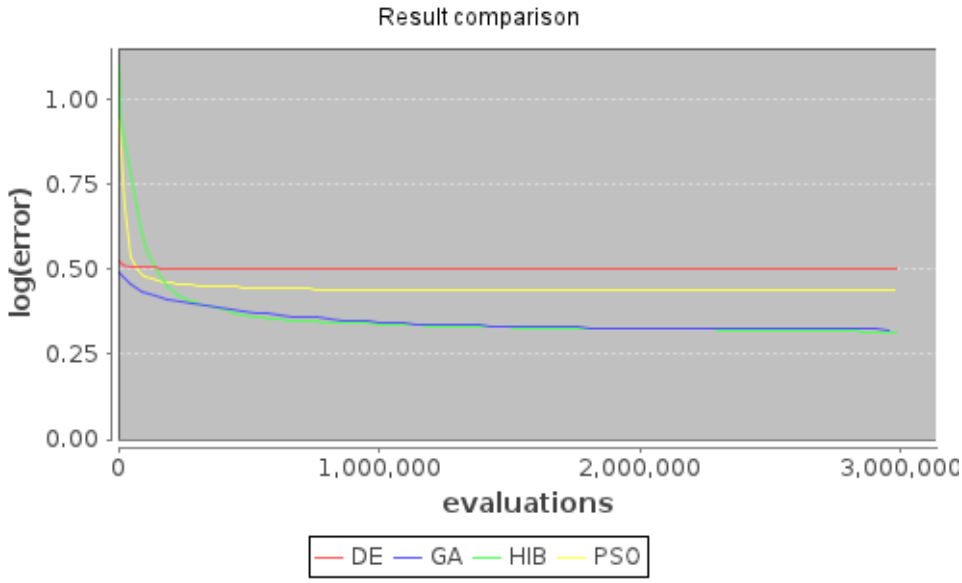
Na slici 7.2 vidljivo je da GA i PSO imaju najmanja, a DE najveća odstupanja u rezultatima.

Iz rezultata s koevolucijom vidljiva su poboljšanja kod svih algoritama osim kod PSO[1]. Najbolje rješenje je pronašao DE algoritam s pogreškom od $2.122 \cdot 10^{-7}$, što je značajno poboljšanje. Također iz slike 7.3 vidljivo je da su rješenja kod DE poprilično ujednačena za razliku od ispitivanja bez koevolucije.

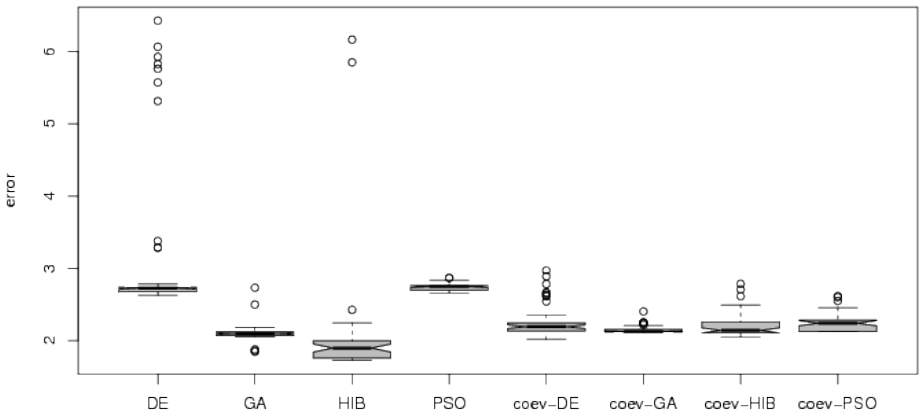
7.2.2. Ispitni skup IS-t2

Najbolju prosječnu pogrešku i najbolje rješenje za ispitni skup IS-t2 ima hibridni algoritam (slika 7.4). Kao i u prethodnom slučaju poboljšanja uz korištenje koevolucije vidljiva su kod svih algoritama osim kod PSO (slika 7.5). Poboljšanja su nešto manja u odnosu na prethodni skup što se može objasniti manjim brojem parametara modela, gdje algoritmi i bez koevolucije daju zadovoljavajuće rezultate.

[1]kod usporedbe algoritama s i bez koevolucije promatrana je prosječna pogreška



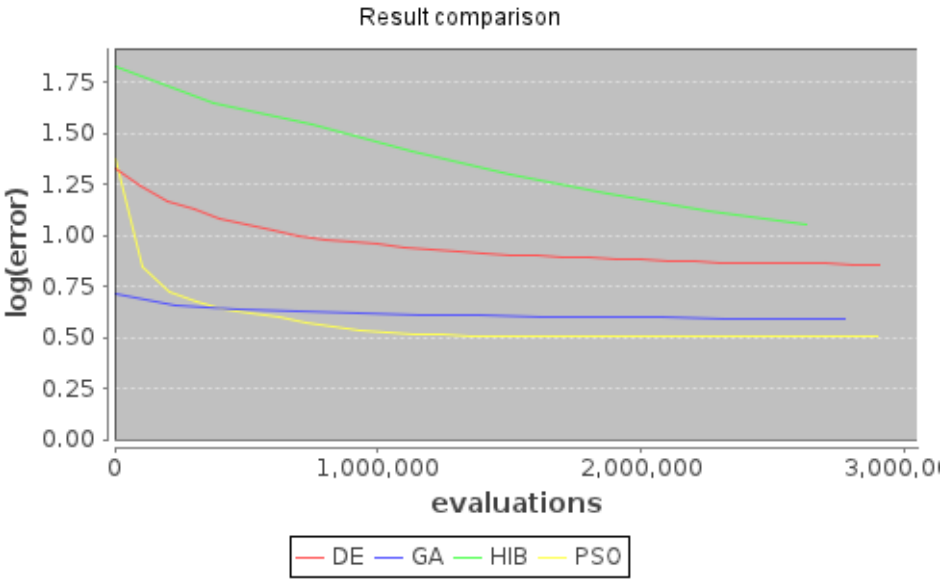
Slika 7.4: Kretanje prosječne pogreške za ispitni skup IS-t2



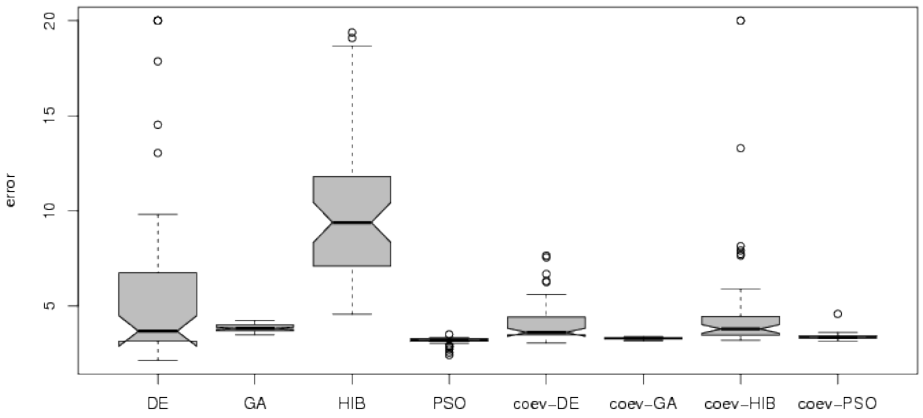
Slika 7.5: Usporedba rezultata za ispitni skup IS-t2

7.2.3. Ispitni skup IS-t5

Ispitni skup IS-t5 ima najviše podatkovnih skupova te iz tog pogleda predstavlja teži problem u odnosu na ostale skupove. U prosjeku najbolje rješenje ima PSO algoritam, a ukupno najbolje rješenje pronašao je algoritam DE. Sa slike 7.6 vidljivo je da pogreška kod hibridnog algoritma najsporije pada, iz čega se može pretpostaviti da bi uz veći broj evaluacija postigao bolje rezultate. Uz primjenu koevolucije poboljšanja se javljaju kod svih algoritama osim kod PSO (slika 7.7).



Slika 7.6: Kretanje prosječne pogreške za ispitni skup IS-t5

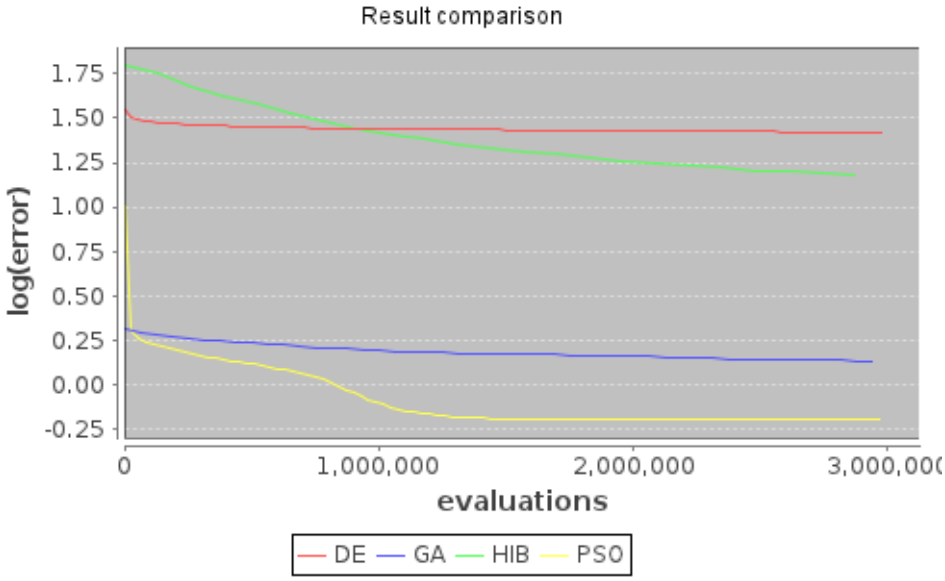


Slika 7.7: Usporedba rezultata za ispitni skup IS-t5

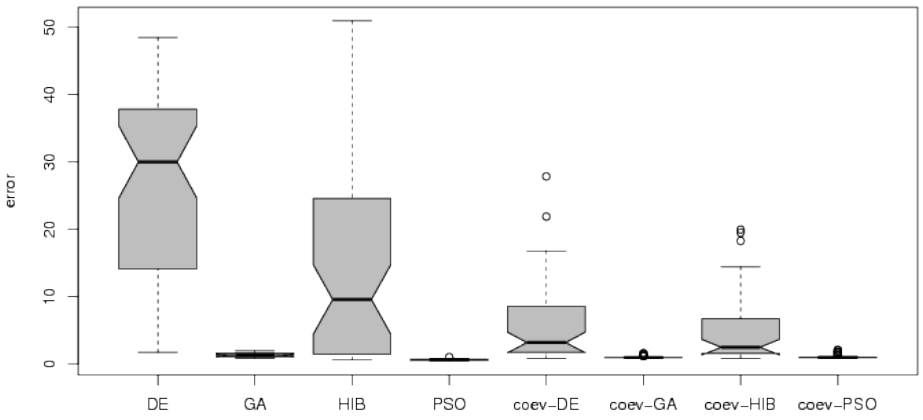
7.2.4. Ispitni skup IS-yeast6

Najbolje rezultate na posljednjem ispitnom skupu ima PSO algoritam. Kao i u prethodnom slučaju hibridni algoritam ima najsporiji pad pogreške (slika 7.8). Zanimljivo je da se prilikom primjene koevolucije i u ovom primjeru poboljšanja javljaju kod svih algoritama osim kod PSO. Jedno od mogućih obrazloženja takvog ponašanja je da se PSO slabije prilagođava stalnim promjenama u prostoru rješenja. Kod koevolucije dobrota svakog rješenja gleda se u ovisnosti o ostalim populacijama, koje se također

mijenjaju. Budući da PSO algoritam sadrži dodatne podatke o brzini kretanja čestice, oni mogu imati negativan utjecaj kod sljedeće iteracije nakon što se ostale populacije promjene.



Slika 7.8: Kretanje prosječne pogreške za ispitni skup IS-yeast6



Slika 7.9: Usporedba rezultata za ispitni skup IS-yeast6

7.2.5. Statistička analiza

Utjecaj koevolucije na rad algoritama dodatno je provjeren primjenom T-testa na rješenja dobivena bez i s koevolucijom. Ovakva vrsta testa daje vjerojatnost da su dva uzorka došla iz dvije iste osnovne populacije, s jednakim srednjim vrijednostima. Za svaki algoritam uspoređivano je 50 rješenja bez i 50 uz korištenje koevolucije za svaki od ispitnih skupova. Rezultati su prikazani u tablici 7.7. Vidljivo je da je većina vrijednosti vrlo mala što upućuje da se radi o rješenjima iz različitih populacija. Kod PSO algoritma populacije su različite, ali na štetu koevolucije (prema prethodno navedenim rezultatima). Također, kod hibridnog algoritma su za dva ispitna skupa vrijednosti nešto veće i teško je generalno zaključiti kolika je u tom slučaju korist od primjene koevolucije.

Tablica 7.7: Rezultati T-testa

	DE	GA	HIB	PSO
IS-param	$4.862 \cdot 10^{-9}$	$1.622 \cdot 10^{-12}$	$2.342 \cdot 10^{-5}$	$3.41 \cdot 10^{-10}$
IS-t2	$3.2 \cdot 10^{-7}$	0.011	0.132	$7.349 \cdot 10^{-36}$
IS-t5	0.053	$2.902 \cdot 10^{-4}$	0.106	$8.262 \cdot 10^{-22}$
IS-yeast6	$4.545 \cdot 10^{-14}$	$1.32 \cdot 10^{-8}$	$2.87 \cdot 10^{-5}$	$2.116 \cdot 10^{-14}$

8. Zaključak

Modeliranje genskih regulacijskih mreža trenutno je aktivno područje istraživanja u području sustavne biologije. Izrada uspješnih modela daje bolji uvid u stanične procese te mogućnost njihovog predviđanja. Algoritmi evolucijskog računanja omogućuju potragu za optimalnim parametrima modela i prema recentnim radovima daju obećavajuće rezultate.

Ostvareno programsko okruženje *EvoGRN* omogućuje ispitivanje DE, GA, PSO i HIB algoritama nad željenim podatkovnim skupovima uz korištenje linearnog vremenski promjenjivog modela. Također, moguća je primjena suradničke koevolucije na svaki od algoritama, a uz pomoć grafičkog sučelja omogućen je lakši rad s većim brojem ispitivanja. Okruženje je lako proširivo s dodatnim algoritmima i vrstama modela za potrebe daljnjih ispitivanja.

Hibridni algoritam pokazao se uspješnim na dva ispitna skupa, dok je iz dobivenih rezultata za preostala dva, moguće pretpostaviti da bi dao bolje rezultate za veći broj evaluacija. Koevolucija je dala poboljšanje kod svih algoritama osim kod PSO-a, za svaki ispitni skup.

Neka od daljnjih mogućih istraživanja uključuju ispitivanje dodatnih algoritama i vrsta modela, dodatna ispitivanja utjecaja parametara algoritama, ispitivanje nad većim podatkovnim skupovima te ispitivanja uz prisutnost šuma.

A. Parametri korištenih podatkovnih skupova

Tablica A.1: Parametri S-sustava za podatkovni skup Tominaga2

i	α_i	$g_{i,1}$	$g_{i,2}$	β_i	$h_{i,1}$	$h_{i,2}$
1	0	2.5	-1	0	3	3
2	-2.5	0	0	2	3	3

Tablica A.2: Parametri S-sustava za podatkovni skup Tominaga5

i	α_i	$g_{i,1}$	$g_{i,2}$	$g_{i,3}$	$g_{i,4}$	$g_{i,5}$	β_i	$h_{i,1}$	$h_{i,2}$	$h_{i,3}$	$h_{i,4}$	$h_{i,5}$
1	5	0	0	1	0	-1	10	2	0	0	0	0
2	10	2	0	0	0	0	10	0	2	0	0	0
3	10	0	-1	0	0	0	10	0	-1	2	0	0
4	8	0	0	2	0	-1	10	0	0	0	2	0
5	10	0	0	0	2	0	10	0	0	0	0	2

B. Tablični prikazi rezultata

Tablica B.1: Rezultati ispitivanja nad ispitnim skupom IS-param

Algoritam	**fit_best**	**fit_worst**	**fit_avg**	**fit_dev**
DE	-0.042639	-10.608	-1.4372	1.4568
GA	-0.021650	-0.26123	-0.054983	0.035873
HIB	-0.0022138	-3.1846	-0.43158	0.58611
PSO	-0.0044652	-0.024634	-0.011275	0.0046827
coev-DE	$-2.1220 \cdot 10^{-7}$	-0.34527	-0.016840	0.062697
coev-GA	-0.0028038	-0.017959	-0.0084812	0.0037957
coev-HIB	-0.00028085	-1.0150	-0.048356	0.19259
coev-PSO	-0.0066508	-0.039721	-0.019383	0.0068116

Tablica B.2: Rezultati ispitivanja nad ispitnim skupom IS-t2

Algoritam	**fit_best**	**fit_worst**	**fit_avg**	**fit_dev**
DE	-2.6242	-6.4273	-3.1730	1.1055
GA	-1.8464	-2.7317	-2.1055	0.12599
HIB	-1.7240	-6.1649	-2.0712	0.82844
PSO	-2.6539	-2.8696	-2.7399	0.050862
coev-DE	-2.0182	-2.9711	-2.2713	0.22220
coev-GA	-2.1124	-2.4036	-2.1512	0.054757
coev-HIB	-2.0472	-2.7855	-2.2062	0.15988
coev-PSO	-2.1226	-2.6163	-2.2402	0.12743

Tablica B.3: Rezultati ispitivanja nad ispitnim skupom IS-t5

Algoritam	fit_best	fit_worst	fit_avg	fit_dev
DE	-2.1378	-39.028	-7.1062	8.0194
GA	-3.4956	-4.2496	-3.8571	0.20422
HIB	-4.5704	-19.378	-10.102	3.7639
PSO	-2.4252	-3.5150	-3.1673	0.19991
coev-DE	-3.0676	-7.6578	-4.1246	1.1100
coev-GA	-3.1603	-3.3999	-3.3010	0.055550
coev-HIB	-3.2055	-25.556	-5.1289	4.1565
coev-PSO	-3.1760	-4.5890	-3.4020	0.20175

Tablica B.4: Rezultati ispitivanja nad ispitnim skupom IS-yeast6

Algoritam	fit_best	fit_worst	fit_avg	fit_dev
DE	-1.7304	-48.465	-26.062	14.006
GA	-0.86012	-1.9609	-1.3443	0.33587
HIB	-0.61023	-50.954	-14.524	14.673
PSO	-0.39690	-1.0462	-0.64015	0.13034
coev-DE	-0.87682	-27.855	-5.8839	5.8841
coev-GA	-0.90135	-1.6290	-1.0185	0.14292
coev-HIB	-0.87280	-19.962	-4.9867	5.3048
coev-PSO	-0.82265	-2.1405	-1.0862	0.30333

C. Popis parametara i oznaka

- N: broj gena u podatkovnom skupu
- M: broj podatkovnih skupova korištenih kod ispitivanja
- T, T_i: broj mjerenja u i-tom podatkovnom skupu
- $X_{k,i}(t)$: izražajnost i-tog gena u trenutku t u k-tom podatkovnom skupu
- $W_{i,j}(t)$: regulacijska matrica
- $Z_i(t)$: regulacijski ulaz LTV modela
- $\alpha_{i,j}, \beta_{i,j}, \phi_{i,j}, \omega_i$: parametri LTV modela
- $\alpha_i, \beta_i, g_{i,j}, h_{i,j}$: parametri S-sustava
- n: veličina populacije kod algoritama
- m: faktor napuhavanja kod GA algoritma
- p_c: vjerojatnost križanja
- p_m: vjerojatnost mutacije
- F: faktor skaliranja kod DE algoritma
- k: veličina susjedstva kod PSO algoritma
- C_1, C_2: faktori utjecaja najboljih rješenja kod PSO algoritma
- w: faktor inercije kod PSO algoritma
- W_{min}, W_{max}: najmanja i najveća inercija kod PSO algoritma
- Δx_0: početni pomak kod Hook-Jeeves GA algoritma
- ε: preciznost kod Hook-Jeeves GA algoritma

POPIS SLIKA

POPIS TABLICA

LITERATURA

Pierre Baldi, G. Wesley Hatfield, i Wesley G. Hatfield. *DNA Microarrays and Gene Expression: From Experiments to Data Analysis and Modeling*. Cambridge University Press, 1 izdanju, 2002.

Larry Bull. Coevolutionary Computation: An Introduction. 2008.

Patrik D'haeseleer, Shoudan Liang, i Roland Somogyi. Genetic network inference: from co-expression clustering to reverse engineering. *Bioinformatics*, 16(8):707–726, 2000.

Robert Hooke i T. A. Jeeves. "direct search" solution of numerical and statistical problems. *J. ACM*, stranice 212–229, 1961.

Mitra Kabir, Nasimul Noman, i Hitoshi Iba. Reverse engineering gene regulatory network from microarray data using linear time-variant model. *BMC Bioinformatics*, 11(Suppl 1):S56, 2010.

Shinichi Kikuchi, Daisuke Tominaga, Masanori Arita, Katsutoshi Takahashi, i Masaru Tomita. Dynamic modeling of genetic networks using genetic algorithm and s-system. *Bioinformatics*, 19(5):643 – 650, 2003.

J. Kim, D. G. Bates, I. Postlethwaite, Heslop P. Harrison, i K. H. Cho. Linear time-varying models can reveal non-linear interactions of biomolecular regulatory networks using multiple time-series data. *Bioinformatics*, 24(10):1286–1292, 2008.

S. Kimura i A. Konagaya. High dimensional function optimization using a new genetic local search suitable for parallel computers. *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, 1:335 – 342, 2003.

S. Kimura, K. Ide, A. Kashihara, M. Kano, M. Hatakeyama, R. Masui, N. Nakagawa, S. Yokoyama, S. Kuramitsu, i A. Konagaya. Inference of S-system models of gene-

tic networks using a cooperative coevolutionary algorithm. *Bioinformatics*, 21(7): 1154–1163, 2005.

Praveen Koduru, Sanjoy Das, Stephen Welch, i Judith L. Roe. Fuzzy dominance based multi-objective ga-simplex hybrid algorithms applied to gene network models. U *GECCO (1)*, stranice 356–367, 2004.

Marcel Kronfeld, Hannes Planatscher, i Andreas Zell. The eva2 optimization framework. U *Proceedings of the 4th international conference on Learning and intelligent optimization*, stranice 247–250, Berlin, Heidelberg, 2010. Springer-Verlag.

Sean Luke. Essentials of metaheuristics. stranice 103–125, 2009a.

Sean Luke. Essentials of metaheuristics. stranice 34–44, 2009b.

Marco A. Montes De Oca, Thomas Stützle, Mauro Birattari, i Marco Dorigo. Frankenstein's pso: A composite particle swarm optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 2009.

S. K. Pal, S. Bandyopadhyay, i S. S. Ray. Evolutionary computation in bioinformatics: a review. 36(5):601–615, 2006.

Kenneth V. Price. An introduction to differential evolution. stranice 79–108, 1999.

Thomas Schlitt i Alvis Brazma. Current approaches to gene regulatory network modelling. *BMC Bioinformatics*, 8(Suppl 6):S9+, 2007a.

Thomas Schlitt i Alvis Brazma. Current approaches to gene regulatory network modelling. *BMC Bioinformatics*, 8(Suppl 6):S9+, 2007b.

Alina Sirbu, Heather Ruskin, i Martin Crane. Comparison of evolutionary algorithms in gene regulatory network model inference. *BMC Bioinformatics*, 11(1):59, 2010.

R. Storn i K. Price. Differential Evolution- A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical report, 1995.

D. Tominaga, M. Okamoto, Y. Maki, S. Watanabe, i Y. Eguchi. Nonlinear Numerical Optimization Technique Based on a Genetic Algorithm for Inverse Problems: Towards the Inference of Genetic Networks. U *German Conference on Bioinformatics (GCB)*, 1999.

Patric Wessa. Notched boxplots (v1.0.6), 2012. URL `http://www.wessa.net/rwasp_notchedbox1.wasp/`.

Modeliranje genske regulacijske mreže pomoću hibridnog koevolucijskog algoritma

Sažetak

Modeliranje genskih regulacijskih mreža trenutno je aktivno područje istraživanja iz sustavne biologije. Napredak na tom području omogućuje bolje razumijevanje staničnih procesa i njihovu kontrolu. Algoritmi evolucijskog računanja primjenjuju se za pronalaženje optimalnih parametara modela. U radu je dana usporedba četiri algoritma: DE, GA, PSO te Hibridni Hook-Jeeves GA uz korištenje linearnog vremenski promjenjivog modela. Zbog porasta broja parametara kod mreža s većim brojem gena primijenjena je suradnička koevolucija uz dvije vrste podjele problema. Nakon odabira parametara za svaki od algoritama, ispitivanja su vršena nad dva umjetno generirana i jednim stvarnim podatkovnim skupom. Primjena koevolucije pokazala se uspješnom kod svih algoritama osim kod PSO. U radu su opisani korišteni algoritmi i modeli, dan je kraći pregled koevolucijskih algoritama te je opisano implementirano *EvoGRN* programsko okruženje.

Ključne riječi: genske regulacijske mreže, hibridni genetski algoritam, suradnička koevolucija, linearan vremenski promjenjiv model, S-sustavi

Inference of gene regulatory networks using hybrid coevolutionary algorithm

Abstract

Inference of gene regulatory networks is currently an active field of research in system biology. Progress in this area allows better understanding of cellular processes and their control. Evolutionary computation algorithms are applied for finding the optimal parameters of models. The paper presents a comparison of four algorithms: DE, GA, PSO and the Hybrid Hook-Jeeves GA using linear time-variant model. Due to the increased number of parameters in networks with a large number of genes, cooperative coevolution with two types of problem divisions, was applied. After selecting the optimal parameters for each of the algorithms, tests were performed on two artificially generated and one real microarray data set. The application of coevolution proved to be successful in all algorithms except for PSO. The paper describes used algorithms and models, gives a brief overview of coevolutionary algorithms and describes the implemented *EvoGRN* framework.

Keywords: gene regulatory networks, hybrid genetic algorithm, cooperational coevolution, linear time-variant model, S-systems