|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| СПКс-11 | РГР | ParadisEO |  |  |
| Семків І.Б. | |
| № залікової: | |
| Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні | | Викладач: | |
| Кривий Р.З. | |

**Мета роботи**

Ознайомитись з можливостями та характеристиками ParadisEO.

**Теоретичні відомості**

ParadisEO - орієнтований на об'єкт структура, присвячений гнучкому дизайну метаеврістікі. Використовує EO, засноване на шаблоні, ANSI-C ++ гнучка бібліотека обчислення. ParadisEO портативний і через систему Windows і через послідовні платформи (Unix, Linux, Mac OS X, і т.д.). ParadisEO поширюється відповідно до ліцензії CeCill і може використовуватися під декількома навколишнім середовищем.

**Огляд**

Paradiseo є об'єктно-орієнтованою бібліотекою, присвячений багаторазовії конструкції метаеврістики, гібридних метаеврістиці і паралельні і розподілені метаеврістиці. Paradiseo надає широкий спектр функцій, включаючи еволюційні алгоритми, локальний пошук, оптимізація рою частинок, найбільш поширених паралельних і розподілених моделей і механізмів гібридизації і т.д. Цей великий вміст і корисність заохочує його використання на міжнародному рівні. Paradiseo ґрунтується на чіткості концептуального поділу методів вирішення проблем, які вони покликані вирішувати. Такий поділ дає користувачеві максимум повторного використання дизайну коду. Крім того, дрібнозернистий характер класів, передбачених рамками дозволяє більш високу гнучкість в порівнянні з іншими системами. Paradiseo складається із рідкісних структур, які забезпечують найбільш поширені паралельні та розподілені моделі. Їх реалізація є портативним на машинах з розподіленою пам'яттю, а також на багатопроцесорних системах із загальною пам'яттю, так як він використовує стандартні бібліотеки, такі як MPI, PVM і PThreads. Моделі можуть бути використані прозорим способом, потрібно просто інстанлювати пов'язані з ними надані класи. Їх експерименти на радіомережі реальних додатках продемонструвати свою ефективність.

paradisEO_small.png

Рис. 1. Логотип фреймворку

**Модулі**

**Paradiseo-EO**

Paradiseo-EO займається метаеврістікj. на основі населення, це шаблони на основі ANSI-C ++, які відповідають еволюційним бібліотекам обчислення (еволюційні алгоритми, оптимізація рою частинок ...). Він містить класи для майже будь-якого виду еволюційних обчислень. Це компонент на основі, так що якщо ви не можете знайти клас вам потрібно в ньому, дуже легко підкласи існуючі абстрактні або конкретні класи.

**Paradiseo-MOEO**

Paradiseo-MOEO надає широкий спектр інструментів для проектування багатокритеріальної оптимізації метаеврістікі: схеми призначення придатності (досягнення функції, рейтинг, заснований на показниках ...), збереження різноманітності механізмів (спільне використання, витісняючи), елітарність, показники ефективності (внесок, ентропія ...), статистичні інструменти і кілька простих у використанні державою в найсучасніших мульти-об'єктивні еволюційні алгоритми (NSGA, NSGA-II, IBEA ...).

**Paradiseo-MO**

Paradiseo-MO займається метаеврістікі на основі одного рішення, вона надає інструменти для розробки єдиного рішення на основі метаеврістікі: Холм альпіністських, пошук із заборонами, Ітеративна локального пошуку (ILS), імітації відпалу, додатковий оцінки, частковий околиці ...

**Paradiseo-PEO**

Paradiseo-PЕО надає інструменти для розробки паралельних і розподілених метаеврістікі: паралельні оцінки, паралельна функція оцінки, острів модель, клітинна модель ... Paradiseo-PЕО також запроваджуються інструменти для розробки розподілених, гібридних і кооперативних моделей.

***Налаштування фреймворку***

Для того щоб почати користування бібліотекою, потрібно завантажити архів з фреймворком http://paradiseo.gforge.inria.fr/index.php?n=Download.Download

***Інсталяція***

1. Встановити G ++ і CMake.

2. Завантаження і розпакування Paradiso.

3. У терміналі, якщо ви перебуваєте в папці Paradiso:

***mkdir build && cd build***

4. Для того, щоб зробити установку простіше, Paradiseo запропонувати Вам два типи установки, які є "Full" та "Мін".

Повний відповідає приклади / уроки, тести і, очевидно, бібліотеки.

Мінімум відповідає бібліотек і заголовків, і це стандартна поведінка.

5. Відповідно до ваших потреб, використовуйте одну з наступних команд:

***cmake .. -G "Unix Makefiles" -DINSTALL\_TYPE=full***

6. І, нарешті, скомпілювати Paradiso

***make***

**Приклади**

Найкраще для ілюстрації роботи фреймворку підійде реалізація генетичного алгоритму вдома методами, мутації.

До мутацій відноситься все те ж, що і до розмноження: є деяка доля мутантів m, що є параметром генетичного алгоритму, і на кроці мутацій необхідно вибрати mN осіб, а згодом змінити їх згідно з заздалегідь заданими операціями мутації.

**Приклад програми 1**

***Поліноміальна Мутація.***

#ifndef POLYNOMIALMUTATION\_H\_

#define POLYNOMIALMUTATION\_H\_

#include <utils/eoRealVectorBounds.h>

template<class EOT> class PolynomialMutation: public eoMonOp<EOT>

{

public:

PolynomialMutation(eoRealVectorBounds & \_bounds, const double& \_p\_mut = 0.5, const double& \_eta = 1.0):

p\_mut(\_p\_mut), eta(\_eta), bounds(\_bounds) {}

/// ім'я класу.

virtual std::string className() const {

return "PolynomialMutation";

}

/\*\*

\*

\* @param \_eo параметр зазнає мутації

\*/

bool operator()(EOT& \_eo)

{

bool hasChanged=false;

double rnd, delta1, delta2, mut\_pow, deltaq, delta\_max;

double y, yl, yu, val, xy;

for (unsigned j=0; j<\_eo.size(); j++)

{

if (rng.flip(p\_mut))

{

y = \_eo[j];

yl = bounds.minimum(j);

yu = bounds.maximum(j);

delta1 = (y-yl)/(yu-yl);

delta2 = (yu-y)/(yu-yl);

// додавання

if ( (y-yl) > (yu-y))

delta\_max = delta2;

else

delta\_max= delta1;

// останнє доповнення

rnd = rng.uniform();

mut\_pow = 1.0/(eta+1.0);

if (rnd <= 0.5)

{

xy = 1.0-delta\_max;//delta\_max au lieu de delta1

val = 2.0\*rnd+(1.0-2.0\*rnd)\*(pow(xy,(eta+1.0)));

deltaq = pow(val,mut\_pow) - 1.0;

}

else

{

xy = 1.0-delta\_max;//delta\_max au lieu de delta2

val = 2.0\*(1.0-rnd)+2.0\*(rnd-0.5)\*(pow(xy,(eta+1.0)));

deltaq = 1.0 - (pow(val,mut\_pow));

}

//доповнення

if (deltaq > delta\_max)

deltaq = delta\_max;

else if (deltaq < -delta\_max)

deltaq= -delta\_max;

// останнє доповнення

y = y + deltaq\*(yu-yl);

bounds.truncate(j, y);

\_eo[j] = y;

hasChanged = true;

}

}

return hasChanged;

}

private:

double p\_mut;

double eta;

eoRealVectorBounds & bounds;

};

#endif /\*POLYNOMIALMUTATION\_H\_\*/

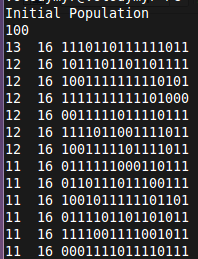


Рис. 4. Частина вхідних даних для прикладу роботи

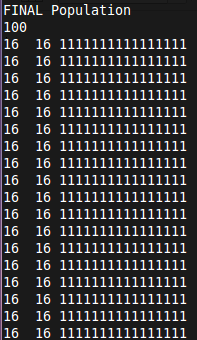


Рис. 2. Частина вихідних даних для прикладу роботи

**Приклад програми 2**

***SBX Кросовер***.

SBXCrossover.h

#ifndef SBXCROSSOVER\_H\_

#define SBXCROSSOVER\_H\_

#include <algorithm> // swap\_ranges

#include <utils/eoParser.h>

#include <utils/eoRNG.h>

#include <es/eoReal.h>

#include <utils/eoRealBounds.h>

#include <utils/eoRealVectorBounds.h>

template<class EOT> class SBXCrossover: public eoQuadOp<EOT>

{

public:

/\*\*\*\*

\* (За замовчуванням) Конструктор.

\* Межі ініціалізується з глобальним об'єктом, який дає повідомлення: немає кордонів. \*

\*

\*/

SBXCrossover(const double& \_eta = 1.0) :

bounds(eoDummyVectorNoBounds), eta(\_eta), range(1) {}

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

/\*\*

\* Конструктор з межами

\* @param \_bounds an eoRealVectorBounds that контейнер, який містить межі

\* @param \_alphaMin the кількість розвідки поза батьків

\* як в BLX альфа-нотації (Ешелман і Шаффер)

\* 0 == стискає додаток

\* Повинно бути позитивним

\*/

SBXCrossover(eoRealVectorBounds & \_bounds,

const double& \_eta = 1.0) :

bounds(\_bounds), eta(\_eta), range(1) {}

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

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

/\*\*

\* Конструктор з синтаксичного аналізатора. Прочитає від аргументу парсер

      \* EoRealVectorBounds, який містить межі

      \* Ета, параметр SBX \*/

SBXCrossover(eoParser & \_parser) :

// По-перше, вирішити, чи слід обмежені об'єктивними змінними

// Увага, повинні бути одні й ті ж ключові слова, ніж інші можливі Bounds об'єкта в іншому місці

bounds (\_parser.getORcreateParam(eoDummyVectorNoBounds, "objectBounds", "Bounds for variables", 'B', "Variation Operators").value()) ,

// потім отримати мета значення

eta (\_parser.getORcreateParam(1.0, "eta", "SBX eta parameter", '\0', "Variation Operators").value()) ,

range(1) {}

# define EPS 1.0e-14

/// назва класц.

virtual std::string className() const {

return "SBXCrossover";

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* SBX кросовер - змінює обоє батьків\*

\* @param \_eo1 Перший батьківський \*

\* @param \_eo2 Перший батьківський \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool operator()(EOT& \_eo1, EOT& \_eo2)

{

unsigned i;

double rand;

double y1, y2, yl, yu;

double c1, c2;

double alpha, beta, betaq;

bool changed = false;

for (i=0; i<\_eo1.size(); i++)

{

if (rng.flip())

{

if (fabs(\_eo1[i] - \_eo2[i]) > EPS) // для уникнення поділу на 0 {

// y2 doit ГЄtre > Г  y1

if (\_eo1[i] < \_eo2[i])

{

y1 = \_eo1[i];

y2 = \_eo2[i];

}

else

{

y1 = \_eo2[i];

y2 = \_eo1[i];

}

yl = bounds.minimum(i);

yu = bounds.maximum(i);

rand = rng.uniform();

beta = 1.0 + (2.0 \* (y1 - yl) / (y2 - y1));

alpha = 2.0 - pow( beta, -(eta + 1.0));

if (rand <= (1.0/alpha))

{

betaq = pow ( (rand \* alpha), (1.0 / (eta + 1.0)));

}

else

{

betaq = pow ( (1.0 / (2.0 - rand \* alpha)), (1.0 / (eta+1.0)));

}

c1 = 0.5 \* ((y1 + y2) - betaq \* (y2 - y1));

beta = 1.0 + (2.0 \* (yu - y2) / (y2 - y1));

alpha = 2.0 - pow( beta, -(eta + 1.0));

if (rand <= (1.0/alpha))

{

betaq = pow ( (rand \* alpha), (1.0 / (eta + 1.0)));

}

else

{

betaq = pow ( (1.0 / (2.0 - rand \* alpha)), (1.0 / (eta + 1.0)));

}

c2 = 0.5 \* ((y1 + y2) + betaq \* (y2 - y1));

bounds.truncate(i, c1);

bounds.truncate(i, c2);

if (rng.flip())

{

\_eo1[i] = c2;

\_eo2[i] = c1;

}

else

{

\_eo1[i] = c1;

\_eo2[i] = c2;

}

changed = true;

}

}

}

return changed;

}

protected:

eoRealVectorBounds & bounds;

double eta;

double range; // == 1

};

#endif /\*SBXCROSSOVER\_H\_\*/

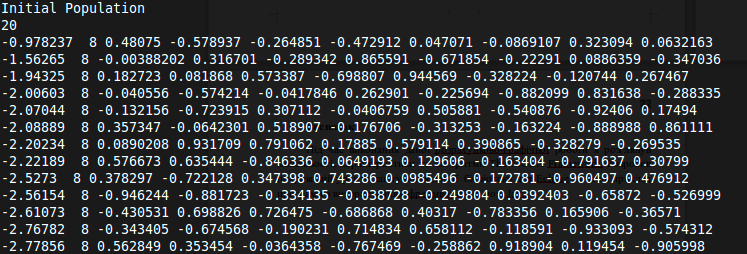


Рис. 3. Частина вхідних даних для прикладу роботи

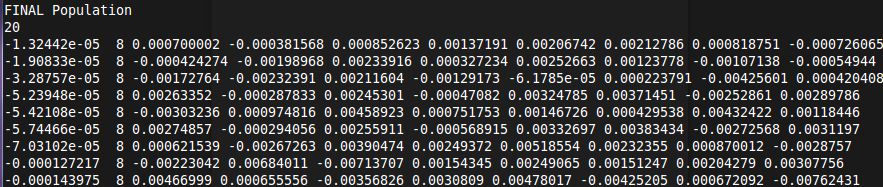


Рис. 4. Частина вихідних даних для прикладу роботи

Висновок

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