*дата :*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| КНC-13 | Розрахункова робота | Фреймворк **ParadisEO** |  |  |
| Назар Богдан | |
| Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні | | Викладач: | |
| Кривий Р. З. | |

1. **Опис фреймворку.**

ParadisEO є об'єктно-орієнтованою структурою білого ящика, присвячена гнучкій конструкції метаевристики. Він використовує ЕО, на основі шаблону, ANSI-C ++ сумісної бібліотеки обчислень. ParadisEO використовується на системі Windows, також на платформи (Unix, Linux, Mac OS X і т.д.). ParadisEO поширюється по ліцензії CeCILL і може бути використаний в рамках декількох середовищ.

EO є шаблон, на основі ANSI-C ++ еволюційної бібліотеки обчислення, яка допомагає писати ваші власні алгоритми стохастичної оптимізації дуже швидко.

Він містить класи для майже будь-якого виду еволюційних обчислень які можна придумати - по крайній мірі, для тих, які ми могли б бути. Він компонентний, тому якщо ви не можете знайти клас, який потрібно в ньому, дуже легко найти підкласи , існуючі абстрактні або конкретні класи.

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

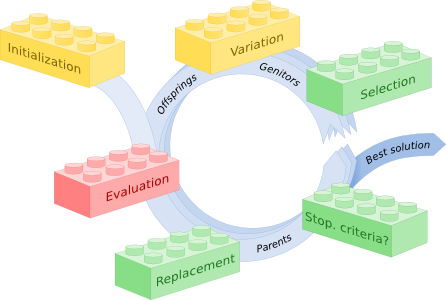


Рис.1.Схема розробки алгоритму

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

Якщо ваша проблема трохи більше екзотична, вам доведеться кодувати клас, який представляє, як представлені ваші особи (рішення вашої проблеми), і, можливо, деякі варіації оператори, але більшість інших операторів (вибір, заміна, зупиняючи критерії , інтерфейс командного рядка і т.д.) вже доступний в ЕО.

1. **Загальний дизайн**

ЕО є основою. Вона орієнтована на сприяння дизайну ADHOC еволюційних алгоритмів. Це не є (на даний момент) повна бібліотека алгоритмів, готових для використання на канонічних проблем.

Якщо у вас є добре відома проблема і хочете її вирішити якомога швидше, спробуйте інше програмне забезпечення. Якщо у вас є реальна проблема, і ви хочете, щоб побудувати кращий еволюційний алгоритм рішення, **ParadisEO є хорошим вибором для цього.**

Bascially, EO маніпулювати «особистості» з «пристосованістю», тобто об'єкти, що кодують вирішення даної задачі оптимізації, пов'язані з якістю цього рішення. Фітнес визначається в класі ЕО, але уявлення рішення не може бути універсальним. Таким чином, EO масово використовувати шаблони, так що ви не будете обмежені інтерфейсами при використанні свого власного уявлення.

Якщо у вас є уявлення, ви будете будувати свій власний еволюційний алгоритм по збірці еволюційних операторів в алгоритмах. В ЕО, більшість об'єктів є функтори, тобто класи з оператором (), який можна назвати так само, як якщо б вони були класичні функції. Наприклад, алгоритм являє собою функтор, що маніпулювати населення окремих осіб, вона буде реалізована як функтор, з членом, як: оператор () (eoPop <EOT>). Після того, як називається на дану популяції, він буде шукати оптимум даної проблеми.

Як правило, оператори інстанціірован один раз і потім переплетені в алгоритмі за допомогою посилання. Таким чином, ви можете легко створити свій власний алгоритм, намагаючись кілька поєднання операторів

1. **Можливості та функції фреймворку.**

**Гнучка конструкція, яка дозволяє легко створити практично будь-який алгоритм**

* Представлення рішення для безперервних і комбінаторних задач:
* Бінарні-струни
* перестановки
* вектори
* легко написати свій власний

**Кілька алгоритмів парадигм:**

* еволюційні стратегії
* генетичні алгоритми
* оцінка розподілу
* оптимізація рою частинок

**Багато вибір і замін операторів:**

* Оцінка на основі
* детерміновані або стохастичні турнір
* рулетка
* елітарність

**Готові до використання варіації операторів:**

* рівномірна ініціалізатор
* Гаусове мутації
* поддерево кросовер

**Просте поєднання кількох операторів:**

* пропорційне поєднання
* послідовний виклик

**Розпаралелювання інструменти:**

* Спільна пам'ять петель розгортаючи (з OpenMP)
* Повідомлення проходження розпаралелювання (з OpenMPI):

карта / зменшити подібну конструкцію, з можливістю вибору операторів, як в ЕО

корисні існуючі оператори (паралельний динамічний многозаходная, статичні оцінки, ...)

* Переносні і легким для читання файли параметри
* Призупинення та населення навантаження з файлів
* Універсальний чекпойнтінг і реєстрація:

графічний дисплей,

файлу дампа

різні статистичні дані

Сигнал лову

**Вихор Мерсенна генератор випадкових чисел (і різні розподілу)**

* Немає марно обчислення (щадний фітнес-виклик, функтори на основі викликів)
* Швидка швидкість бігу, завдяки C ++
* І більше!

1. **Паралельність і гібридизація. Дизайн для Multi-Об'єктів. Проблеми використання ParadisEO**

На практиці, мульти-об'єктивні проблеми оптимізації різноманітні, вони постійно розвиваються (щодо потреб, обмежень, цілей і т.д.), вони обробляють велику кількість змінних рішення, і вони повинні мати справу з екземплярами

збільшення розміру. Незважаючи на це, загальна мета, як і раніше знайти рішень приближених до Парето- оптимальних. Як показано на рис.2., на додаток до паралельних і розподілених середовищах, рамки ParadisEO вкладається широкий діапазон особливості, в тому числі еволюційних алгоритмів (оскільки вона заснована на системі ЕО), різні місцеві пошуки (так як він заснований на структурі МО) і багатоцільові механізми (так як він заснований на фреймворку MOEO). Крім того, її загальний аспект дозволяє легко додавати деякі з його особливостей до задачі MOEO.

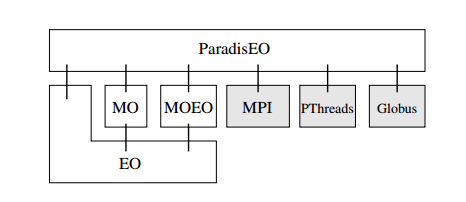


Рис. 2. Паралельність в ParadisEO

1. **Структура даних в ЕO**

Будь-яка структура даних може бути розвинена, якщо принаймні один оператор варіації передбачений для таких структур. Дещо раніше існуючі уявлення вже існують, від скромної бітової рядки, аж до і включаючи GP дерева розбору і багатошарового уявлення.

Які функції структури даних повинні бути в EO? Це повинно бути ініціалізація; вибір і відтворювання; і або змінюванні або комбінаторні. Ці властивості будуть використовуватися в якості обчислювальних аналогів для трьох критеріїв еволюції, викладених Мейнарда-Сміта , а саме спадковість, мінливість і плодючості і які будуть розглянуті в свою чергу:

- Initialisability: Це властивість, в той час як важливо в EA, насправді не має природне зустрічну участі в будь-якому з біологічних моделей еволюції (звичайно, ми не розглядаємо тут creativism як модель еволюції!). Це, як правило, також приділяється мало уваги в існуючих бібліотеках, так як існують стандартні процедури для стандартних уявлень. Однак, навіть такі стандартні процедури можуть бути сумнівними в деяких ситуаціях. В ЕО ініціалізатор об'єктів wliich дозволяє використовувати більш ніж одну процедуру ініціалізації.

- Selectability: один з головних компонентів дарвінівської еволюції є природний добір, іноді також розглядається як виживання найбільш пристосованих.

- Replicability: повинно бути можливістю отримати (можливо, недосконалу) копію об'єкта, будь то самі по собі або tlirough використання інших об'єктів (реплікатори). Також повинна бути можливість створювати об'єкти з нуля, використовуючи об'єктні фабрики.

- Mutability : це перша можлива реалізація критерію Мейнарда-Сміта варіабельності, який говорить, що недосконалість генотипу процесу копіювання LIAS, це те що потомство рути не дорівнює батьку (їй). Мутація збільшує різноманітність населення. Мутаційні оператори або мутатор, не може змінити еволюцію об'єкта в одному або декількох напрямках, але внутрішні роботи мутації не повинні бути відомі зовні. Може бути гарантія тільки в тому, що об'єкт буде змінюватися (зазвичай стохастичним) способом.

- Combinability: Інший можливий оператор варіації об'єднати два або кілька об'єктів, щоб створити нову (подібним чином до кросоверу ГА). Tliis не завжди можливо, але коли операція зазвичай зменшується різноманітність, в тому сенсі, що робить об'єкти в популяції більш схожі на один одного (хоча в деяких випадках, таких як двійковий кросовер Noil-шанобливою кордонів генів або відстань зберігає Кросовер Mcrz і Frcisleben , це може привести до збільшення різноманітності). Як це відбувається з мутацією, точні внутрішні механізми рекомбінації не повинні бути відомі для клієнта. Ці об'єкти, як правило, називаються комбінатори або матер. Один із способів забезпечити мінімальну осмисленість матера повинні слідувати деякі з правил Гогта рекомбінації. Оскільки на практиці це взагалі неможливо слідувати за всіма з них, кожен комбайнер повинен слідувати по крайній мірі один. Combinabil- таким чином ОТВ може служити як в якості компонента спадковості і мінливості компонента, це в залежності від точної природи комбінації. Гарна новина полягає в тому, що більшість завдань, що вирішуються за допомогою комп'ютера можуть бути реалізовані в структурах даних, що мають ці характеристики, в тому числі самих еволюційних алгоритмів, wliich були еволюціонували вже на Fogcl і співавторами , Bacck і Grefenstette ; Дійсно, в рамках ЕО, алгоритми можуть бути просто як ще один об'єкт, і багаторівневий еволюційний алгоритм може бути, природно, встановлений в рамках ЕО.

1. **Функція Фітнес**

Фітнес в ЕА є єдиним способом, щоб визначити, що являє собою природне середовище в природній еволюції. У більшості бібліотек EA, на жаль, фітнес обмежується однією скалярною величиною, і природний відбір, отже, в кінцевому підсумку, грунтуючись на порівнянні цих скалярних значень. Проте, такий вибір має дуже обмежувальний характер, і не передбачає інші механізми відбору, такі як вибір на основі обмежень, на основі кількох проблем, або більш складних процесів коеволюції за участю одної популяції часткових рішень або кількох конкуруючих або співпрацюючих населень.

1. **Оператори варіації**

Оператори варіації в об'єктах дуги ЕО. Існує поза генотипів вони діють так: отже, будь-яка кількість операторів варіації може бути розроблена для однієї і тієї ж. Крім того, оператори варіації можуть приймати будь-яку кількість входів і генерувати будь-яку кількість виходів, що дозволяє, наприклад, використовувати прості реалізації операторів або ES операторів глобальної рекомбінації . Оператори варіації можуть володіти деякими особистими даними: наприклад, спеціальний селектор для вибору помічника першого партнера може дати оператору кросоверів, дозволяючи переваги, які необхідно прийняти до уваги, всі ці приватні параметри можуть бути змінені під час виконання, що дозволяє легко реалізувати C.G. стандартне відхилення гауссовских мутацій в еволюційних стратегіях.

Оператори варіації можуть бути об'єднані по-різному. Існують дві основні конструкції: послідовне поєднання, в wliich операторів варіації дуги застосовуються один за іншим до всього населення з конкретними ставками (як в простих генетичних алгоритмів). Пропорційна комбінація, яка вибирає тільки один оператор серед тих, яких він знає, на основі відносних про певні ваги. Крім того, ці два способи об'єднання операторів варіації можуть бути рекурсивно вбудовані. Наприклад, дуже популярна комбінація операторів змішувати різні crossoversand різних мутацій в межах простої структури GA .

1. **Еволюція**

Еволюційні двигуни можуть бути надані по-різному: Звичайно, найбільш популярні двигуни (центр ваги Generational Г.А., Steady-State GA, EP, як ES + і ES, Strategics) доступні. Але крім того, всі параметри еволюційного двигуна можуть бути вказані у великих деталях: оператор вибору і його параметри, число нащадків для генерації, частка сильної елітарності (краще індивідууми дуга копіюється на наступне покоління Регар dless потомства), процедура заміни (чи включає батько чи ні), слабка елітарність (замінити найгірший індивіда в новій популяції за кращим батькові, якщо кращий фітнес зменшується) ... Таким чином, нові еволюційні системи можуть бути визначені тільки шляхом складання існуючих EO будівельних блоків.

1. **Технічний опис**

Всі ідеї ЕО були введені в практику в бібліотеці класів EOlib, бібліотека з відкритим вихідним кодом C ++ wliich доступний з http://eodev.sourceforge.net, разом з усіма зручностями відкритого проекту в SourccForgc: кілька списків розсилки, доступ до CVS в вихідному дереві, звіти про помилки, ... поточна версія 0.9.1, провідний нуль в версії вказує на те, що вона ще не завершена. EOlib потрібен-C ANSI ++ сумісний компілятор, наприклад GCC Foundation Free Software (в Linux, інші версії Unix або навколишньому середовищу Cygwin для Win95 / 98 / NT); більшість класів також працюють з комерційними компиляторами, таких як Visual C ++ від Microsoft 6.0.

Крім «еволюційних класів», згаданих в попередньому розділі, загальні можливості для застосування ЄС, наприклад, реєстрації, що вказує для зупинки і перезапуску програми, збираючи кілька статистики, графічний онлайн-уявлення, яке використовує GNUPLOT в Linux також передбачені. Крім того, EOlib відкритий: з використанням існуючих файлів шаблонів підручник, впровадження власних нових статистичних даних і відображення його на лінії.

Є два способи використання EOlib. Найчастіший випадок, коли ваше уявлення вже визначено в EO (будь то bitstrings, речовинний вектор або parse- дерева), і ви просто хочете оптимізувати цю функцію придатності. Єдине, що повинно бути запрограмовано там це те, що функція придатності обчислення, і все інші- компоненти алгоритму можуть бути просто введені в якості параметрів програми.

З іншого боку, за допомогою спеціального уявлення вимагає кодування основного уявлення залежить від певного алгоритму: визначення уявлення, ініціалізації та операторам варіації. Крім того, тестування нового вибору, наприклад, може бути зроблено, просто підключивши його до існуючої програми ЕО, весь інший час без змін.

Ще одним плюсом є створення сховища об'єктів, так що якщо щось запрограмоване за допомогою EOlib, класи об'єктів можуть бути негайно розміщені для всіх, щоб використовувати їх.

1. **Встановлення фреймворку.**

**У терміналі, якщо ви перебуваєте в папці ParadisEO:**

> mkdir build && cd build

**Відповідно до ваших потреб, використовуйте одну з наступних команд:**

> CMake .. -G -DINSTALL\_TYPE "MinGW Makefiles" = ful

**або**

> CMake .. -G "MinGW Makefiles" -DINSTALL\_TYPE = min

**що еквівалентно**

> CMake .. -G "MinGW Makefiles"

**Нарешті, скомпілювати ParadisEO:**

> make

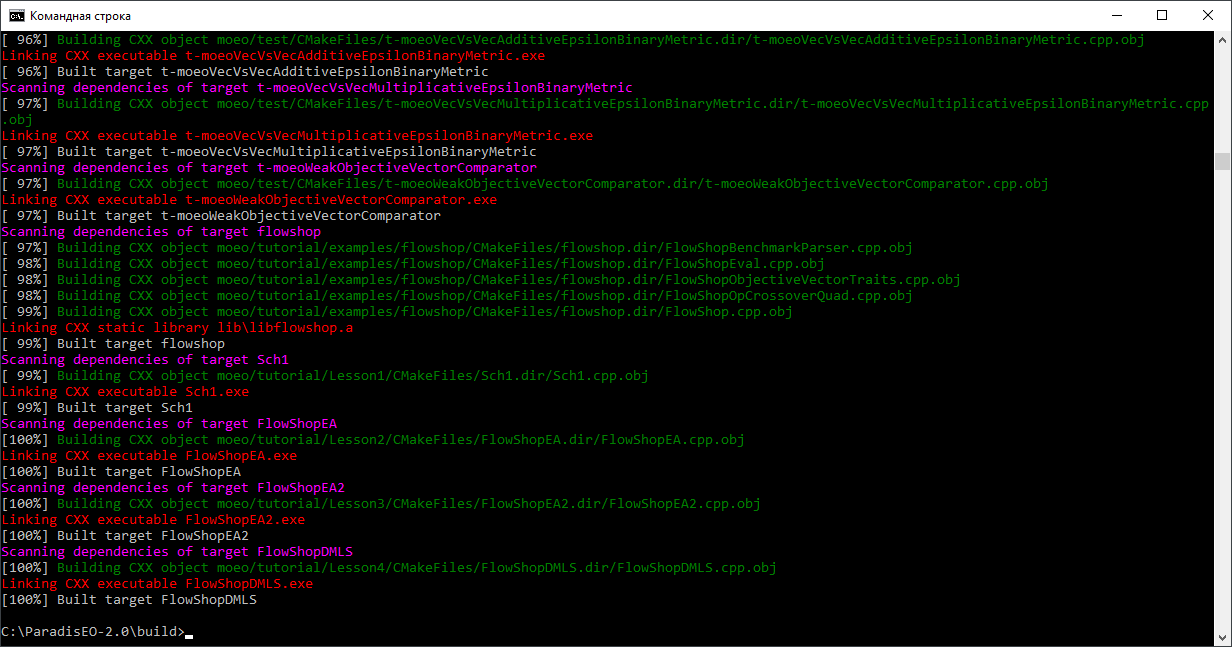


Рис. 3. Встановлення ParadisEO

1. **Приклад використання Paradis-EO**

**1.Запуск програми в visual c++ 2008,та використання ga.**

**Припустимо, що ми вже відкрили рішення, яке ви хочете скомпілювати.**

**Спочатку виберемо режим компіляції.**

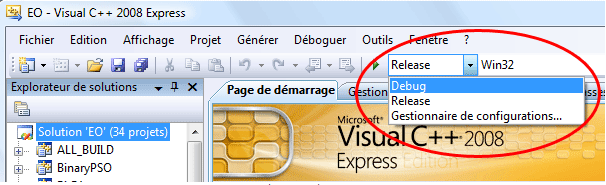


Рис. 4. Режим компіляції

**Сформувати рішення (правою кнопкою миші на Solution> лівою кнопкою миші на Генерувати рішення.**

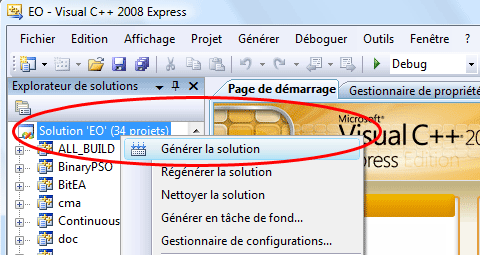


Рис.5. Формуєм рішення

**Якщо рішення ParadisEO вже складено ви можете використовувати рішення команди Regenerate.**

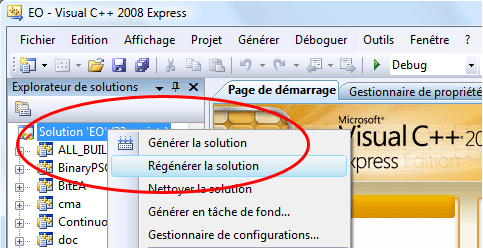


Рис. 6. Команда Regenerate

**Ви також можете очистити рішення (всі виконувані в обраному режимі компіляції будуть видалені).**

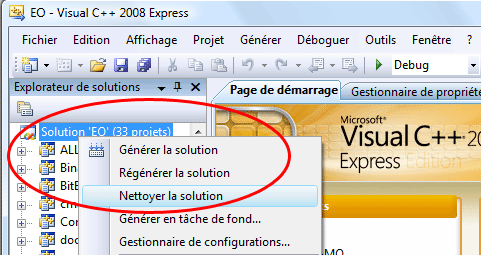


Рис. 7. Очищення рішення

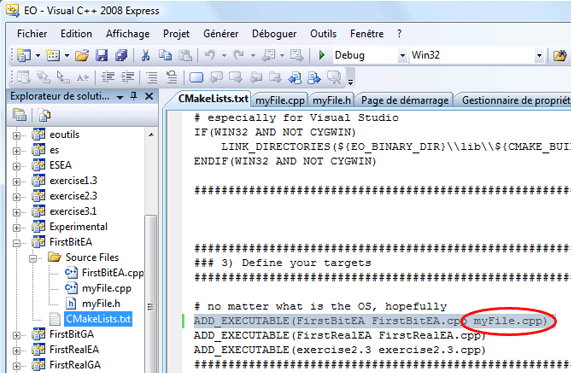


Рис. 8. Добавлення власного файла у проект.

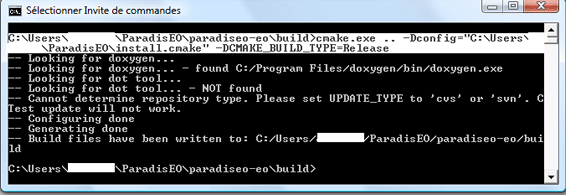


Рис. 9. Запуск cmake

Визначаємо проект, який ви хочете виконати.

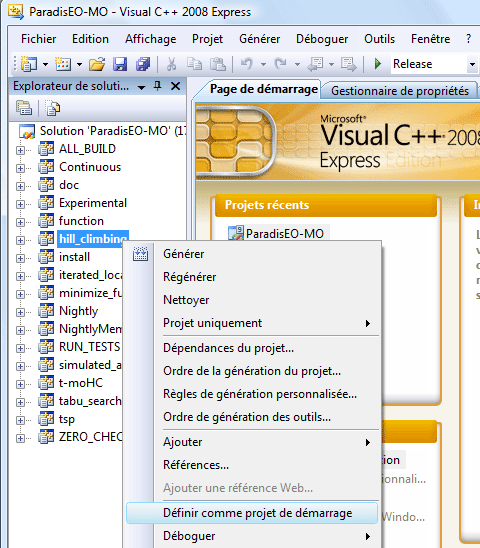


Рис. 10. Вибір проекта

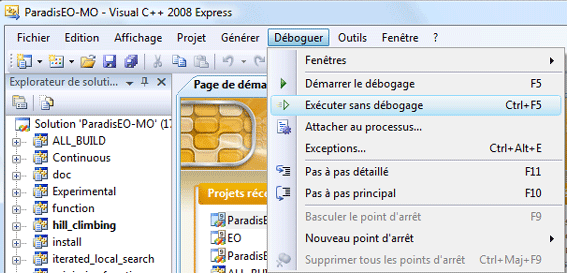


Рис. 11. Запуск проекта

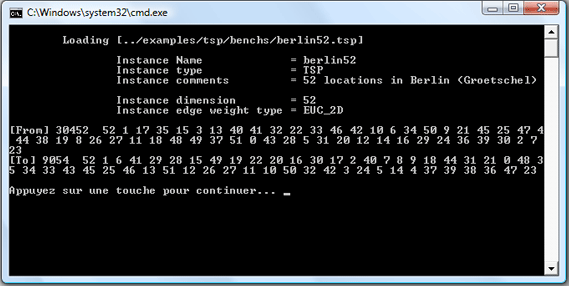


Рис. 12. Результат запуску проекта

Заходим в параметри проекта, і тут ми можем задати їх вручну.

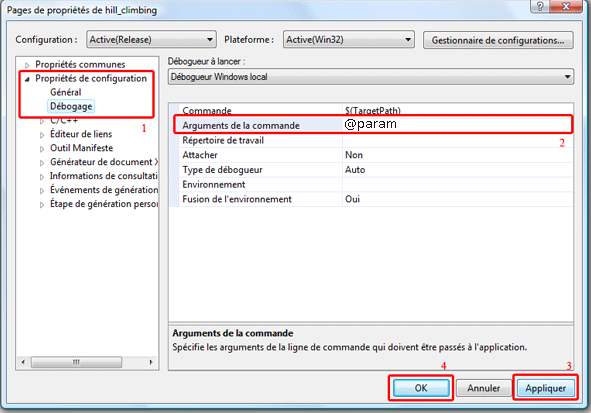


Рис. 13. Задання параметрів

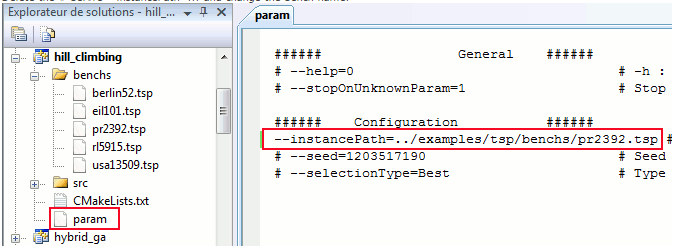


Рис. 14. Задаєм параметри через файл

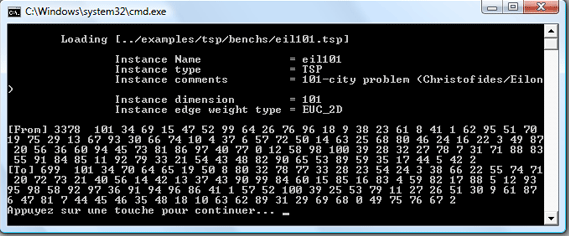


Рис. 15. Результат виконання.

**2.Визначити уявлення генотипу**

Цей клас є генотип кожної людини в генетичному алгоритмі. Це відповідає нашому уявленню в класі QAP. Тоді ми повинні викликати функцію create() нашого класу QAP для ініціалізації.

class ProblemInit : public eoInit<Problem> {

/\* Thanks to the previous line, genotype of each individual

is the same as the representation of our Problem\*/

public:

void operator()(Problem & \_problem) {

/\* It is just a call of our problem initialization \*/

\_problem.%color=blue%create()%%;

}

};

1. **Визначити функцію оцінки**

Клас функції оцінки був уже визначений, це просто виклик функції evaluate () нашого класу QAP.

1. **Визначення кросовера**

Кросовер обраний тут дуже наївно. Для того, щоб зробити дитину, ми копіюємо загальні атрибути обох батьків. Інші елементи вибираються випадковим чином.

class ProblemXover : public eoQuadOp<Problem> {

public:

/\* The two parameters in input are the parents.

The first parameter is also the output ie the child

\*/

bool operator()(Problem & \_problem1, Problem & \_problem2) {

int i;

int random, temp;

int unassigned\_positions[n];

int remaining\_items[n];

int j = 0;

/\* 1) find the items assigned in different positions for the 2 parents \*/

for (i = 0 ; i < n ; i++){

if (\_problem1.solution[i] != \_problem2.solution[i]){

unassigned\_positions[j] = i;

remaining\_items[j] = \_problem1.solution[i];

j++;

}

}

/\* 2) shuffle the remaining items to ensure that remaining items

will be assigned at random positions \*/

for (i = 0; i < j; i++){

random = rand()%(j-i) + i;

temp = remaining\_items[i];

remaining\_items[i] = remaining\_items[random];

remaining\_items[random] = temp;

}

/\* 3) copy the shuffled remaining items at unassigned positions \*/

for (i = 0; i < j ; i++)

\_problem1.solution[unassigned\_positions[i]] = remaining\_items[i];

// crossover in our case is always possible

return true;

}

};

**2) Визначення мутації**

Мутація тут тільки простий обмін між 2-х елементами. Звичайно, ви можете собі уявити більш відповідну мутацію до проблеми яку ви маєте на руках.

class ProblemSwapMutation: public eoMonOp<Problem> {  
public:

bool operator()(Problem& \_problem) {

int i,j;

int temp;

// generate two different indices

i=rand()%n;

do

j = rand()%n;

while (i == j);

// swap

temp = \_problem.solution[i];

\_problem.solution[i] = \_problem.solution[j];

\_problem.solution[j] = temp;

return true;

}

};

1. **Виконання програми**

Тоді більше немає компоненти для перевизначення. Ось декларація основної програми. Компоненти червоного кольору є класи, які ми тільки що визначили.

#include <eo>

...

// Class involving a simple call to the function of initialisation of a solution

ProblemInit chromInit;

int popSize = 10;

eoPop<Problem> pop(popSize, chromInit); // Initialise the population

// Declaration of class wrapping the evaluation function of the QAP

ProblemEvalFunc eval;

// SELECT

// The robust tournament selection

int tSize=2; // size of the tournament

eoDetTournamentSelect<Problem> selectOne(tSize);

// is now encapsulated in a eoSelectPerc (entage)

eoSelectPerc<Problem> select(selectOne);// by default rate==1

ProblemXover Xover; // CROSSOVER ;

double pCross = 0.6; //probability for crossover

ProblemSwapMutation mutationSwap; // MUTATION

double pMut = 0.3; //probability for mutation

// The operators are encapsulated into an eoTRansform object

eoSGATransform<Problem> transform(Xover, pCross, mutationSwap, pMut);

// replacement engine

eoPlusReplacement<Problem> replace;

eoGenContinue<Problem> genCont(100); // number of maximum generation;

eoEasyEA<Problem> gga(genCont, eval, select, transform, replace);

// Apply algo to pop - that's it!

gga(pop);

Зверніть увагу на те, що для наших класів в червоному кольорі які ми тільки що визначили, не використали ніяких шаблонів.

**Висновок**

В дані розрахунковій роботі здійснено огляд фреймворку для евристичних і еволюційних алгоритмів ParadisEO. Проведено огляд програмного середовища, описано його основні функції та можливості. Проведений огляд наявних алгоритмів та проблем, які можна вирішувати з допомогою фреймворку. Фреймворк містить велику кількість різноманітних алгоритмів та задач, також наявні приклади до них. Наведено приклад використання фреймворку.