Оптимизација роjевима – колонијом пчела

(енīл. Artificial Bee Colony Optimization Algorithm)

Катарина Димитријевић и Мирјана Јочовић

Сажетак

У овом раду демонстрирамо оптимизацију роjевима, конкретно применом *Artificial Bee Colony Optimization(ABC)* алгоритма. Приказан је рад алгоритма на два проблема, први проблем је минимизација континуалних функција, док је други оптимизација дискретних проблема, конкретно проблема суме подскупова. Резултат рада алгоритма је проналажење глобалног минимума (или приближног решења). У раду је приказано и поређење перформанси рада овог алгоритма са перформансама других алгоритама који решавају ове проблеме.

1 Увод

Artificial Bee Colony алгоритам се бави решавањем оптимизационих проблема, ти проблеми могу имати ограничења, али не морају.
Овај алгоритам први је описао Karaboga 2005. године у свом раду. Алгоритам се заснива на понашању роја пчела при потрази за храном [1] . Прво је дефинисан за решавање нумеричких проблема без ограничења. Касније, 2007. године Karaboga и Basturk су поредили перформансе ABC алгоритма са другим такође хеуристичким алгоритмима (генетским алгоритмом, алгоритмом оптимизације роjевима …) [2]. Након тога је настала верзија овог алгоритма за решавање проблема са ограничењима.

2 Проблем

Као што је речено, овај алгоритам се заснива на понашању пчела у природи, стога прво следи опис понашања пчела у природи.

2.1 Опис понашања пчела у природи

Самоорганизација роја пчела је заснована на неколико једноставних правила која поштује свака пчела. Свака пчела одлучује да ли ће да истражује извор хране пратећи пчеле које су већ истражиле тај извор. Свака кошница има тзв. плесни подијум где пчеле које су већ истражиле изворе хране „плешу“ тако да покушавају да убеде друге пчеле да их прате. Ако нека пчела одлучи да напусти кошницу и крене у потрагу за храном, она прати једну од пчела које су плесале на подијуму. Након што се пчела врати из потраге за храном има три могућности:

а) напуштање тог извора хране и враћање у кошницу
б) настављање истраживања тог извора хране без позивања других пчела да је прате
в) настављање истраживања тог извора хране уз повремени повратак у кошницу ради позивања других пчела да је прате

г) У природи није баш најјасније како се пчеле одлучују коју пчелу са плесног подијума ће да прате, али се претпоставља да је то у вези са квалитетом извора хране, који пчеле описују кроз плес на неки начин.

2.2 Опис имплементације алгоритма

У ABC алгоритму постоје 3 врсте пчела:

1. Employed bees
2. Onlookers
3. Scouts

Половина роја пчела су e*mployed bees*, док друга половина укључује o*nlookers.* Прве групе пчела има исто колико и извора хране. Ти извори хране представљају потенцијална решења и идеја је да пчеле својим тргањем пронађу најквалитетнији извор (оптимално решење или њему блиско). Извори хране су инстанце класе која има три основна поља, позиција решења, фитнес решења, и објективна вредност функције циља решења који се у зависности од конкретног проблема бирају.
Алгоритам се састоји из три фазе, тако да у свакој учествује одређена група пчела:

1. Employed bee phase
2. Onlooker bee phase
3. Scout bee phase

Employed bee phase

Као што је речено, employed пчела има исто колико и потенцијалних решења. У овој фази свака од employed пчела у петљи истражује тачно једно решење, тј његову околину тако што мало мења његову текућу вредност.
Ново решење се добија тако што се текуће измени уз помоћ насумично одабраног партнера и насумичне вредности из одређеног скупа и врши се интерполација између њих, док вредност партнера остаје непромењена. Новодобијено решење ће заменити старо само уколико има већи фитнес од њега. За свако решење се води евиденција колико пута се његовом изменом добило лошије решење *(trial).*

Onlooker bee phase

На почетку ове фазе се за свако решење рачунају вероватноће са којима ће то решење бити истражено на основу његовог квалитета (тј. фитнеса) . Постоји више начина израчунавања ових вероватноћа, најчешће се користи класична дефиниција вероватноће:

$$p_i = \frac{fit_i}{\sum_{n=1}^{N} fit_n} \quad \text{i = 1, 2 , … N – број решења (извора)} \quad (1)$$

Свака од onlooker пчела у петљи бира које решење ће истражити на основу израчунатих вероватноћа, при чему се истраживање решења врши на исти начин као и у претходној фази. Петља се неће зауставити док овај услов не буде испуњен, што води ка томе да нека од решења неће бити истражена од стране ове групе пчела док ће нека решења бити истражена више пута.

Scout bee phase

Разматрају се *trial* вредности свих решења и одређује се које од њих је веће од унапред задате границе.

- Уколико постоји само једно такво решење тада се то решење мења новим насумично креираним решењем
- Уколико постоји више таквих решења која имају међусобно једнаке вредности поља *trial* насумично се бира једно од њих и замењује новим насумично креираним решењем
- Уколико постоји више таквих решења који имају међусобно различите вредности поља *trial* се оно са највећом вредности и замењује новим насумично креираним решењем

ABC алгоритам се извршава у више итерација чији број се прослеђује алгоритму као параметар, а свака од тих итерација се састоји из три изнад наведене фазе.

3 Експериментални резултати

Рад ABC алгоритама је тестиран на два проблема. Први је минимизација континуалне функције, док је други оптимизација дискретног проблема – минималан број новчића.

3.1 Минимизација континуалне функције

Проблем минимизације је објашњен на неколико различитих функција: Rastrigin, Rosenbrock (које представљају примере функција које је веома тешко минимизовати), као и на неким једноставнијим функцијама (нпр. конвексна функција која има само један минимум и он глобални).
У свим овим ситуацијама коришћен је исти начин кодирања проблема: позиција решења је кодирана као листа чији су елементи одговарајуће координате позиције решења, за функцију циља узета је сама функција која се минимизује, а вредност фитнеса се рачуна на основу вредности функције циља најчешће на следећи начин:

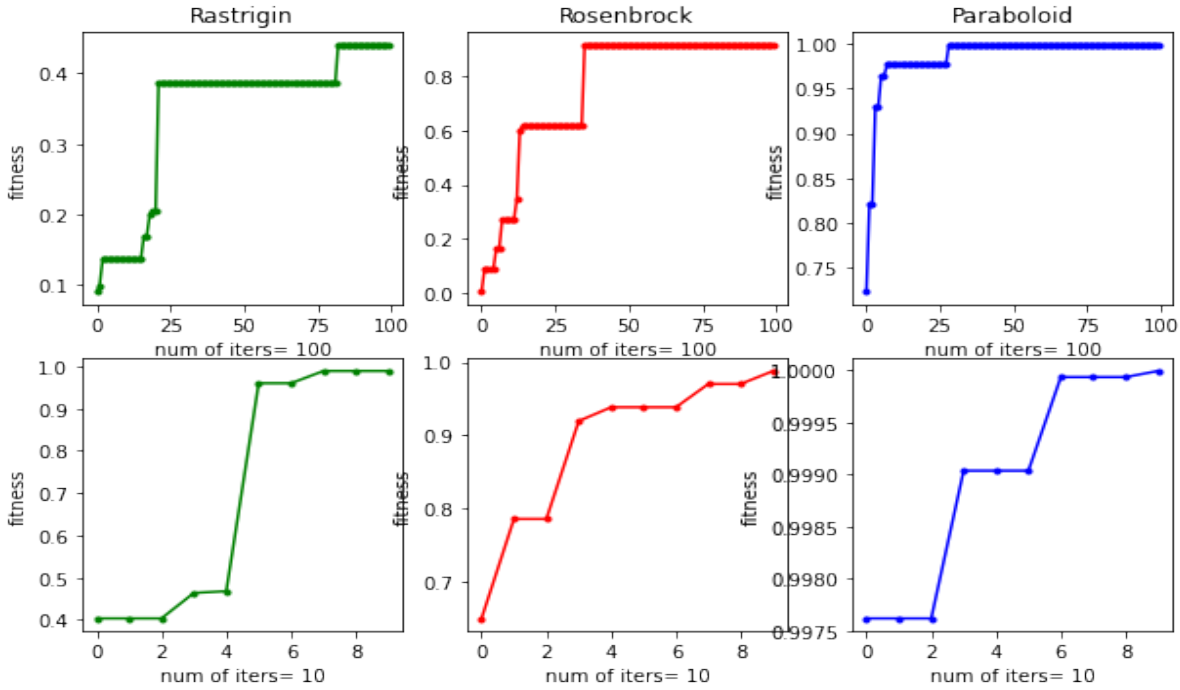
$$fit = \begin{matrix} \frac{1}{1+f}, f \geq 0 \\ 1+|f|, f < 0 \end{matrix} \quad (2)$$

Главни циљ је минимизовати задату функцију, тако да се потенцијална решења међусобно пореде на основу своје фитнес вредности и фаворизују се она са већом вредношћу.

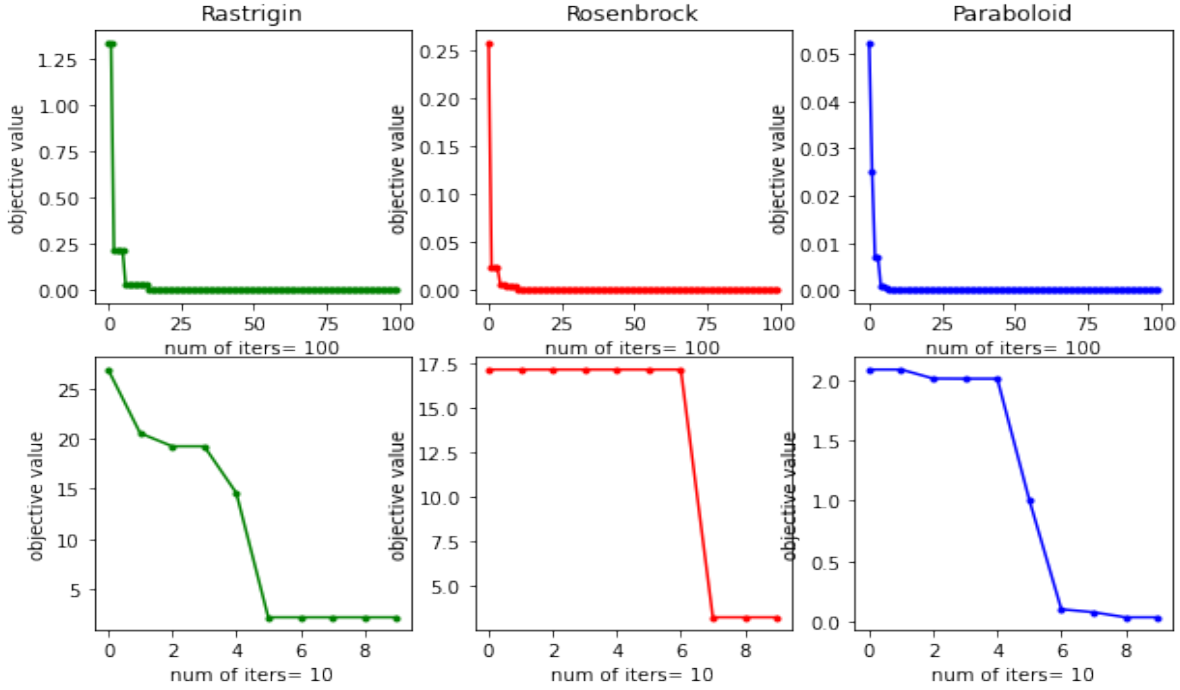
Резултати извршавања ABC алгоритма у зависности од прослеђених параметара

Резултат извршавања овог алгоритма зависи од вредности прослеђених параметара. Њихов одабир се врши искључиво на основу експерименталних резултата и претходног искуства и могу знатно да утичу на брзину извршавања. Параметри од значаја су: limit (раније споменут), population_size (број потенцијалних решења у свакој итерацији) i iteration_size (број итерација).

У наставку следи графички приказ промене вредности фитнеса кроз итерације - постепено расте док се не достигне тражени минимум или њему блиска вредност (слика 1), а након тога приказ промене вредности функције циља кроз итерације - постепено опада док се не достигне тражени минимум или њему блиска вредност (слика 2). Може се приметити да за лакше функције фитнес почиње у ранијим итерацијама да расте и брже достигне циљ.



Слика 1: Приказ промене фитнеса кроз итерације за различите функције, респективно: Rastrigin, Rosenbrock i paraboloid



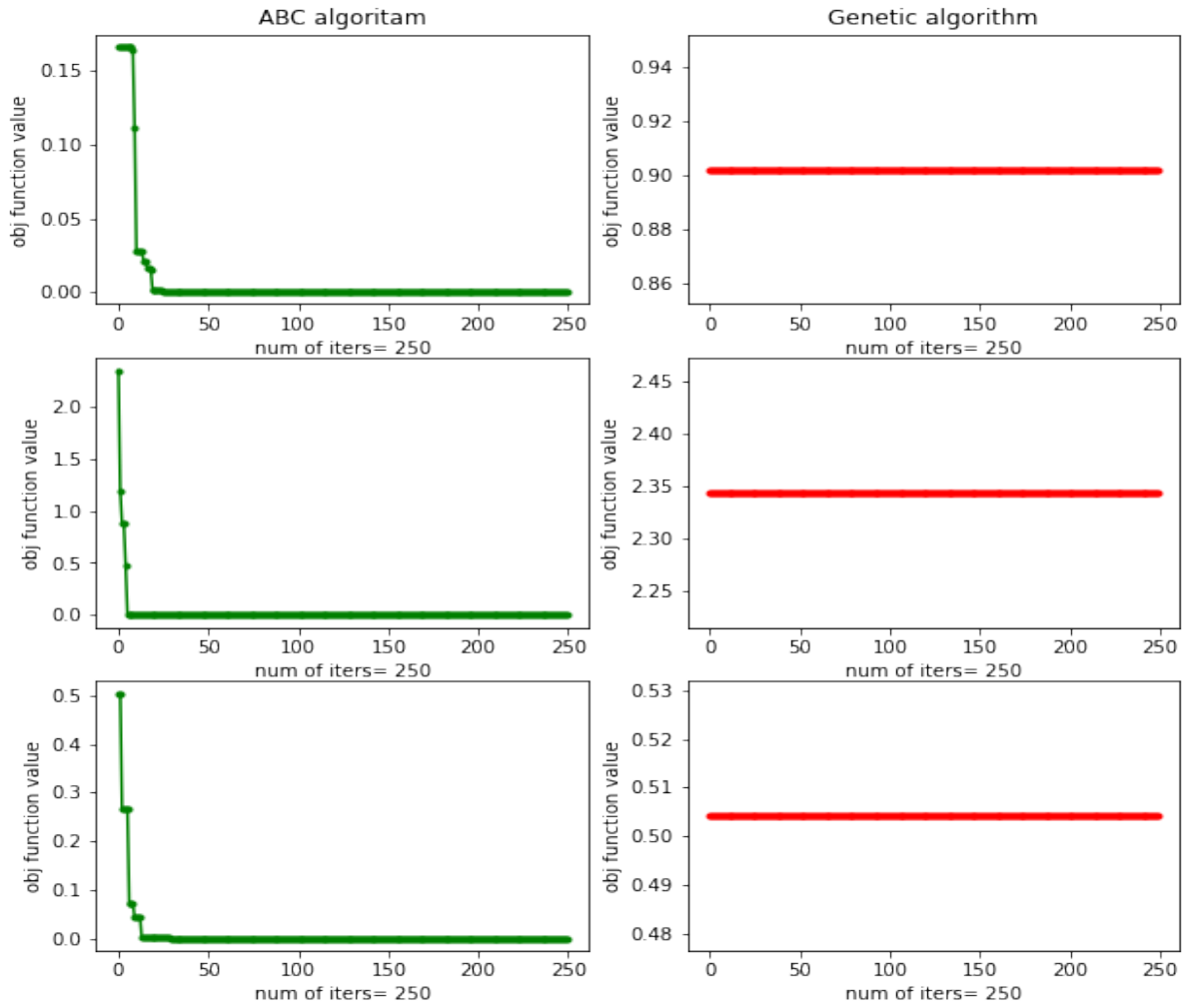
Слика 2: Приказ промене функције циља кроз итерације за различите функције, респективно: Rastrigin, Rosenbrock i paraboloid

Може се приметити да број итерација знатно утиче на квалитет пронађеног решења. Мањи број итерација не мора увек да доведе до потпуно тачног решења, чак некад не пронађе ни приближно тачно решење. То зависи од тежине задатог проблема, за теже проблеме је обично

потребно више итерација. На брзину проналажења решења и његов квалитет утичу и други фактори попут величине популације или почетног решења које се бира насумично.

Поређење перформанси ABC и генетског алгоритма (GA)

ABC алгоритам, попут генетског, спада у групу хеуристичких алгоритама, прецизније P metaheuristika. То су алгоритми који нису настали како би решавали конкретан проблем, већ се користе за решавање веома разноврсног скупа проблема у ситуацијама када егзактни алгоритми тих проблема не дају добре резултате (у контексту времена извршавања) или их није могуће променити због временских и меморијских прекорачења.
Поређење ова два алгоритма има смисла јер оба користе сличну логику. Квалитет решења се мери помоћу вредности фитнеса која се израчунава на исти начин (2) .
Не може се поуздано тврдити који од ова два алгоритма има боље перформансе – то зависи од конкретног проблема и вредности параметара. У проблему минимизације континуалне функције са две променљиве се експериментално показује да боље перформансе, у смислу брзине проналажења решења и његовог квалитета , даје ABC алгоритам. У наставку следи графички приказ понашања функција циља ова два алгоритма у зависности од броја итерација, тако да почетно решење у оба случаја има исту вредност (слика 3).



Слика 3

На графицима су приказане вредности функције циља за исти број итерација – 250, али се у свакој разликује бројност популације. Јасно се може уочити да ABC проналази решење

веома брзо – већ за око десетак итерација, и оно је веома прецизно, док GA даје лоша решења и не напредује кроз генерације (итерације). Узрок томе може бити посматрање функција са само две променљиве што доводи до проблема у укрштању. Перформансе GA би се знатно повећале ако би се посматрале функције са великим бројем променљивих.

3.2 Минималан број новчића

Демонстрација рада ABC алгоритма у претходној глави показује да алгоритам ради на исправан начин и да има веома добре перформансе. Ради потпуности тестиран је и рад алгоритма на дискретном проблему, а то је проблем минималног броја новчића.
Формулација овог проблема је да на располагању имамо новчиће одређених апоена и да нам је дат један цео број који представља циљну суму. Задатак је од доступних новчића формирати циљну суму тако да број употребљених новчића буде минималан, са ограничењем колико новчића сваког апоена се може употребити. Овако дефинисан проблем у литератури се назива *проблем суме подскупа*, за који се зна да је NP тежак проблем [4].
Стога, овај проблем се решава ABC алгоритмом не би ли се уверили да је алгоритам способан да реши и теже проблеме од минимизације функције.

Проблем је кодиран тако да је позиција решења листа чији су елементи цели бројеви који за сваки од апоена новчића говоре колико новчића тог апоена је употребљено у том решењу. Функција циља се рачуна по испод наведеној формули (3). Уз помоћ *alpha* и *beta* параметара могу се скалирати сабирци, а начин одабира ових параметара утиче на квалитет рада алгоритма, што ће бити детаљно објашњено касније, док се вредност фитнеса рачуна на основу вредности функције циља по формули (2).

$$alpha * \left| target - \sum_i values_i * position_i \right| + beta * \sum_i position_i \quad \text{, i = 1, … , N – бр доступних апоена} \qquad (3)$$

Потенцијална решења се међусобно пореде на основу своје фитнес вредности и фаворизују се она са већом вредношћу.

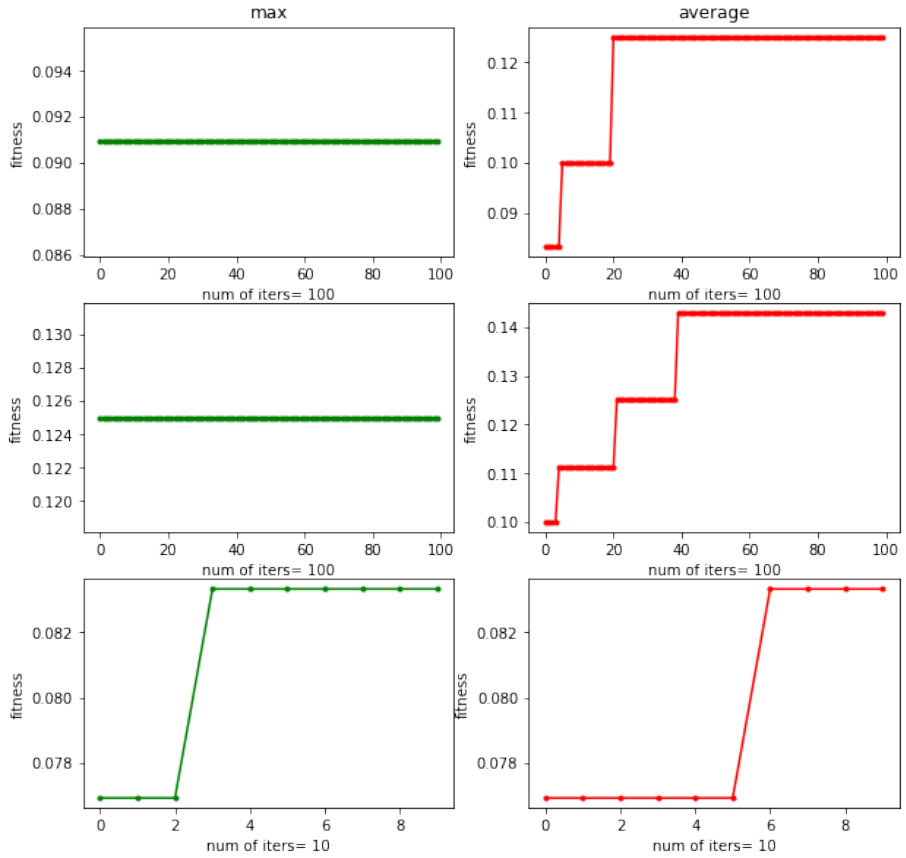
Резултати извршавања ABC алгоритма у зависности од прослеђених параметара

Као што је наведено у претходној глави, резултат извршавања алгоритма зависи од прослеђених параметара. Начин одабира тих параметара је исти као и у претходној глави, као и то који су параметри од значаја.

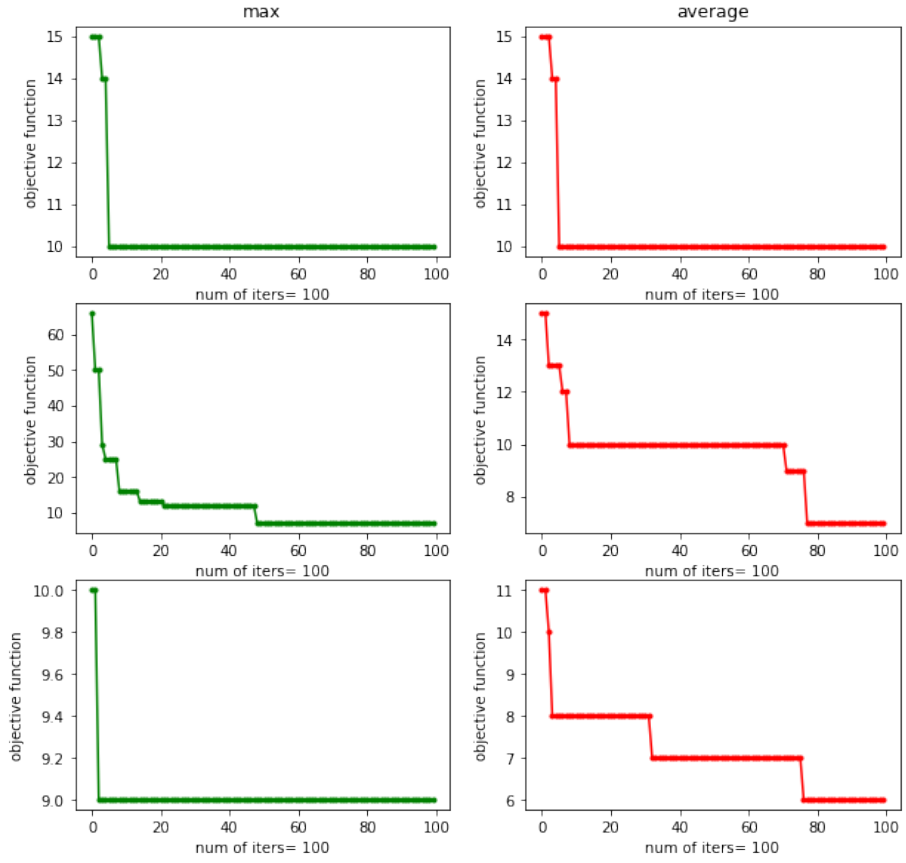
У наставку следи графички приказ промене вредности фитнеса и промене вредности функције циља кроз итерације (слике 4 и 5) у више различитих покретања, за различите прослеђене параметре.

Начин на који број итерација и величина популације утичу на квалитет рада алгоритама је исти као у претходној глави, овде ћемо обрадити како утиче начин креирања новог решења од постојећег на квалитет рада алгоритма.
Имплементирана су два могућа начина за креирање новог решења од постојећег, *average* и *max*. Average начин креира ново решење тако што нову вредност координате, која је изабрана да се мења, рачуна одређујући просек одговарајућих координата старог решења и одабраног партнера. Док max начин бира већу од тих координата.

Може се приметити да се постиже већа фитнес вредност када се користи average начин, која постепено расте, тј. мања вредност функције циља која постепено опада, док уколико се користи max начин у неким покретањима се фитнес, тј функција циља практично не мењају.



Слика 4

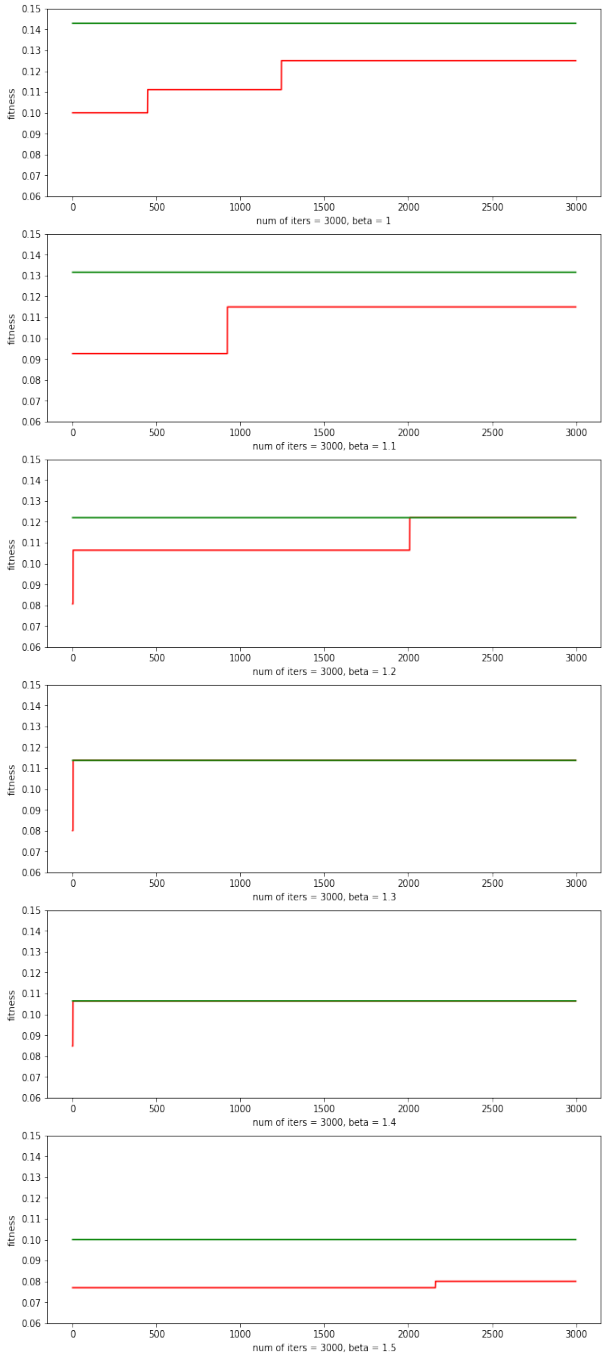


Слика 5

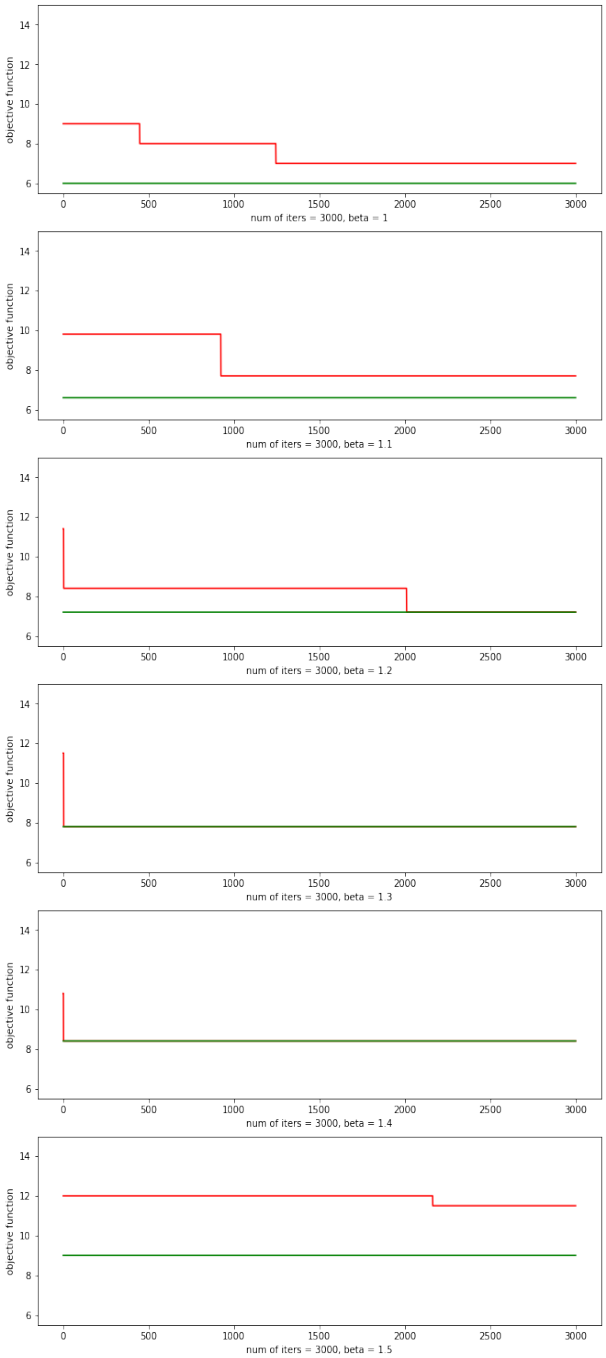
Поређење брзине проналажења и квалитета решења у зависности од начина рачунања вредности функције циља

Функција циља се рачуна по формули (3). Као што је наговештено, начин одабира *alpha* и *beta* параметара утиче на брзину проналажења и квалитет решења. Параметар *alpha* одређује колико се цени да текуће решење даје тачну циљну суму, док параметар *beta* одређује колико се цени да текуће решење користи што мањи број новчића.

У наставку је дат графички приказ како одабир *beta* параметра утиче на брзину проналажења и квалитет решења (слике 6 и 7).



Слика 6



Слика 7

Идеја тестирања понашања алгоритма променом *beta* параметра потиче од тога да алгоритам лакше проналази тачно решење него што то ради са минималним број новчића, стога се постепено повећава *beta* параметар и прати се како се алгоритам понаша. У обе групе графика поред праћења брзине раста односно опадања фитнеса и вредности функције циља, може се пратити колико су те вредности далеко од идеалних вредности за то покретање алгоритма. Наиме, захваљујући алгоритму динамичког програмирања које може да пронађе потпуно тачно решење (више о томе у наредној секцији) могу се израчунати идеалне фитнес вредности и вредности функције циља у том покретању, и мерити колико се вредности које проналази ABC алгоритам разликују од њих (зеленом линијом су приказане идеалне вредности, а црвеном вредности које проналази ABC алгоритам).
Из приказаних графика можемо уочити да се ABC алгоритам постепено боље понаша како вредност параметра *beta* расте. Вредности брже расту, тј. опадају, а видимо да у покретањима када је *beta*=1.3 и *beta*= 1.4 алгоритам досеже идеалне вредности практично одмах. Када је *beta*=1.2 алгоритам такође досеже идеалне вредности али нешто касније него у претходна два случаја, док у осталим случајевима не успева да досегне идеалне вредности. Након вредности *beta*=1.4 алгоритам почиње да има лошије перформансе, јер почиње превише да цени мали број новчића, а да занемарује тачност решења, па самим тим ни не успева да од задатих новчића формира циљну суму ни на један начин.

Поређење перформанси ABC алгоритма и алгоритма динамичког програмирања

Као што је већ речено, ABC алгоритам се користи у ситуацијама када није могуће употребити егзактан метод за решавање одређеног проблема, у контексту времена извршавања или меморијског заузећа.
Проблем минималног броја новчића се може решити егзактном методом – динамичким програмирањем, које само по себи не спада у brute-force алгоритме, већ проблем решава на мало паметнији начин, али ипак у одређеним ситуацијама и даље не може решити проблем.

За мање улазе, а то су они код којих се креира мања матрица динамичког програмирања, тј. они код којих је циљна сума мања и број доступних апоена новчића мањи, динамички алгоритам ради сасвим задовољавајуће. Чак и изгледа да је бољи од ABC алгоритма, јер враћа решење доста брже од њега које је у потпуности тачно, док ABC у неким покретањима ни не враћа потпуно тачно решење, већ неко приближно.
Резултат извршавања програма у тој ситуацији (дин. матрица димензије 75x5):

```
Execution time of the abc algorithm:  21.7 sec
Execution time of the dynamic algorithm:  0.005 sec
```

За веће улазе, а то су они код којих се креира већа матрица динамичког програмирања, тј они код којих је циљна сума већа и број доступних апоена новчића већи, динамички алгоритам је неупотребљив. Време извршавања ABC алгоритма остаје приближно исто као и у ситуацијама када су улази, док време извршавања динамичког алгоритма страшно расте, или чак програм пуца услед недостатка RAM меморије.
Наиме, на примеру у ком се креира матрица динамичког програмирања димензије 50000x26000 излаз програма је:

```
Execution time of the acb algorithm:  29.276 sec
Execution time of the dynamic algorithm:  1201.933 sec
```

За овај пример се може рећи да је последњи за који програм не „пуца“, за било који већи улаз програм не успева да се изврши, тј. пуца услед недостатка RAM меморије.

4 Закључак

На основу изнад описаних експеримената може се закључити да ABC алгоритам успешно проналази минимум континуалних функција, чак и оних изузетно тешких за минимизацију. Такође, из примера са минималним бројем новчића, закључује се да алгоритам успешно решава и NP тешке проблеме.

5 Литература

(1) D. Karaboga, An Idea Based On Honey Bee Swarm For Numerical Optimization, Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

(2) B. Basturk, D.Karaboga, An Artificial Bee Colony (ABC) Algorithm for Numeric function Optimization, IEEE Swarm Intelligence Symposium 2006, May 12- 14, 2006, Indianapolis, Indiana, USA.

(3) BEE COLONY OPTIMIZATION – A COOPERATIVE LEARNING APPROACH TO COMPLEX TRANSPORTATION PROBLEMS Dušan TEODOROVIĆ,2, Mauro DELL' ORCO URL: https://neuro.bstu.by/ai/To-dom/My_research/failed%25201%2520subitem/For-courses/Job-SSP/Bee/ID161%255B1%255D.pdf

(4) Wikipedia URL: https://en.wikipedia.org/wiki/Subset_sum_problem

(5) Materijali sa predmeta „Algoritmi i strukture podataka“ prof. Filipa Marica URL: http://poincare.matf.bg.ac.rs/~filip/asp/asp.pdf