Evolucijska arena

Borko Bošković, Janez Brest, Damjan Casar, Viljem Žumer

Inštitut za računalništvo
Fakulteta za elektrotehniko, računalništvo in informatiko
Univerza v Mariboru
Smetanova ul. 17, 2000 Maribor, Slovenija

Evolutionary Arena

The paper presents the Evolutionary Arena, a program for testing evolutionary algorithms. Already defined problems in form of shared libraries from CEC Special Sessions/Competition are extended and a protocol for communication between graphical user interface and evolutionary algorithm is defined. We implemented a prototype application that is designed as a graphical user interface implemented with the Qt library. The application uses extended shared libraries and a newly defined protocol. These enable users to run and analyse different evolutionary algorithms on different problems.

1 Uvod

V tem prispevku predstavljamo način za poenoteno analizo in primerjavo evolucijskih algoritmov za reševanje enokriterijskih optimizacijskih problemov. V ta namen smo izdelali prototipno aplikacijo, ki smo jo poimenovali Evolucijska arena. Načrtovali smo jo tako, da je znotraj nje možno zaganjati različne evolucijske algoritme, ki so lahko implementirani v poljubnem programskem jeziku. Aplikacija omogoča tudi nalaganje različnih problemov v obliki deljenih knjižnic (ang. *Shared Library*). Tako izdelana aplikacija je poleg primerjave in analize evolucijskih algoritmov primerna tudi za uporabnike, ki niso eksperti na področju evolucijskih algoritmov. Ti uporabniku lahko rešujejo poljubne enokriterijske probleme s pomočjo poljubnega evolucijskega algoritma.

Želeli smo tudi, da aplikacija ne obremenjuje preveč računalniških virov, je uporabnikom prijazna in da jo lahko uporablja čim več uporabnikov. Zato smo aplikacijo implementirali s pomočjo programskega jezika C++ in programskih knjižnic Qt [1]. Programski jezik C++ omogoča implementacijo programa, ki računalniških virov ne obremenjuje preveč. Programska knjižnica Qt omogoča izgradnjo uporabniku prijaznega grafičnega vmesnika in aplikacijo na nivoju izvorne kode, prenosljivo na različne platforme.

Zgradbo prispevka smo zasnovali tako, da v drugem poglavju podajamo opis problema in sorodna dela, ki se ukvarjajo z analizo in s primerjavo evolucijskih algoritmov. V tretjem poglavju podajamo opis izdelane prototipne aplikacije. Kratek zaključek in smernice za nadalnje delo podajamo v četrtem poglavju.

2 Opis problema in sorodna dela

Z nastajanjem novih evolucijskih algoritmov oz. njihovih izboljšav, je potrebno le-te analizirati in med seboj primerjati. V ta namen se izvajajo tekmovanja med algoritmi, ki rešujejo vnaprej pripravljene probleme [6, 3, 8, 2, 7]. Problemi so podani v obliki izvorne kode in v obliki deljenih knjižnic. Tako pripravljene probleme nato raziskovalci integrirajo v svoje programe in z reševanjem problemov preizkušajo svoje evolucijske algoritme. Tak način preizkušanja in primerjave algoritmov ima dve slabosti. Prva je, da raziskovalci porabijo nekaj časa za integracijo problemov in algoritmov. Druga slabost je, da se pri merjenju rezultatov in njihovi analizi lahko uporabijo različne oz. nekoliko spremenjene metode. Primer neprimerljivih meritev je, če en raziskovalec uporabi pri svoji analizi večje število ovrednotenj posameznikov kot drugi ali če se uporabita različni metodi za analizo rezultatov. Na osnovi tako doseženih rezultatov je težko oz. nemogoče primerjati algoritme. Da bi se temu izognili, smo načrtovali aplikacijo, ki bo omogočila hitrejšo integracijo problemov in algoritmov ter poenotila analizo. Raziskovalci bodo tako morali le implementirati svoje algoritme, lahko tudi v različnih programskih jezikih. Popolno analizo implementiranega algoritma bodo naredili s pomočjo naše aplikacije. S pomočjo aplikacije bodo izvedli več neodvisnih zagonov, dobljene rezultate bodo shranili in dobili primerjavo algoritma glede na algoritme, ki so že bili uporabljeni za reševanje določenih problemov.

3 Evolucijska arena

Program za poenoteno primerjanje evolucijskih algoritmov smo poimenovali Evolucijska arena. Da bi zadostili zahtevam, ki smo jih opisali v prejšnjem poglavju, smo definirali protokol za komunikacijo med programom, ki vsebuje implementiran evolucijski algoritem in Evolucijsko areno. Za probleme smo uporabili že definirane deljene knjižnice, ki smo jim dodali dodatno funkcionalnost. V nadaljevanju tega poglavja bomo posebej predstavili vsakega od sestavnih delov, ki so potrebni za preizkušanje evolucijskih algoritmov.

3.1 Deljene knjižnice

Za definiranje problemov, ki jih želimo reševati, smo uporabili deljene knjižnice. Le-te lahko uporabimo tako, da se povezovanje izvrši implicitno ali eksplicitno. Implicitno povezovanje se izvrši v času prevajanja, eksplicitno v času izvajanja. V našem primeru smo uporabili eksplicitno povezovanje. To nam omogoča, da v času izvajanja aplikacije naložimo določene probleme in na osnovi njih analiziramo določen algoritem. Nato lahko probleme zamenjamo in analizo ponovimo na drugih problemih.

Zgradbo deljenih knižnic smo zasnovali glede na že definirane deljene knjižnice, ki so bile uporabljene na tekmovanjih [6, 3, 8, 2, 7]. Vsaka knjižnica vsebuje več problemov v obliki programskih funkcij. Prototipi funkcij so definirani glede na tip problemov. Trenutno imamo definirana dva tipa funkcij: funkcije brez omejitev (f1) in funkcije z omejitvami (f2):

double f1(**const int** dim, **const double** x[])
void f2(**const double** x[], **double** f[], **double** g[],
double h[], **const int** nx)

Funkcije brez omejitev imajo definirana argumenta dim in x. Dimenzijo problema določa dim, posameznika pa x. Rezultat funkcije predstavlja oceno (ang. *Fitness*) posameznika. Funkcije z omejitvami imajo definiranih pet argumentov. Posameznika predstavlja polje x, argument f pa predstavlja polje, s pomočjo katerega funkcija vrne rezultat. Enakostne in neenakostne omejitve predstavljata g in h in nx določa dimenzijo problema. Opisane probleme smo dopolnili še z dodatnimi informacijami. Dodali smo ime knjižnice, tip problemov, število funkcij in seznam funkcij. Seznam funkcij vsebuje ime funkcije, dimenzijo in optimum. Te informacije so dostopne v deljeni knjižnici s pomočjo naslednje funkcije:

```
struct problem{
    char name[255];
    double D;
    double optimum;
};

int info(char name[255], char type[255], problem p[255])
```

Vsaki funkciji smo dodali še t. i. funkcijo $info$, ki poda dodatne informacije o določenem problemu. Prototipa funkcij imata naslednjo obliko:

void f1_Info(**const int** dim, **double** min[], **double** max[])
void f2_Info(**const int** nx, **double** min[], **double** max[],
int &g, **int** &h, **double** &e)

V funkcijah predstavljata dim in nx dimenzijo problema, polji min in max predstavljata spodnje in zgornje meje intervalov iskalnega prostora parametrov. Argumenta g in h predstavljata število enakostnih in število neenakostnih omejitev. Argument e predstavlja konstanto, s pomočjo katere enakostne omejitve preoblikujemo v neenakostne omejitve. Tako razširjena definicija problemov vsebuje več informacij in posledično omogoča hitrejšo integracijo algoritmov in problemov ter analizo algoritmov. Raziskovalci več ne rabijo skrbeti za te informacije.

3.2 Evolucijski algoritem

Evolucijski algoritem oz. program, ki bo reševal probleme smo zasnovali tako, da je neodvisen od programskega jezika. V ta namen smo definirali protokol za komunikacijo med našo aplikacijo in evolucijskim algoritmom. Ta protokol omogoča, da naša aplikacija krmili evolucijski algoritem. Delovanje protokola bomo razložili na prototipnem programu, ki je vseboval implementiran algoritem diferencialne evolucije [4, 5]. Evolucijski program se zažene kot zunanji proces, ki ga krmilimo s pomočjo pip (standardni vhodno/izhodni podatkovni tokovi).

S pomočjo ukaza **info** evolucijski program prikaže osnovne informacije o programu in informacije o parametrih programa. Ukaz se zaključi z vrstico, ki vsebuje rezervirano besedo **infoend**, kot prikazuje naslednji primer:

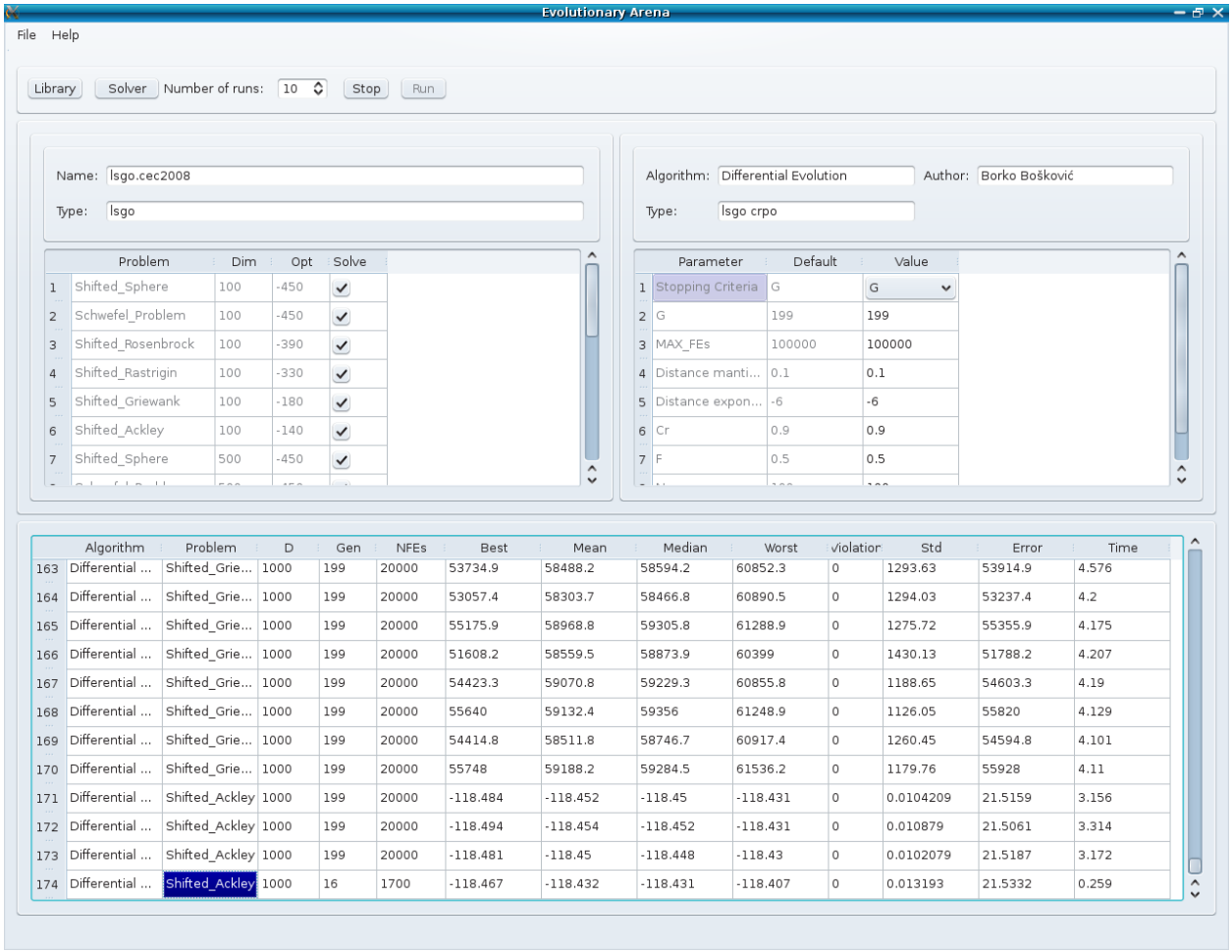
```
info
author Borko Bošković
algorithm Differential Evolution
type lsgo crpo
parameter name Library type string default ../crpo/l
ibcrpo.so value ../crpo/libcrpo.so
parameter name Problem type string default C01 value
C01
parameter name D type int default 10 min 2 max 21474
83647 value 10
parameter name Stopping Criteria type option default
G values G, Distance, MAX FEs value G
parameter name G type int default 199 min 1 max 2147
483647 value 199
parameter name MAX_FEs type int default 100000 min 1
max 2147483647 value 100000
parameter name Distance mantissa type double default
0.1 min 0 max 10 decimal 5 value 0.1
parameter name Distance exponent type int default -6
min -2147483648 max 2147483647 value -6
parameter name Optimum type double default -450 min
-1.79769e+308 max 1.79769e+308 decimal 10 value -450
parameter name Cr type double default 0.9 min 0 max
1 decimal 3 value 0.9
parameter name F type double default 0.5 min 0 max 2
decimal 3 value 0.5
parameter name Np type int default 100 min 4 max 214
7483647 value 100
infoend
```

Vidimo, da s pomočjo ukaza **info** dobimo informafcije o avtorju programa, imenu evolucijskega algoritma, vrstah problemov, ki jih algoritem zna reševati, in seznam parametrov. Parametri so definirani s pomočjo imena, tipa, vrednosti, privzete vrednosti, minimalnih in maksimalnih vrednosti. Za vsak tip parametrov smo definirali atribute. Npr., parameter *Cr*, ki predstavlja krmilni parameter križanja, je tipa *double*. Ima definirano privzeto vrednost, spodnjo in zgornjo mejo ter natančnost s pomočjo števila decimalnih mest.

Drugi ukaz, ki smo ga definirali, je **setparameter** in omogoča spreminjanje vrednosti parametrom. Spodnji primer prikazuje, kako smo spremenili vrednost parametru *Cr* na 0.5.

```
setparameter name Cr value 0.5
info parameter Cr set to 0.5
```

Tako s pomočjo tega ukaza nastavljamo krmilne parametre algoritma in parametra problemov. Parametra problemov sta ime knjižnice in ime problema oz. funkcije, ki jo



Slika 1: Evolucijska arena

želimo reševati. Za zagon algoritma smo definirali ukaz **run**, kot prikazuje naslednji primer:

```
run
...
result gen 195 nfes 19600 best -401.68 violation 2 wor
st -66.3484 mean -219.048 median -211.21 std 74.3748 e
rror 48.3196 time 0.084
result gen 196 nfes 19700 best -401.68 violation 2 wor
st -66.3484 mean -219.375 median -223.292 std 74.3614
error 48.3196 time 0.084
result gen 197 nfes 19800 best -401.68 violation 2 wor
st -66.3484 mean -220.751 median -225.512 std 75.1892
error 48.3196 time 0.085
result gen 198 nfes 19900 best -401.68 violation 2 wor
st -66.3484 mean -221.763 median -227.553 std 74.9941
error 48.3196 time 0.085
result gen 199 nfes 20000 best -401.68 violation 2 wor
st -66.3484 mean -223.717 median -227.553 std 74.626
error 48.3196 time 0.085
runend gen 199 nfes 20000 best -401.68 violation 2 wor
st -66.3484 mean -223.717 median -227.553 std 74.626
error 48.3196 time 0.086
```

Ko algoritem zaženemo, le ta vrača vmesne rezultate (**result**). Vsak vmesni rezultat vsebuje trenutno generacijo (**gen**), število ovrednotenj posameznikov (**nfes**), oceno najboljšega posameznika (**best**), število kršitev najboljšega posameznika (**violation**), oceno najslabšega posameznika (**worst**), povprečno oceno (**mean**), mediano ocen (**median**), standardno deviacijo ocen (**std**), napako oz. oddaljenost najboljšega posameznika od optimuma (**error**) in čas, ki je bil potreben, da je algoritem dosegel določeno generacijo. Ukaz **run** se konča z ukazom **runend**, ki vsebuje enake informacije kot vmesni rezultati.

3.3 Grafični vmesnik

Grafični vmesnik oz. aplikacija Evolucijska arena združuje definirane probleme in evolucijske algoritme v obliki programov, ki imajo implementiran prej definiran protokol. Aplikacija vsebuje uporabniško prijazen grafični vmesnik, ki je prenosljiv na različne platforme (Qt) na nivoju kode.

Grafični vmesnik je zasnovan tako, da vsebuje štiri področja. Prvo področje je ukazna vrstica, ki omogoča, da naložimo določene probleme (gumb Library) in evolucijski algoritem (gumb Solver), definiramo število potrebnih zagonov (vnosno polje Number of runs) in zaženemo reševanje problemov (gumb Run). Naslednje področje podaja informacije o problemih. Vsebuje ime in tip problemov ter seznam funkcij. Seznam funkcij vsebuje ime, dimenzijo in optimum problema ter izbiro. Izbira uporabniku omogoči, da izbere samo določene funkcije, ki jih želi reševati. Tretje področje je namenjeno evolucijskemu algoritmu. To področje vsebuje ime algoritma, imena tipov funkcij, ki jih algoritem zna reševati, avtorja programa in seznam krmilnih parametrov evolucijskega algoritma. V seznamu parametrov so podana imena, privzete vrednosti in trenutno nastavljene vredno-

sti parametrov. Zadnje polje vsebuje določene grafične gradnike, ki so načrtovani tako, da onemogočajo uporabniku, da bi izbral oz. nastavil neveljavne vrednosti parametrom. Zadnje področje vsebuje informacije o zagonih. Ko v ukazni vrstici s pomočjo gumba Run sprožimo reševanje problemov, se v tem področju prikazujejo vmesni in končni rezultati zagonov. Prikazani rezultati poleg informacij vmesnih rezultatov (glej poglavje Evolucijski algoritem) vsebuje še ime uporabljenega algoritma in imena problemov, ki jih algoritem rešuje.

4 Zaključek

V prispevku smo prikazali program Evolucijska arena, ki omogoča poenoteno primerjavo evolucijskih algoritmov. V ta namen smo že definirane probleme v obliki deljenih knjižnic razširili in definirali protokol za komunikacijo med evolucijskim algoritmom in grafičnim vmesnikom. Grafični vmesnik je načrtovan tako, da omogoča uporabniku prijazno zaganjanje različnih evolucijskih algoritmov in reševanje različnih problemov. Načrtovan program smo tudi implementirali v obliki prototipne aplikacije, ki je vsebovala implementiran algoritem diferencialne evolucije in probleme z in brez omejitev.

V nadaljevanju bomo poskušali predstavljeno delo še nadgraditi tako, da bomo v deljene knjižnice dodali še informaciji o pogoju za končanje evolucijskega procesa (ang. *Stopping Criteria*) in o številu neodvisnih zagonov. To nam bo omogočilo, da v grafični vmesnik dodamo še analizo rezultatov s pomočjo določenih statističnih metod.

Literatura

[1] Programska knjižnica Qt. dostopna na: http://qt.nokia.com/products/.

[2] C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H. g. Beyer in P. N. Suganthan. Benchmark Generator for CEC'2009 Competition on Dynamic Optimization. Technical report, 2008.

[3] J. J. Liang, T. P. Runarsson, E. Mezura-Montes, M. Clerc, P. N. Suganthan, C. A. Coello Coello in K. Deb. Problem Definitions and Evaluation Criteria for the CEC 2006, Special Session on Constrained Real-Parameter Optimization. Technical report, Nanyang Technological University, Singapore, 2006.

[4] R. Storn in K. Price. Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, Berkeley, CA, 1995.

[5] R. Storn in K. Price. Differential Evolution - A Simple and Efficient Heuristic for Global Optimisation Over Continuous Spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[6] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger in S. Tiwari. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Technical Report #2005005, Nanyang Technological University, Singapore and IIT Kanpur, India, 2005.

[7] K. Tang, Xiaodong Li, P. N. Suganthan, Z. Yang in T. Weise. Benchmark Functions for the CEC'2010 Special Session and Competition on Large Scale Global Optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2009.

[8] K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. P. Chen, C. M. Chen in Z. Yang. Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2007.