**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**

**іНСТИТУТ КОМП’ютерних НАУК та ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

### *Кафедра “Системи автоматизованого проектування”*



**JavaEvA/EvA** 

Звіт до РГР

з дисципліни “ Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні”

Виконав

Студент КНМ-14

Магеровський А.В.

Перевірив

Кривий Р.З.

Львів-2017

**МЕТА РОБОТИ**

Ознайомитися з основними теоретичними відомостями про фреймворк **JavaEvA/EvA**  для рішення задач, використовуючи генетичні алгоритми у мові програмування Java.

**ВСТУП**

EvA 2 (фреймворк Еволюційні алгоритми) являє собою всеосяжний евристичний оптимізаційний фреймворк з акцентом на еволюційні алгоритми, реалізовані в Java.

EvA 2 об'єднує кілька безкоштовних дериваційних методів оптимізації, переважно на основі популяцій, такі як еволюційна стратегія, генетичні алгоритми, диференціальна еволюція, часткова Swarm оптимізація, а також класичні методи, такі як мультипоточне сходження Хілла або імітаційне випалюваня.

EvA 2 призначена для двох груп користувачів. По-перше, для застосування користувачем, який нічого не знає про теорію еволюційних алгоритмів, але хоче, використовувати їх, щоб вирішити специфічну проблему. По-друге, для користувача, який хоче дослідити продуктивність різних оптимізаційних алгоритмів або хоче порівняти ефективність альтернативних або спеціалізованих еволюційних або евристичних операторів. Останній, як правило, знає більше про еволюційні або евристичні оптимізації і здатний розширити ЕVА 2, шляхом додавання специфічних стратегій оптимізацій.

Стандартний оптимізатор може бути визначений як більш-менш популярний алгоритм оптимізації з операторами і параметрами зумовлені в загальному так що користувач може використовувати його out-of-the-box і очікувати хороші результати оптимізації в цілому.

Звичайно, кожен експерт може мати власне уявлення про те, які параметри є кращими взагалі. Тому наведено два приклади використання за допомогою API. У будь-якому випадку, необхідний крок унаслідувати ЕVA2, базовий клас, що визначає цільову функцію. Можна ,знову ж таки, використовувати simpleprobs пакет. Проте, це робить необхідним використовувати клас-оболонку SimpleProblemWrapper, який відмінно підходить для безпосереднього використання графічного інтерфейсу, але може зробити програмування трохи складнішим. Тому рекомендовано тут перейти на один щабель вище в ієрархії і унаслідувати AbstractProblemDouble або AbstractProblemBinary.

**ПІДКЛЮЧЕННЯ**

Бібліотека підключається, як звичайний jar-файл.

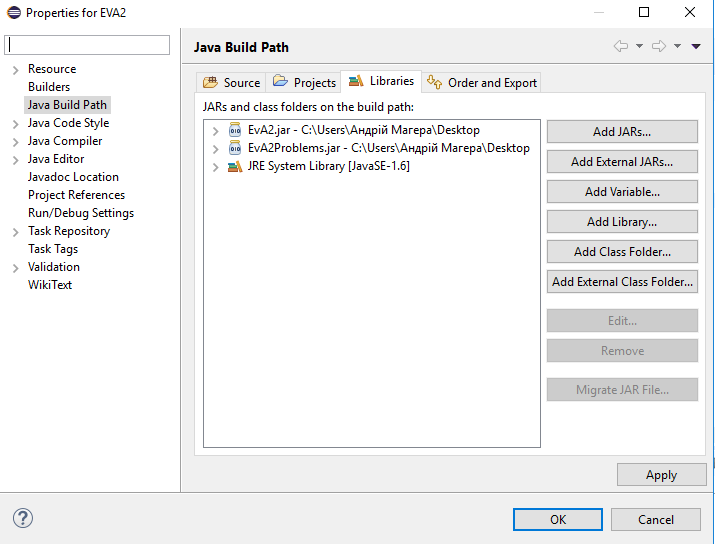


Рис.1.Результати підключення бібліотеки.

**ABSTRACT PROBLEM КЛАСИ**

Піддерево класу problem інкапсулює властивості цільових функцій, які можуть бути безпосередньо оптимізованими в рамках ЕVA 2, і починається клас AbstractOptimizationProblem. Основні властивості класу problem функціональні:

(1) Проблема сам по собі ініціалізована,

(2) Проблема знає, як ініціалізувати набір можливих рішень,

(3) Проблема оцінює набір потенційних рішень.

Для того, щоб реалізувати свою власну цільову функцію в ЕVA 2, рекомендовано наслідуватися від: AbstractProblemDouble або AbstractProblemBinary, в залежності від пріоритетної презентації даних.

Основні функції:

*public Object clone()*:Object оператор копіювання. Рекомендовано реалізувати копію конструктора public YourProblem (YourProblem о) і посилаючись до нього зсередини методу clone(). Можна викликати cloneObjects з AbstractProblemDouble, щоб скопіювати super- члени. Треба переконатися в тому, що copy-конструктор копіює всі необхідні члени змінні, які додані в свій клас.

*public void evaluate(AbstractEAIndividual individual):*Основний метод оцінки.  
Рекомендовано не перевантажувати його.

*public abstract double[] eval(double[] x):*Основний метод оцінки.Перевантажити  
його,щоб реалізувати бажану цільову функцію.Треба переконатися, що отримане рішення завжди однієї і тієї ж розмірності m. Навіть ,якщо проблема одновимірна, повертає масив довжини 1 з єдиним значенням фітнес-функції.   
  
*public abstract int getProblemDimension():*Повертає розмір problem в просторі рішення. Треба переконатися, що він завжди дорівнює n під час виконання оптимізації.Якщо реалізувати відповідний метод public void setProblemDimension(int n) і визначити клас всередині того ж пакету,можна змінити розмірність через графічний інтерфейс.

*public void initProblem():* Викликається перед оптимізацією в цілому. Якщо визначити змінні класу, які не є константами, рекомендується задати їм початкові значення в цій функції. Якщо перевизначити, переконайтеся, що викликаєте super-метод з перевизначеної версії.

*public void initPopulation(Population population):*Цей метод викликається перед  
оптимізацією з певною популяцією. Він ініціалізує популяцію до діапазону проблеми, використовуючи індивідуальний шаблон і викликає метод колекції Population для ініціалізації індивідів випадковим чином визначенимим всередині діапазону. Якщо треба перевизначити спосіб ініціалізації індивідів, рекомендовано перевизначити цей метод.

Клас AbstractProblemBinary. Двійковий варіант аналога до Abstract- ProblemDouble.. Головні відмінності полягають в тому, що немає діапазону визначення(діапазон завжди 0n - 1n), і що шаблон іншого типу , а саме GAIndividualBinaryData, і забезпечує Java BitSet як представлення даних замість double-вектора.Сигнатура методу eval змінюється відповідним чином, і для реалізації цільової функції ви повинні працювати з об'єктом BitSet.

Основні види алгоритмів:

|  |  |
| --- | --- |
| ID | Short Description |
| STD\_ES | A standard (15,50)-Evolution Strategy. |
| CMA\_ES | (15,50)-Evolution Strategy with Covariance Matrix Adaptation. |
| STD\_GA | Standard Genetic Algorithm with elitism |
| PSO | Particle Swarm Optimization with constriction. |
| DE | Di\_erential Evolution |
| TRIBES | Tribes: an adaptive PSO |
| RANDOM | Random search (Monte-Carlo) |
| HILLCL | Multi-start hill climbing |
| CL\_HILLCL | Clustering multi-start hill climbing |
| CBN\_ES | Clustering-based Niching ES |

Табл.1.Таблиця оптимізаційних стратегій

**OPTIMIZERFACTORY.**

Щоб отримати доступ до алгоритмів оптимізації за замовчуванням легко, є визначені в класі OptimizerFactory Це дозволяє визначити алгоритм оптимізації за допомогою ідентифікаційного номера і в основному має клас проблем і додатковий вихідний файл в якості вхідних даних. Наприклад, якщо у вас є клас з double - проблемами і просто хочете отримати одне рішення, потрібно використовувати optimizeToDouble(final int optType, AbstractOptimizationProblem problem, String outputFilePrefix) метод,який повертає рішення у вигляді double - вектора.

Таблиця 1 наводить доступні стратегії оптимізації. Короткий приклад показаний в лістингу 1, де PSO використовуються для оптимізації простої

гіпер-параболи (в 10 вимірах за замовчуванням).

Простий прилад використання OptimizerFactory

Лістинг1

**import** eva2.OptimizerFactory ;

**import** eva2.problems.F1Problem ;

**public** **class** TestingF1PSO {

**public** **static** **void** main ( String [ ] args ) {

F1Problem f1 = **new** F1Problem ( ) ;

**double** [ ] sol ;

OptimizerFactory.*setEvaluationTerminator* ( 50000 ) ;

sol = OptimizerFactory.*optimizeToDouble* ( OptimizerFactory .***PSO***, f1 , **null** ) ;

System.***out***.println( "Found solution : " ) ;

**for** ( **int** i =0; i<f1.getProblemDimension ( ) ; i++) System.***out***.print (sol[ i ] + " " ) ;

System.***out***.println ( ) ;

} ;

}

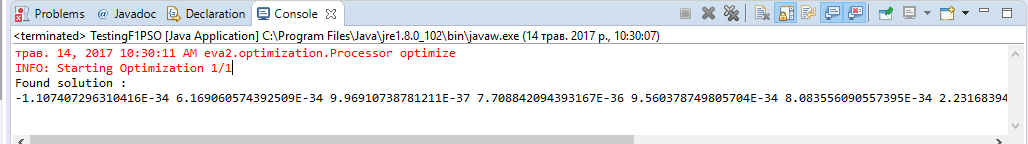


Рис.2.Результати виконання

**КРИТЕРІЙ ЗУПИНКИ**

Для того, щоб налаштувати критерії припинення процесу оптимізації,що почався через OptimizerFactory, використовується сімейство setTerminator.За замовчуванням terminator зупиняється на максимальній кількості фітнес оцінок, наприклад 10, 000. Для зміни числа оцінок, що треба виконанити, наприклад, 50, 000, викличте OptimizerFactory.setEvaluationTerminator (50000) перед початком оптимізації, як і в Лістингу1.

Для більш гнучких критеріїв завершення, існує кілька класів параметрів Terminator,ви можете використовувати і встановити їх безпосередньо за допомогою OptimizerFactory.setTerminator(term). Доступними є наступні варіанти:

*EvaluationTerminator:* Побудувати з максимальним числом оцінок. Параметри конструктора:

maxEval: ціле число оцінок, які будуть виконуватися як максимум.

*GenerationTerminator:* Припинити після заданого числа поколінь. Так як не всі алгоритми використовують постійні розміри популяцій, пропонується використовувати EvaluationTerminator переважно. Параметри конструктора:

gens: ціле число поколінь, які будуть виконуватися на максимумі.

*FitnessValueTerminator:* Мінімальне фітнес-значення (вектор) має бути досягнуто ,щоб припинити процес. Побудувати з double масивом, що представляє цільове фітнес-значення.Параметри конструктора:

v: double масив ,якого потрібно досягнути значення фітнес-вектора.

*FitnessConvergenceTerminator:* Припиняється як тільки навряд чи будь-яке поліпшення можливе в фітнес-значенні для певного періоду часу (період застою), визначається числом оцінки або покоління.

*CombinedTerminator:* При ефективній оптимізації, може знадобитися об'єднати кілька критеріїв в логічному шляху, тобто припиняти виконання, якщо 20, 000 оцінок були виконані або кращі фітнес-значення не змінюються протягом 1 000 оцінок. Щоб врахувати це, забезпечено клас CombinedTerminator який просто бере Terminator-сутності і комбінує їх логічно. Таким чином, термінатори можуть бути вкладені у відповідності до вимог.

Параметри конструктора:

t1: перший термінатор об'єднання.

t2: другий термінатор об'єднання.

bAnd: Логічний флаг, який вказує, чи слід використовувати з'єднувальну (AND, for bAnd == true) або диз'юнктивну (OR, for bAnd == false) комбінацію.

Зауваження по термінаторів:

• Зверніть увагу, що критерій припинення завжди перевіряються після однієї ітерації. Це означає, що, якщо число оцінок не кратно розміру популяції або розмір популяції є змінним, максимальне число оцінок може бути трохи перевищено. Те ж саме справедливо для часу гальмування в convergence- термінатора.

• Щодо convergence terminators: Зверніть увагу, що для плоского плато в фітнес-просторі, фітнес навряд чи може змінитися, поки є прогрес в просторі рішень. З іншого боку, зверніть увагу, що сильно нелінійні задачі може ледве змінитися в фенотипі, але значно змінитися в фітнесі. При використанні convergence terminators, пропонується встановити період застою досить високий.

Приклад використання критеріїв зупинки:

Лістинг2

**import** java.util.BitSet;

**import** eva2.problems.B1Problem;

**import** eva2.OptimizerFactory;

**import** eva2.optimization.OptimizationParameters;

**import** eva2.optimization.operator.selection.SelectTournament;

**import** eva2.optimization.operator.selection.SelectXProbRouletteWheel;

**import** eva2.optimization.operator.terminators.\*;

**import** eva2.optimization.population.\*;

**import** eva2.optimization.strategies.GeneticAlgorithm;

**public** **class** TestingGAB1 {

**public** **static** **void** main ( String [ ] args ) {

B1Problem b1 = **new** B1Problem ( ) ;

BitSet sol ;

// default go−parameter instance with a GA

OptimizationParameters gaParams = OptimizerFactory.*standardGA*(b1);

// add an evaluation terminator

gaParams.setTerminator (**new** EvaluationTerminator (1000) ) ;

// set a specific random seed

gaParams.setRandomSeed(2342);

// access the GA

GeneticAlgorithm ga = ( GeneticAlgorithm ) gaParams.getOptimizer();

// set no elitism

ga.setElitism( **false** ) ;

// set roulette wheel selection

ga.setParentSelection(**new** SelectXProbRouletteWheel()) ;

// set population size 150

ga.setPopulation(**new** Population ( 150 )) ;

// run optimization and print intermediate results to a file with given prefix

sol = OptimizerFactory.*optimizeToBinary*( gaParams , "ga−opt−results " ) ;

System.***out***.println( "Found solution : " ) ;

**for** ( **int** i =0; i<b1.getProblemDimension ( ) ; i++) System.***out***.print(sol.get( i )+" " ) ;

System.***out***.println( ) ;

} ;

}

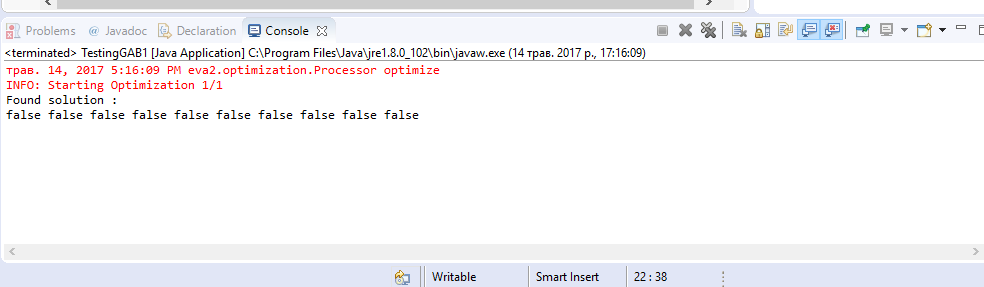


Рис.3.Результати виконання

**НАЛАШТУВАННЯ ОПТИМІЗАЦІЇ.**

Крім стандартних оптимізаторів, ви можете налаштувати параметри оптимізації вручну або перебирати різні параметри в циклі. Щоб зробити це, ви повинні отримати доступ до структури GOParameters і змінити її значення перед початком оптимізації. Примірник GOParameters містить всі необхідні настройки для виконання оптимізації: цільова функції, оптимізатор, випадкове початкове число, параметри постобробки і критерій припинення; в основному все, що також встановлюється через вікна інструментальних засобів графічного інтерфейсу користувача. Для того, щоб змінити специфічні налаштування досить змінити екземпляр GOParameters згенерований OptimizerFactory, а потім почати оптимізацію за допомогою цього зміненого параметра примірника.

Приклади налаштування оптимізації:

Лістинг3

**import** eva2.OptimizerFactory ;

**import** eva2.problems.F1Problem ;

**import** eva2.optimization.operator.terminators.\*;

**import** eva2.optimization.operator.terminators.PopulationMeasureTerminator.ChangeTypeEnum;

**import** eva2.optimization.operator.terminators.PopulationMeasureTerminator.DirectionTypeEnum;

**import** eva2.optimization.operator.terminators.PopulationMeasureTerminator.StagnationTypeEnum;

**public** **class** TestingTerminators {

**public** **static** **void** main ( String [ ] args ) {

F1Problem f1 = **new** F1Problem();

**double**[] sol ;

// A combined terminator for fitness and phenotype convergence

CombinedTerminator convT = **new** CombinedTerminator(

// fitness-based stagnation period, absolute threshold, consider stagnation

// in both direction (per dim.) or w.r.t. minimization only

**new** FitnessConvergenceTerminator(0.0001, 1000, StagnationTypeEnum.***fitnessCallBased***, ChangeTypeEnum.***absoluteChange***, DirectionTypeEnum.***decrease***),

**new** PhenotypeConvergenceTerminator(0.0001, 1000, StagnationTypeEnum.***fitnessCallBased***, ChangeTypeEnum.***absoluteChange***, DirectionTypeEnum.***bidirectional***), **true**

);

// Adding an evaluation terminator with OR to the convergence criterion

OptimizerFactory.*setTerminator*(**new** CombinedTerminator(

**new** EvaluationTerminator(20000),

convT,

**false**));

sol = OptimizerFactory.*optimizeToDouble*(OptimizerFactory.***PSO***, f1, **null**);

System.***out***.println(OptimizerFactory.*lastEvalsPerformed*()

+ " evals performed. "

+ OptimizerFactory.*terminatedBecause*()

+ " Found solution: ");

**for** (**int** i=0; i<f1.getProblemDimension(); i++) System.***out***.print(sol[i] + " ");

System.***out***.println ();

} ;

}

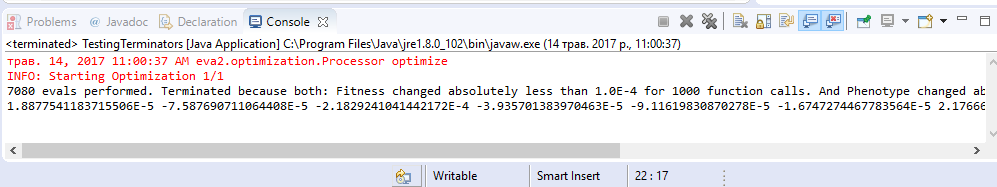


Рис.4.Результати виконання

У лістингу 4, (1 + 5) CMA-ES є налаштованим і працює на простій бімодальній цільовій функції з глобальним оптимумом поблизу (1,7 / 0) і локальним поруч (-1,44 / 0). (1 + 5) - CMA-ES є потужним і знайде глобальний оптимум більшості випадків. Іноді, однак, через відносно високий тиск селекції і еластичну стратегію,вона буде сходитися в локальний оптимум, в залежності від випадкової ініціалізації.

Лістинг4

**import** eva2.OptimizerFactory;

**import** eva2.optimization.OptimizationParameters;

**import** eva2.optimization.individuals.AbstractEAIndividual;

**import** eva2.optimization.operator.crossover.CrossoverESDefault;

**import** eva2.optimization.operator.mutation.InterfaceMutation;

**import** eva2.optimization.operator.mutation.MutateESCovarianceMatrixAdaption;

**import** eva2.optimization.operator.terminators.EvaluationTerminator;

**import** eva2.optimization.strategies.EvolutionStrategies;

**import** eva2.problems.FM0Problem;

**public** **class** TestingPlusCMAES {

**public** **static** **void** main ( String [ ] args ) {

// a simple bimodal target function , two optima

FM0Problem fm0 = **new** FM0Problem ( ) ;

AbstractEAIndividual bestIndy ;

// create standard ES parameters

OptimizationParameters esParams = OptimizerFactory.*standardES*(fm0) ;

esParams.setTerminator(**new** EvaluationTerminator(2000)) ;

esParams.setRandomSeed(0);

// set evolutionary operators and probabilities

AbstractEAIndividual.*setOperators*(

fm0.getIndividualTemplate() ,

**new** MutateESCovarianceMatrixAdaption() , 0.9 ,

**new** CrossoverESDefault() , 0.1 ) ;

// access the ES

EvolutionStrategies es = (EvolutionStrategies)esParams.getOptimizer ( ) ;

// set a (1+5) selection strategy

es.setMu(1);

es.setLambda(5);

es.setPlusStrategy(**true**);

// run optimization and retrieve winner individual

bestIndy = ( AbstractEAIndividual ) OptimizerFactory.*optimizeToInd*( esParams , **null** ) ;

System.***out***.println ( "Found solution : "

+ AbstractEAIndividual.*getDefaultDataString* ( bestIndy ) ) ;

} ;

}

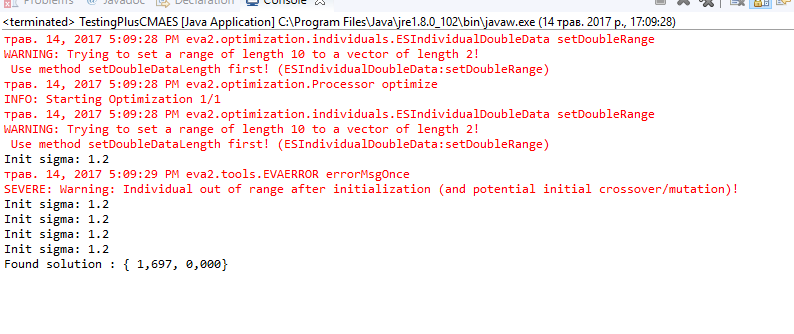


Рис.5.Результати виконання

**ВИСНОВКИ**

Виконуючи розрахунково-графічну роботу я ознайомився із теоретичними відомостями про фреймворк **JavaEvA/EvA** для мови програмування Java. Даний фреймворк надає зручний спосіб для розв’язання задач, використовуючи генетичні алгоритми.

Ознайомлення із засобом **JavaEvA/EvA** показав, що розв'язувати задачі, використовуючи генетичні алгоритми можна просто і не прописуючи багато коду,не вникаючи в дрібні деталі побудови генетичних алгоритмів. Для отримання результату достатньо оформити у потрібному форматі вхідні дані та передати їх у відповідні методи класів фреймворка. Якщо користувачу необхідно певним чином модифікувати деякі алгоритми він може це зробити, здійснивши операцію @override потрібного методу і використати його для рішення задачі.