|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| СПКс-11 | РГР | Evolving Objects |  |  |
| Кравчук В. А. | |
| № залікової: | |
| Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні | | Викладач: | |
| Кривий Р.З. | |

**Мета роботи**

Ознайомитись з можливостями та характеристиками Evolving Objects.

**Теоретичні відомості**

### Evolving Objects (EO) є шаблонною, ANSI-C ++ бібліотекою обчислення еволюційних алгоритмів, яка допомагає писати власні стохастичні алгоритми оптимізації дуже швидко.

Він містить класи для майже будь-якого виду еволюційних обчислень які можна придумати - по крайній мері за словами авторів. Він базується на компонентах, тож якщо ви не можете знайти клас який вам потрібно, дуже легко створити підклас на основі існуючих абстрактних або конкретних класів.

Розробка алгоритму з ЕО полягає у виборі того які компоненти ви хочете використовувати для ваших конкретних потреб, так само, як побудова структури з лего.

Якщо у вас є класична проблема, для якої код існує (наприклад, якщо у вас є проблеми чорного ящика з реальними змінними), ви просто вибираєте компоненти, щоб сформувати алгоритм і підключити його до функції придатності (який обчислює якість даного рішення).

Якщо ваша проблема є трохи більш екзотичною, вам доведеться створювати власний клас, що відображає, як представлені рішення вашої проблеми, і можливо, деякі варіації операторів, але більшість інших операторів (вибір, заміна, критерії зупинки , інтерфейс командного рядка і т.д.) вже доступні в ЕО.



Рис. 1. Логотип фреймворку

**Загальний опис**

ЕО це фреймворк. Він орієнтований на полегшення створення еволюційних алгоритмів. Він не є (на даний момент) повноцінною бібліотекою алгоритмів, готових для використання з стандартними проблемами.

Якщо у вас є добре відома проблема і ви хочете її вирішити якомога швидше, спробуйте інше програмне забезпечення. Якщо у вас є реальна проблема, і ви хочете побудувати найкращий еволюційний алгоритм, щоб вирішити цю проблему, ви зробили правильний вибір.

В основному, EO маніпулює "особинами" з "пристосуванням", тобто об'єктами, що кодують рішення даної задачі оптимізації, пов'язаними з якістю цього рішення. Присиосування визначається в класі ЕО, але представлення рішення не може бути універсальним. Таким чином, EO масово використовувує шаблони, так що ви не будете обмежені інтерфейсами при використанні свого власного представлення.

Після того, як у вас є уявлення, ви будете будувати свій власний еволюційний алгоритм шляхом складання еволюційних операторів в алгоритмах. В EO, більшість об'єктів є функтори, тобто класи з оператором (), які можна назвати так само, як якби вони були класичні функції. Наприклад, алгоритм є функтором, що маніпулює населенням окремих осіб, він буде реалізований як функтор, з такис членом як: operator()(eoPop <EOT>). Після того, як називається на цій популяції, він буде шукати оптимальний даного завдання.

Як правило, оператори інстанційовані один раз, і потім передаються в алгоритм як посилання. Таким чином, ви можете легко створити свій власний алгоритм, використавши декілька комбінацій операторів.

**Налаштування фреймворку**

Для того щоб почати користування бібліотекою, потрібно завантажити архів з фреймворком <https://sourceforge.net/projects/eodev/> та розпакувати папку “eo” (рис. 2).

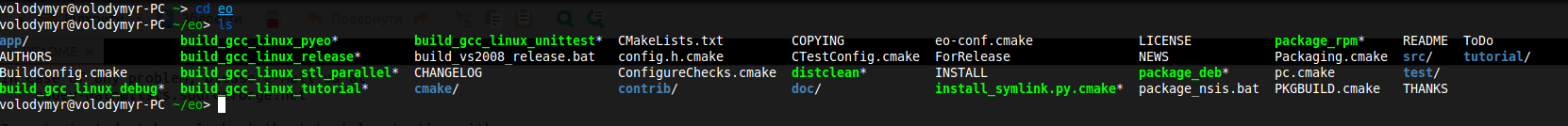


Рис. 2. Структура розархівованого каталогу

Найпростіший спосіб cкомпілювати потрібні бібліотеки це запустити один з сценаріїв що починаються з "build\_". Кожен скрипт дозволяє створювати різні частини каркаса, з різними варіантами. Я запустив скрипт “build\_gcc\_linux\_release” який створює будує основні компоненти необхідні для роботи з фреймворком.



Рис. 3. Запуск команди компіляції

Після виконання даної команди була створена папка “release” в якій містяться потрібні для роботи бібліотеки і бінарні файли.

**Приклади**

Найкраще для ілюстрації роботи фреймворку підійде реалізація генетичного алгоритму вдома методами, еволюціонуючими бітовими рядками і еволюціонуючим вектором реальних чисел.

Генети́чний алгори́тм (англ. genetic algorithm) — це еволюційний алгоритм пошуку, що використовується для вирішення задач оптимізації і моделювання шляхом послідовного підбору, комбінування і варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію.

**Приклад програми 1**

Алгоритм побудований на основі еволюціонуючих бітових рядків максимізує кількість одиниць в цих рядках.

**FirstBitGA.cpp**

// стандартні бібліотеки

#include <stdexcept>

#include <iostream>

#include <strstream>

// бібліотеки ОЕ

#include <eo>

#include <ga.h>

//-----------------------------------------------------------------------------

// визначення особин

typedef eoBit<double> Indi;

//-----------------------------------------------------------------------------

// просто функція що підраховує кількість одиниць

// @param \_indi A biststring individual

double binary\_value(const Indi & \_indi)

{

double sum = 0;

for (unsigned i = 0; i < \_indi.size(); i++)

sum += \_indi[i];

return sum;

}

//-----------------------------------------------------------------------------

void main\_function(int argc, char \*\*argv)

{

// захардкоджені параметри

const unsigned int SEED = 42;

const unsigned int T\_SIZE = 3;

const unsigned int VEC\_SIZE = 8;

const unsigned int POP\_SIZE = 20;

const unsigned int MAX\_GEN = 100;

const float CROSS\_RATE = 0.8;

const double P\_MUT\_PER\_BIT = 0.01;

const float MUT\_RATE = 1.0;

//////////////////////////

// Випадкове “насіння”

//////////////////////////

rng.reseed(SEED);

/////////////////////////////

// Фітнес функція

////////////////////////////

eoEvalFuncPtr<Indi> eval( binary\_value );

////////////////////////////////

// Ініціалізація популяції

////////////////////////////////

// опис популяції

eoPop<Indi> pop;

// заповнення

for (unsigned int igeno=0; igeno<POP\_SIZE; igeno++)

{

Indi v; // оголошення особини яку будемо змінювати

for (unsigned ivar=0; ivar<VEC\_SIZE; ivar++)

{

bool r = rng.flip(); // нове значення випадкове на {0,1}

v.push\_back(r); // присвоєння випадкового значення особині

}

eval(v); // перевірка значення

pop.push\_back(v); // додання особини до популяції

}

// сортування популяції

pop.sort();

// вивід популяції

cout << "Initial Population" << endl;

cout << pop;

/////////////////////////////////////

// селекція і заміна

////////////////////////////////////

// Надійний турнірний відбір

eoDetTournamentSelect<Indi> select(T\_SIZE);

//////////////////////////////////////

// Оператори варіації

//////////////////////////////////////

// 1-очковий перехід для бітового рядка

eo1PtBitXover<Indi> xover;

// стандартна зміна бітів для бітового рядка

eoBitMutation<Indi> mutation(P\_MUT\_PER\_BIT);

//////////////////////////////////////

// Умова завершення

/////////////////////////////////////

// зупинити після MAX\_GEN поколінь

eoGenContinue<Indi> continuator(MAX\_GEN);

/////////////////////////////////////////

// алгоритм

////////////////////////////////////////

eoSGA<Indi> gga(select, xover, CROSS\_RATE, mutation, MUT\_RATE,

eval, continuator);

// застосування алгоритму для популяції

gga(pop);

// Вивід популяції

pop.sort();

cout << "FINAL Population\n" << pop << endl;

}

// Обробка помилок і запуск

int main(int argc, char \*\*argv)

{

#ifdef \_MSC\_VER

int flag = \_CrtSetDbgFlag(\_CRTDBG\_LEAK\_CHECK\_DF);

flag |= \_CRTDBG\_LEAK\_CHECK\_DF;

\_CrtSetDbgFlag(flag);

#endif

try

{

main\_function(argc, argv);

}

catch(exception& e)

{

cout << "Exception: " << e.what() << '\n';

}

return 1;

}

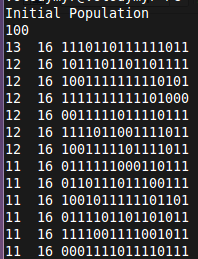


Рис. 4. Частина вхідних даних для прикладу роботи

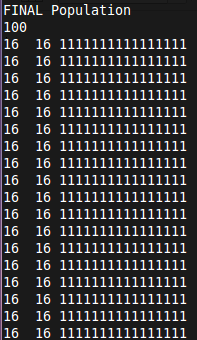


Рис. 5. Частина вихідних даних для прикладу роботи

**Приклад програми 2**

Алгоритм побудований на основі еволюціонуючого вектором реальних чисел мінімізує суму квадрата змінних.

**RealGA.cpp**

// стандартні бібліотеки

#include <stdexcept>

#include <iostream>

#include <strstream>

// бібліотеки ОЕ

#include <eo>

#include <es.h>

//-----------------------------------------------------------------------------

// визначення особин

typedef eoReal<double> Indi;

//-----------------------------------------------------------------------------

// Проста функція придатності, яка обчислює евклидову норму реального вектора

// @param \_indi A real-valued individual

double real\_value(const Indi & \_indi)

{

double sum = 0;

for (unsigned i = 0; i < \_indi.size(); i++)

sum += \_indi[i]\*\_indi[i];

return (-sum);

}

//-----------------------------------------------------------------------------

void main\_function(int argc, char \*\*argv)

{

// захардкоджені параметри

const unsigned int SEED = 42;

const unsigned int VEC\_SIZE = 8;

const unsigned int POP\_SIZE = 20;

const unsigned int T\_SIZE = 3;

const unsigned int MAX\_GEN = 500;

const float CROSS\_RATE = 0.8;

const double EPSILON = 0.01;

const float MUT\_RATE = 0.5;

//////////////////////////

// Випадкове “насіння”

//////////////////////////

rng.reseed(SEED);

/////////////////////////////

// Фітнес функція

////////////////////////////

eoEvalFuncPtr<Indi> eval( real\_value );

////////////////////////////////

// Ініціалізація популяції

////////////////////////////////

// опис популяції

eoPop<Indi> pop;

// заповнення

for (unsigned int igeno=0; igeno<POP\_SIZE; igeno++)

{

Indi v; // оголошення особини яку будемо змінювати

for (unsigned ivar=0; ivar<VEC\_SIZE; ivar++)

{

double r = 2\*rng.uniform() - 1; // нове значення випадкове на {0,1}

v.push\_back(r); // присвоєння випадкового значення особині

}

eval(v); // перевірка значення

pop.push\_back(v); // додання особини до популяції

}

// сортування популяції

pop.sort();

// вивід популяції

cout << "Initial Population" << endl;

cout << pop;

/////////////////////////////////////

// селекція і заміна

////////////////////////////////////

// The robust tournament selection

eoDetTournamentSelect<Indi> select(T\_SIZE); // T\_SIZE in [2,POP\_SIZE]

//////////////////////////////////////

// Умова завершення

/////////////////////////////////////

// зупинити після MAX\_GEN поколінь

eoGenContinue<Indi> continuator(MAX\_GEN);

//////////////////////////////////////

// Оператори варіації

//////////////////////////////////////

// Потомство (i) рівномірно вибирається в [parent(i)-epsilon, parent(i)+epsilon]

eoUniformMutation<Indi> mutation(EPSILON);

// Потомство (i) являє собою лінійну комбінацію батька(i)

eoSegmentCrossover<Indi> xover;

/////////////////////////////////////////

// алгоритм

////////////////////////////////////////

eoSGA<Indi> gga(select, xover, CROSS\_RATE, mutation, MUT\_RATE,

eval, continuator);

// застосування алгоритму для популяції

gga(pop);

// Вивід популяції

pop.sort();

cout << "FINAL Population\n" << pop << endl;

}

// Обробка помилок і запуск

int main(int argc, char \*\*argv)

{

#ifdef \_MSC\_VER

// rng.reseed(42);

int flag = \_CrtSetDbgFlag(\_CRTDBG\_LEAK\_CHECK\_DF);

flag |= \_CRTDBG\_LEAK\_CHECK\_DF;

\_CrtSetDbgFlag(flag);

// \_CrtSetBreakAlloc(100);

#endif

try

{

main\_function(argc, argv);

}

catch(exception& e)

{

cout << "Exception: " << e.what() << '\n';

}

return 1;

}

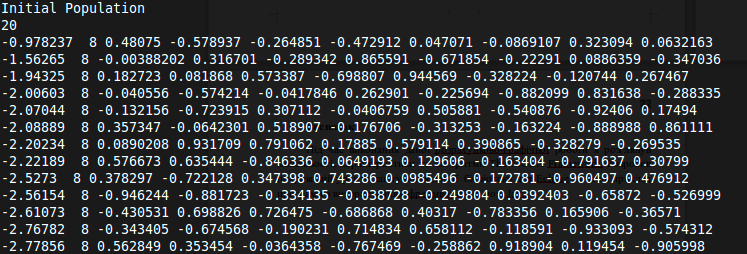


Рис. 6. Частина вхідних даних для прикладу роботи

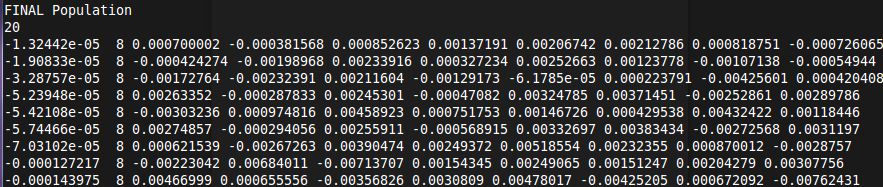


Рис. 7. Частина вихідних даних для прикладу роботи

**Висновок**

Під час виконання даної розрахунково-графічної роботи я розглянув можливості, характеристики та алгоритми Evolving Objects. Розробив покроковий урок налаштування даної бібліотеки для OS Linux та навів приклад програми з використанням фреймворку на мові C++.