МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

*Кафедра САПР*



**Розрахункова робота**

з курсу: «Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні»

на тему: «Еволюційна Обчислення бібліотека Evolving Objects»

Виконав:

ст. гр. КНС-13

Писаренко В. Б.

Прийняв:

асистент каф. САП

Кривий Р.З.

**Львів 2017**

Зміст

[Вступ 3](#_Toc410947531)

[1.1 Еволюційні алгоритми 3](#_Toc410947531)

[1.2. Бібліотеки EA 4](#_Toc410947531)

[2.1 Структури даних 6](#_Toc410947531)

[2.2 Функція значимості 6](#_Toc410947531)

[2.3 Оператори варіації 8](#_Toc410947531)

[3. Технічний опис 9](#_Toc410947531)

[4. Додатки 10](#_Toc410947531)

[Висновки 11](#_Toc410947531)

[Список літератури 13](#_Toc410947531)

**Вступ**

Evolving Objects - об'єктно-орієнтоване середовище для еволюційних обчислень (EС), метою якого є забезпечити гнучкий набір класів для створення додатків ЄС. Мета Olib полягає в тому, щоб мати можливість розвивати будь-якої об'єкт, EO концентрується на інтерфейсах; будь-який об'єкт може розвиватися, якщо він наділений інтерфейсом. У цій статті ми описується функції, які повинен мати об'єкт для розвитку, і деякі приклади, як ЕО було впроваджено в практику розвитку еволюційних нейронних мереж, і інших нових додатків.

Еволюційні алгоритми (EAs) є стохастичними алгоритмами оптимізації, які базуються на грубому наслідуванні природної дарвінівської еволюції. Нещодавно вони стали популярнішими в різних областях досліджень, і розробники стикаються з важкою дилемою: або вони використовують існуючу бібліотеці EA, а потім повинні дотримуватися її обмежень або писати свої власні, що представляють собою величезну роботу.

Наприклад, при розробці будь-яких об'єктів (наприклад, нейронних мереж) складним питанням було те, в основному через відсутність гнучкості нинішніх еволюційних обчислювальних бібліотек та варіанти  
Операторів, які можуть бути використані для цього подання. Більшість бібліотек допускають тільки кілька визначених уявлень.  
Розвиток інших типів структур даних, отже, часто доводиться починати з вирівнювання. Їх до одного зі звичайних уявлень, таких як бітовий рядок, з плаваючою комою.

Для декодування для оцінки мережі (наприклад, на навчальному наборі у випадку проблеми регресії, так як іноді вона не має точності (наприклад, в разі бітового строкового представлення) або виразу рядок, незалежно від її форми відбувається серіалізація складної структури даних та еволюція рядки з використанням стандартних операторів варіацій на основі рядків, що дозволяють зберігати фактичні будівельні блоки разом важче, ніж еволюція структури, більш тісно представляє нервові сітки, такі як два масиви ваг і зсуви для тришарових персептронів, або, більш загальніше, масив об'єктів, що становлять нейрони.  
Аналогічним чином, більшість існуючих бібліотек пропонують тільки обмежений набір способів застосування дарвінівських оператори до популяцій (наприклад, обмежених деяким пропорційним відборомта зміною поколінь), або взагалі один метод застосування.

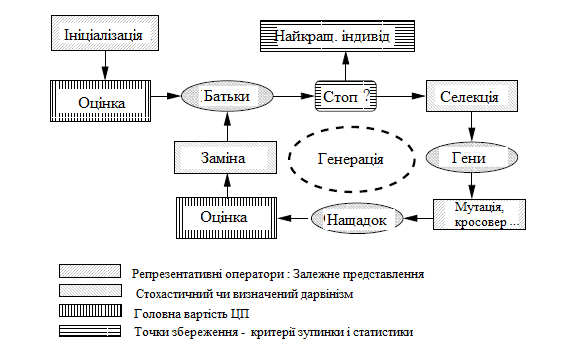
Різні види операторів варіації для членів цих груп популяцій (наприклад, обмежені послідовно застосовують до всіх членів популяції один кросовер  
І один оператор мутації, кожен із заданою вірогідністю). Однак існують  
безліч інших шляхів та сильну взаємодію між усіма параметрами  
еволюційного алгоритму, що унеможливлює апріорі вирішити, який шлях  
кращий.

У цій статті представлений EOlib, вільна від парадигми бібліотека еволюційних обчислень, що дозволяє легко розвивати будь-які структури даних (об'єкти), які виконують невеликий набір умов. Більш того, алгоритми, запрограмовані в EOlib, не обмежуються основними існуючими EC парадигмами, як генетичні алгоритми, стратегіями еволюції, еволюційним або генетичне програмуванням, будь то на рівні популяції еволюції або варіації. Хоча все перераховане вище існує в EO, оригінальні експерименти можуть бути легко виконані з використанням EOlib building блоків.

**1.1 Еволюційні алгоритми**

У цьому розділі ми коротко зупинимося на основних етапах EA, підкресливши взаємозалежності різних компонентів. Проблема полягає в оптимізації заданої цільової функції у даному пошуковому простору. Чисельність популяції піддається штучній дарвінській еволюції, в якій пристосованість індивідуума безпосередньо пов'язана зі значеннями  
цільової функції в заданій точці.

Після (в основному випадкової) ініціалізаціі популяцій, генерація петель алгоритму описана на рисунку 1:



*Рис 1 : Ескіз еволюційного алгоритму*

**Критерій зупинки** (і збору статистики): найпростіший критерій зупинки  
заснований на лічильнику генерації t (або на числі функцій оцінки). Можливо використовувати більш складні критерії зупинки, які залежать або від еволюції придатності в популяції поколінь (т. ч. Вимірювання градієнта коефіцієнтів посилення по кількостями поколінь) або за деякими показниками різноманітності популяцій.

**Селекція**: вибір деяких особин, які будуть генерувати потомство.   
Можуть бути використані процеси вибору, детерміновані або стохастичні. Усе залежить від придатності осіб, безпосередньо пов'язаних з функцією. Залежно від обраної схеми відбору деякі особини можуть бути обиратися більше одного разу. У цей момент окремі особини породжують копії самих себе, своїх творців.

**Застосування варіаційних операторів**: до кожної з цих копій застосовуються деякі оператори , породжуючи одного або декількох нащадків. Ці оператори як правило, є стохастичними операторами, і зазвичай розрізняють кроссовер (або рекомбінації) і оператори мутації:

Оператори перетину - це оператори типу Ek (в більшості випадків, k = 2) в E,  
Тобто деякі батьки обмінюються генетичним матеріалом для створення одного потомства.Оператори мутації (в загальному випадку) є стохастичними операторами типу E в E.

**Оцінка:** обчислення придатності всіх новостворених нащадків. Як уже згадувалося критерій придатності індивіда безпосередньо пов'язаний з його  
значенням цільової функції. В будь-якому реальному додатку 99%  
загальної вартістості ЦП для ЕО виходить з оціночної частини.

**Заміна**: вибір, які особини стануть частиною наступного покоління.  
Вибір може бути зроблений або з набору нащадків (в цьому випадку  
всі батьки «помруть») або від обох наборів нащадків і батьків. В будь-якому випадку, ця процедура заміни може бути детермінованою або стохастичною.

Описані вище компоненти можна розділити на дві підмножини, які  
відносяться до принципів природної еволюції Дарвіна: виживання найбільш пристосованих та невеликих неорієнтованих варіацій.  
- Вибір і заміна, також званий двигуном еволюції, описують спосіб дарвінських еволюцій застосовується до популяцій.Тільки пов'язані з пристосованістю індивідуумів, і повністю незалежні уявлення (простору пошуку).  
- Оператори ініціалізації і варіації є специфічними для подання, але мають (в більшості випадків) не пов'язане з відповіднстю, дотримуючись ідеї, що варіації повинні бути ненаправленими.

Ця базова класифікація вже дає деякі підказки про те, як проектувати еволюційної бібліотеки, яка стане однією з основ проектування ЕО.

**1.2. Бібліотеки EA**

Існує безліч безкоштовних бібліотек EA;

Більшість бібліотек занадто складні у використанні, дуже Бібліотека еволюційних обчислень, що включає кілька парадигм, подань, та безліч операторів варіації. Однак їй не вистачає гнучкості у ряді областей.  
По-перше, вибір існуючих уявлень також обмежений масивами (біт, цілих чисел або з плаваючою комою, або будь-якою комбінацією), хоча його можна розширити шляхом підклассіфікаціі. Наприклад, розвиток нейронної мережі, означає стискання його в масив. По-друге, він допускає тільки два варіаційних оператора для кожного геному: мутацію і кроссовер (крім оператора ініціалізації). Більш того, ці оператори завжди застосовуються послідовно, і єдина ступінь свободи в цьому відношенні є можливостями застосування. Аналогічним чином, немає простого способу реалізації стратегії розвитку самоадаптивної мутації або глобальною рекомбінації [2].

І останнє, але не менш важливе: реалізована тільки скалярна придатність, що ускладнює методи обробки обмежень в загальному вигляді і майже неможливим для багатокритеріальної оптимізації.

Бібліотека, представлена ​​в цій статті, EOlib, володіє гнучкістю для об'єктно-орієнтованого проектування: кожна структура даних, кожен оператор, кожна статистична обчислювальна процедура є об'єктом.

**2.1 Структури даних**

Будь-яка структура даних може бути розвинена, якщо надається хоча б один оператор зміни. Для таких структур будуть використовуватися в якості обчислювальних аналогів для трьох критеріїв еволюції, описані Мейнардом-Смітом [26], а саме спадковість, мінливості і плодючості, і будуть розглянуті по черзі:  
**- Ініціалізація:** це властивість, хоча і істотною для експерта, насправді не має  
природної протидію в будь-який з біологічних моделей еволюції. В цілому  
приділяється мало уваги в існуючих бібліотеках, оскільки існують стандартні процедури для стандартних уявлень. Однак навіть такі стандартні процедури можуть бути сумнівним в деяких ситуаціях [21]. У будь-якому випадку, в EO, ініціалізатор об'єктів, що дозволяє використовувати більше однієї ініціалізації.  
Процедура, загальна функція в GP, наприклад [3].  
**- Можливість вибору:** одна з основних складових дарвінівської еволюції – природного вибору, іноді також розглядається як виживання найбільш пристосованих. У EO, як в всіх бібліотеках EA, всі об'єкти прикріплені до відповідності, використовується для такого вибору.

- реплікаційна: можливо отримати (можливо недосконалі) копії об'єкта, за допомогою інших об'єктів (реплікаторів). Має близьку аналогію з критерієм спадковості. Він також повинен створювати об'єкти з нуля, використовуючи об'єктні фабрики.

-**Мутабельність:** це перша можлива реалізація критерію Мейнарда-Сміта варіабельності, яка говорить про те, що процес копіювання генотипу має недосконалості.  
Таким чином, потомство не дорівнює батьківському, мутація збільшує  
різноманітність населення. Мутаційні оператори, або мутатор, можуть змінювати розвиток об'єкта одним або декількома способами. Клієнту може бути гарантовано, що об'єкт буде змінюватися деяким (зазвичай стохастичним) способом.  
**- Комбінованість:** Інший можливий варіаційний оператор об'єднує два або більше об'єкти для створення нового (аналогічно кросоверу GA). Це не завжди можливо, але коли це так, операція зазвичай зменшує різноманітність.  
Об'єкти зазвичай називаються комбінаторами або матрицями. Один із способів забезпечити мінімальну значимість матеріалів - дотримуватись деяких правил форми рекомбінації [31]. Оскільки на практиці комбінаторам практично неможливо слідувати за всіма з них [11], кожен комбайнер повинен слідувати хоча би один.

У структурах даних, що мають ці характеристики, включаючи еволюційні алгоритми які були розвинені Фогелем і співробітниками [15][1] і Грефенстетт [17].

**2.2 Функція значимості**

Значимість експертів - це єдиний спосіб вказати, що представляє собою природне середовище в природної еволюції. У більшості бібліотек EA, на жаль, значимість обмежена до одного скалярного значення, і природний відбір, закінчується на основі порівняння цих скалярних значень. Однак такий вибір є дуже обмежувальним, і не передбачає інших механізмів відбору, таких як відбір на основі обмежень, заснованих на кількох завданнях або більш складних процесах спільної еволюції за участю однієї популяції часткових рішень [9] або декількох конкуруючих популяцій [30].

У EOlib значимість може бути будь-якого типу (більш технічно: все Evolutionary Objects розділяються над типом придатності. Звичайно, скалярна (реальна) придатність все ще найбільш широко використовується,та доступні найпопулярніші варіанти вибору і заміни для повноцінної придатності.

Але також можливо використовувати значимості, які є векторами дійсних чисел, і для розробки багатоцільового [12] або універсального селектора обробки обмежень [29]. І наприклад, в NSGAII [13] існують адаптивні сегрегаційні обмеження [4] вже реалізовані в EO.

**2.3 Оператори варіації**

Варіаційні оператори в EO - це об'єкти, які існують поза генотипів, на які вони діють. Будь-яке число операторів варіації може бути спроектовано для однієї і тієї ж еволюційної структури даних. Крім того, оператори варіації можуть приймати будь-яку кількість вхідних даних і генерувати будь-яку кількість виходів, що дозволяє, наприклад, легко реалізувати операторів глобальної рекомбінації ES [2].

Будучи окремими об'єктами, оператори варіації можуть володіти деякими особистими даними: наприклад, можна вибрати спеціальний селектор для вибору помічника першого партнера. Всі ці приватні параметри можуть бути змінені під час виконання, що дозволяє легка реалізація. Стандартне відхилення гаусових мутацій в стратегіях розвитку, або слідуючи добре відомим правилом 1/5 [32], або використовуючи механізми самоадаптації [2].

Оператори варіації можуть комбінуватися по-різному. Існують дві базові конструкції : послідовна комбінація, в якій застосовуються оператори варіації  
та для всього населення з певними показниками (як в «простій генетиці». Пропорційна комбінація, яка вибирає тільки одну на основі відносних заздалегідь визначених ваг. Більш того, ці два способи комбінування операторів варіації можуть бути рекурсивно вбудованими. Наприклад, дуже популярна комбінація операторів - змішувати різні кросовери і різні мутації в рамках Simple GA, що являє собою послідовну рекомбінацію пропорційного вибору серед наявних кросоверів і пропорційного вибору серед наявних мутацій. Ці конструкції, будучи самими об'єктами, можуть бути створені під час виконання, наприклад. Модифікуючи різні ставки, засновані на минулій еволюції.

**3. Технічний опис**

Всі ідеї EO були реалізовані на практиці у бібліотеці класів EOlib, Open  
Вихідна бібліотека C ++, доступна за адресою http://eodev.sourceforge.net, разом  
З усіма можливостями відкритого проекту в SourceForge. ,, Поточна версія 0.9.1,. EOlib потребує сумісний з ANSI-C ++ компілятор, такий як Free Software Foundation gcc (В Linux, інших Unix-версіях або середовищі CygWin для Win95 / 98 / NT); Більшість класів також працюють з комерційними компіляторами, такими як Microsoft Visual C ++ 6.0.

Крім «еволюційних класів», згаданих в попередньому розділі, загальні  
Засоби для додатків EC, такі як контрольні точки для зупинки і перезапуску  
Додатків, багаторазовий збір статистики, графічне представлення в Інтернеті, яке використовує інтерактивні графіки в Linux. EOlib відкритий: використовує існуючі файли шаблонів підручників, реалізації власної нової статистики і її відображення в режимі он-лайн.

Існує два способи використання EOlib (будь то бітові рядки, речовинний вектор або парні дерева). Єдина річ яка повинна бути запрограмована, полягає в тому, що функція обчислення придатності і всі інші компоненти алгоритму можуть бути просто введені в якості параметрів програми.  
З іншого боку, використання спеціального уявлення вимагає кодування іншого від уявлення частини алгоритму: визначення уявлення, операторів ініціалізації і варіації (див. Розділ 2.1). ,, функції значимості, звичайно. Аналогічним чином, тестування нового вибору відбувається вставляючи його в існуючу програму EO, все інше без змін.

Ще один план - надати репозиторій об'єктів, щоб, запрограмовані з використанням EOlib, класи об'єктів можуть бути негайно опубліковані для використання. Одним з основних результатів було б поліпшення відтворюваності результатів EC: тоді як документ написаний з використанням EOlib, вихідний код всіх експериментів доступний, і подальші дослідження можуть реально використовувати його як відправний пункт. Посилання може бути додане в репозиторій документів, такий як один з європейських мереж еволюційних обчислень, EvoNet чи інших.

**4. Додатки**

До сих пір ЕО застосовувалося в ряді різних областей. Велика гнучкість  
бібліотеки використана для реалізації складних уявлень (наприклад, багаторівневих персептронів, діаграми Вороного ), разом з їх конкретними операторами варіації, багатокритерійної оптимізації, спеціальних механізми обробки обмежень, гібридних операторів мутації.

В результаті еволюції були створені багатошарові персептрони  
самих себе. Необхідно було розробити нові оператори для урізноманітнення: додати або усунути прихований шар нейрона, прихований кросовер шару, і мутувати початкові ваги.

Алгоритм навчання зворотного поширення також використовувався в якості оператора мутації [7]. Ця програма є також http://geneura.ugr.es/~pedro/GProp.  
- Генетичний натхненник. У разі гри Mastermind [27], GA. Був запрограмований на пошук прихованої комбінації, яка поліпшує результати в попередніх реалізаціях. Оператори варіацій також були адаптовані до цих об'єктів: оператор перестановки і повзучості. Алгоритм досліджував тільки 25% простору, було досліджений до [5], тобто близько 2% від загального обсягу пошукового простору, і таким чином, отримане рішення набагато швидше. Гру можна грати онлайн за адресою http://geneura.ugr.es/~jmerelo/GenMM; Код можна завантажити з на тому ж сайті. Еволюція нечітких логічних контролерів [33]: Двовимірні нечіткі логічні контролери були розвинені для апроксимації функцій з двома змінними; Додані оператори варіації і відняті рядки і стовпці, змінені значення прецедентів і наступні. Розвинений об'єкт апроксимувати функцію, і, крім того, знайшов правильну кількість рядків і стовпців для контролера.  
- Еволюція нейронних мереж RBF [34]: структури даних, що представляють RBF з різноманітними радіусами в кожному вимірі; Оператори варіацій додають і віднімають RBF, а також змінюють положення центрів і значення  
радіусів. Розвинені RBF, як правило, менші і точніше, ніж інші знайдені  
за допомогою додаткових процедур. Еволюційна сегментація голосу [28]: проблема полягає в знаходженні розподілу мовного потоку, різні слова, фрази або фонеми можуть бути розділені. EO розвиває маркери сегментації, причому дуже гарні результати. В цьому випадку розвинена структура даних урахована до лінійної сегментації. - Паралельна версія EOlib з використанням MPI і PVM знаходиться в розробці;

Версія була протестована для декількох еталонних проблем [6].  
- EOlib застосовувався до сегментації зображень в [39,40], в якому застосовується генетичний алгоритм випрямлення смуг, які використовуються для обробки, а потім стиснення зображення ембріонів плодових мух.

З використанням комбінації багатоцільового та обмеженого обслуговування методів (див. [37 - запроваджено і протестований гібридний сурогатний оператор мутації для параметричної оптимізації. Перші результати, також і є дуже перспективними [38].

Топологічне оптимальне проектування конструкцій -[24]. Однак нещодавно він був перенесений в середу EO [18]  
Так як це в основному багатоцільова задача (мінімізує вага і максимізує зміщення при заданому навантаженні). ВЕО, вдалося реально порівняти обидва підходи, оскільки вони використовують точно такі ж оператори уявлення і варіації.

Адаптивне логічне програмування [22] Використовувалася хромосома змінної довжини для управляння контуром через логічну програму, щоб генерувати (обмежені) математичні вирази.

**Висновки**

У цій роботі був предаствлений еволюційний обчислення під назвою EO, яке буде включати всі еволюційні обчислення, парадигми еволюції нових структур даних, загальних або окремих операторів варіації, і будь-яких операторів зміни рівня населення. EO має практичну реалізацію у вигляді бібліотеки класів EO, яка є публічною і вільно доступною в рамках LGPL (Бібліотека FSF-менш загальна загальнодоступна ліцензія) з http://eodev.sourceforge.net. В якості можливих напрямків майбутньої роботи буде спроба реалізувати EOlib на різних мовах, таких як Java. Ще однією особливістю є генератор додатків, який буде використовувати високорівневу еволюційну мову обчислень, таку як EASEA [10], і незалежний від операційної системи графічний інтерфейс користувача.

**Код прикладу на мові С++ для задачі комівояжера :**

#include "Beagle/EC.hpp"

#include "Beagle/IntVec.hpp"

#include "TSPEvalOp.hpp"

#include <cstdlib>

#include <iostream>

#include <stdexcept>

#include <vector>

#include <numeric>

using namespace std;

using namespace Beagle;

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

{

try {

// Build the system

System::Handle lSystem = new System;

// Install the GA integer vector package

const unsigned int lNumberOfTowns = 25;

lSystem->addPackage(new IntVec::Package(lNumberOfTowns-1));

// Add distance map component

lSystem->addComponent(new TSPDistanceMap(lNumberOfTowns));

// Add evaluation operator allocator

lSystem->setEvaluationOp("TSPEvalOp", new TSPEvalOp::Alloc);

// Set initialization, crossover and mutation operators to use

lSystem->getFactory().setConcept("InitializationOp", "GA-InitIndicesIntVecOp");

lSystem->getFactory().setConcept("CrossoverOp", "GA-CrossoverIndicesIntVecOp");

lSystem->getFactory().setConcept("MutationOp", "GA-MutationShuffleIntVecOp");

// Set the use of a minimization fitness

lSystem->getFactory().setConcept("Fitness", "FitnessSimpleMin");

// Initialize the evolver

Evolver::Handle lEvolver = new Evolver;

lEvolver->initialize(lSystem, argc, argv);

// Create population

Vivarium::Handle lVivarium = new Vivarium;

// Launch evolution

lEvolver->evolve(lVivarium, lSystem);

} catch(Exception& inException) {

inException.terminate(cerr);

} catch(std::exception& inException) {

cerr << "Standard exception caught:" << endl << flush;

cerr << inException.what() << endl << flush;

return 1;

}

return 0;

}

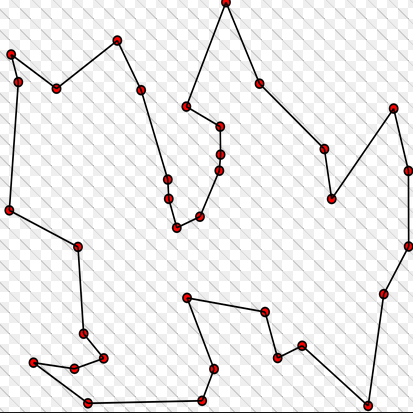


Рис 2 : Приклад виконання програми

**Код прикладу на мові С++ для максимізації функції :**

#include "Beagle/EC.hpp"

#include "MaxFctEvalOp.hpp"

#include <cmath>

using namespace Beagle;

/\*!

\* \brief Construct the individual evaluation operator for maximising the function.

\*/

MaxFctEvalOp::MaxFctEvalOp() :

EvaluationOp("MaxFctEvalOp")

{

for(unsigned int i=0; i<5; i++) {

BitStr::BitString::DecodingKey lKey(-200.0, 200.0, 25);

mDecodingKeys.push\_back(lKey);

}

}

/\*!

\* \brief Evaluate the fitness of the given individual.

\* \param inIndividual Current individual to evaluate.

\* \param ioContext Evolutionary context.

\* \return Handle to the fitness value of the individual.

\*/

Fitness::Handle MaxFctEvalOp::evaluate(Individual& inIndividual, Context& ioContext)

{

Beagle\_AssertM(inIndividual.size() == 1);

BitStr::BitString::Handle lBitString = castHandleT<BitStr::BitString>(inIndividual[0]);

Beagle::DoubleArray lX;

lBitString->decodeGray(mDecodingKeys, lX);

double lU = 10.0;

double lSum = 0.0;

for(unsigned int i=0; i<5; i++) {

lSum += ((lX[i])\*(lX[i])) + (lU\*lU);

lU += lX[i];

}

lSum += (lU\*lU);

double lF = 161.8 / lSum;

return new EC::FitnessSimple(lF);

}

**Код прикладу на мові С++ для мінімізаціцї функції :**

#include "Beagle/EC.hpp"

#include "Beagle/BitStr.hpp"

#include "ZeroMinEvalOp.hpp"

#include <cstdlib>

#include <iostream>

#include <stdexcept>

#include <vector>

#include <numeric>

using namespace std;

using namespace Beagle;

/\*!

\* \brief Main routine for the ZeroMin minimization problem.

\* \param argc Number of arguments on the command-line.

\* \param argv Arguments on the command-line.

\* \return Return value of the program.

\* \ingroup ZeroMin

\*/

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

{

try {

// Build the system.

System::Handle lSystem = new System;

// Install the GA bit string package

const unsigned int lNumberOfBits = 50;

lSystem->addPackage(new BitStr::Package(lNumberOfBits));

// Add evaluation operator allocator

lSystem->setEvaluationOp("ZeroMinEvalOp", new ZeroMinEvalOp::Alloc);

// Set the use of FitnessSimpleMin

lSystem->getFactory().setConcept("Fitness", "FitnessSimpleMin");

// Initialize the evolver

Evolver::Handle lEvolver = new Evolver;

lEvolver->initialize(lSystem, argc, argv);

// Create population

Vivarium::Handle lVivarium = new Vivarium;

// Launch evolution

lEvolver->evolve(lVivarium, lSystem);

} catch(Exception& inException) {

inException.terminate(cerr);

} catch(std::exception& inException) {

cerr << "Standard exception caught:" << endl << flush;

cerr << inException.what() << endl << flush;

return 1;

}

return 0;

}

**Список літератури**

*1. T. B¨ack. Self-adaptation in genetic algorithms. In F. J. Varela and P. Bourgine,*

*editors, Proceedings of the First European Conference on Artificial Life. Toward a*

*Practice of Autonomous Systems, pages 263–271, MIT Press, Cambridge, MA.*

*2. Th. B¨ack and H.-P. Schwefel. An overview of evolutionary algorithms for parameter*

*optimization. Evolutionary Computation, 1(1):1–23, 1993.*

*3. W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. Genetic Programming*

*— An Introduction On the Automatic Evolution of Computer Programs and Its*

*Applications. Morgan Kaufmann, 1998.*

*4. S. BenHamida and M. Schoenauer. An adaptive algorithm for constrained optimization*

*problems. In M. Schoenauer et al., editor, Proceedings of the 6th Conference*

*on Parallel Problems Solving from Nature, pages 529–539. Springer-Verlag,*

*LNCS 1917, 2000.*

*5. J. L. Bernier, C. Ilia Herr´aiz, J. J. Merelo, S. Olmeda, and A. Prieto. Solving*

*mastermind using GAs and simulated annealing: a case of dynamic constraint*

*optimization. In Parallel Problem Solving from Nature IV, pages 554–563. Springer-*

*Verlag, LNCS 1141, 1996.*

*Evolving Objects: A General Purpose Evolutionary Computation Library 241*

*6. J. G. Castellano, M. Garc´ıa-Arenas, P. A. Castillo, J. Carpio, M. Cillero, J. J.*

*Merelo, A. Prieto, V. Rivas and G. Romero. Objetos evolutivos paralelos. In XI*

*Jornadas de Paralelismo, Universidad de Granada Depto. ATC, pages 247–252,*

*2000.*

*7. P. A. Castillo, J. Gonz´alez, J. J. Merelo, A. Prieto, V. Rivas, and G. Romero.*

*G-Prop-III: Global optimization of multilayer perceptrons using an evolutionary*

*algorithm. In GECCO99, 1999.*

*8. P.A. Castillo, J.J. Merelo, V. Rivas, G. Romero, and A. Prieto. Evolving Multilayer*

*Perceptrons. Neural Processing Letters 12(2):115–127, 2000.*

*9. P. Collet, E. Lutton, F. Raynal, and M. Schoenauer. Polar ifs + individual gp =*

*efficient inverse ifs problem solving. Genetic Programming and Evolvable Machines,*

*1(4), 2000.*

*10. P. Collet, E. Lutton, M. Schoenauer, and J. Louchet. Take it easea. In M. Schoenauer*

*et al., editor, Proceedings of the 6th Conference on Parallel Problems Solving*

*from Nature, pages 891–901. Springer Verlag, LNCS 1917, 2000.*