|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| СПКм-12 | РГР | **ECJ** |  |  |
| Цедуляк Т. Б. | |
| № залікової: 1508517 | |
| Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні | | Викладач: | |
| Кривий Р.З. | |

1. **Мета роботи**

Ознайомитись з можливостями ECJ. Здобути навики використання програмного продукту для вирішення поставлених задач.

1. **Короткі теоретичні відомості**

**Про ECJ**

ECJ це фрейморк спрямований на роботу з еволюційними алгоритмами, який написаний на Java. Фреймворк був розроблений для потреб великих, експерементальних обчислень і надає інструменти, які забезпечують безліч популярних алгоритмів, але з особливим акцентом у бік генетичного програмування. ECJ є фреймворком з відкритим вихідним кодом.

ECJ уже налічує більше десяти років ефективної та стабільної роботи і є стабільною платформою, в якій було виявлено відносно мало серйозних помилок протягом багатьох років. Його конструкція легко розширюється новими юнітами, в тому числі і юнітами для роботи з багатокритеріальними алгоритми оптимізації, острівних моделей, стаціонарні і стратегії еволюції методи, та багато інших. Система широко використовується в генетичному програмуванні і є досить популярним в цілому світі.

ECJ був розроблений для великих проектів і надає широкий спектр послуг. Основним джерелом натхнення ECJ є Lil-gp, якій він багато чим зобов'язаний. Робота над ECJ почалася восени 1998 року після того, як з досвідом Lil-gp в еволюційно модельованих команд робо футболу. Цей проект потребував значних модифікацій (Lil-gp), зокрема для виконання паралельних оцінок, багаторазове використання модулів, і суворої типізації. Такі аргументи ясно дали зрозуміти, що Lil-pg не може розвиватись далі без значних зусилью. ECJ призначений для щонайменше, десяти ічного терміну корисного використання, і він добре зарекомендував себе і досі.

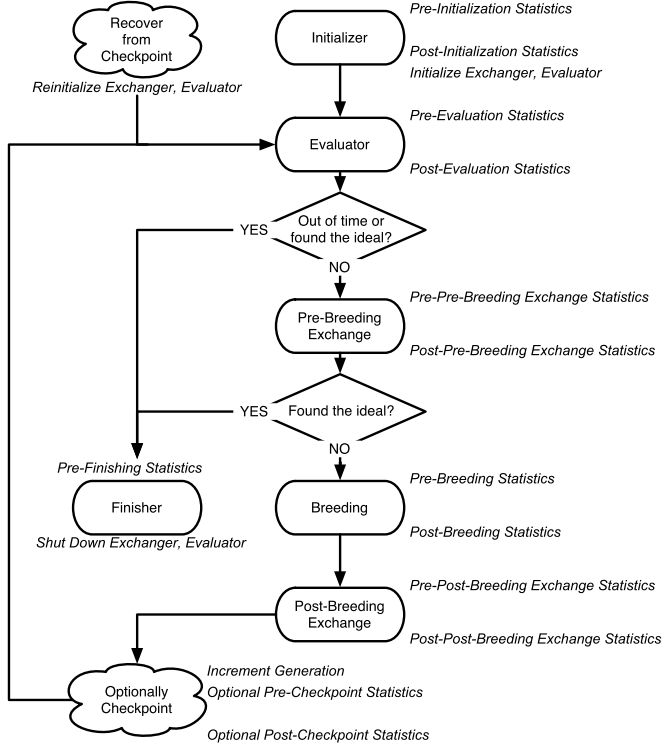


Рис. 1. Високорівневі операції класу SimpleEvolutionState ECJ.

**Огляд**

**Петлі верхнього рівня**

ECJ наслідує весі стани еволюційного алгоритму з одного примірника підкласу EvolutionState. Це дозволяє ECJ серіалізовати увест стан системи до контрольної точки, точки відновлення. На малюнку 1.1 показана петля верхнього рівня простого нащадка EvolutionState у ітерації циклу між розведенням і оцінкою, після кожного з них. Статистична петля викликається перед і після кожного періоду розмноження, оцінки і обміну, а також до і після ініціалізації популяції і "очищення" (очищення до виходу з програми).

Конкретні варіанти цих Singleton об'єктів визначаються експериментатором, хоча ми пропонуємо варіанти, які виконують загальні їх набори.

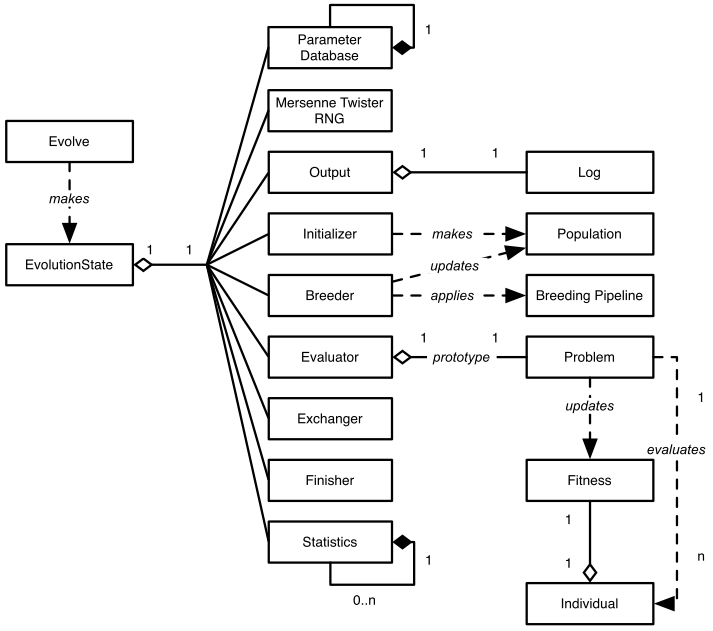


Рис. 1.2. Оператори верхнього рівня і підкласи в EvolutionState, а також їх зв'язок з окремими об'єктами.

**Оцінка.** Оцінювач проводить оцінку популяції шляхом пропускання одного або кількох індивідів до проблеми підкласу, який оцінювач клонував від свого прототипу.

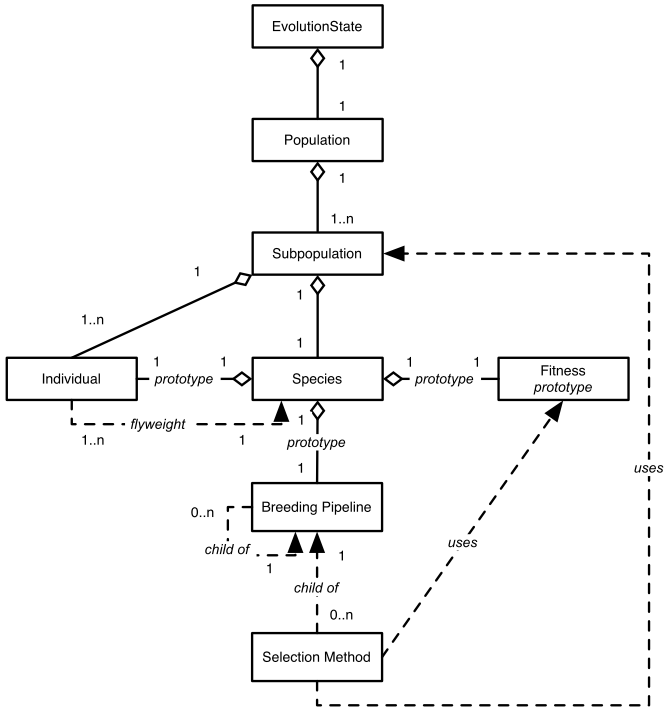


Рис. 1.3. Об'єкти верхнього рівня даних що використовується в процесі еволюції.

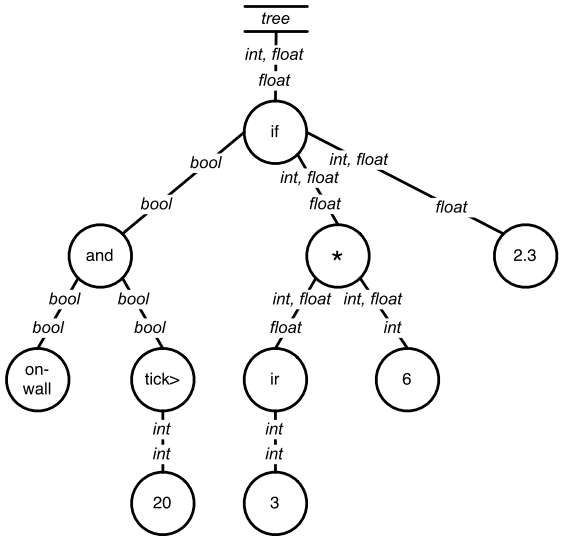


Рис. 1.4. Типізоване генетичне дерево синтаксичного аналізу.

Оцінка також може бути зроблено в багатопотоковому режимі, без блокування, використовуючи окремий потік для кожної проблеми. Індивіди можуть також піддаватися багаторазовій оцінці.

У більшості проектів з використанням ECJ, основне завдання полягає в тому, щоб побудувати відповідний підклас Problem. Завдання Problem полягає в тому, щоб оцінити індивідів і встановити їх придатність до певних вимог. Класи проблема також можуть повідомити, коли було виявлено ідеального індивіда.

Якість генератора випадкових чисел має важливе значення для системи стохастичної оптимізації.

**З коробки ECJ забезпечує підтримку наступних алгоритмів:**

• Generational алгоритми: (μ, λ) і (μ + λ) Еволюція стратегій, генетичний алгоритм.

• Просторово-похідний еволюційний алгоритм

• Випадкові зхрещування

• Багатоцільова оптимізація, в тому числі NSGA-II і алгоритмів SPEA2.

• Оцінка багатозадачності і розведення.

• Паралельний синхронний і асинхронний алгоритм роботи на мережі комп'ютерів.

• Внутрішні синхронні Island Моделі всередині популяції.

• Асинхронна еволюція.

• Гнучка еволюція, під час якої віддалені машини опрацьовують свої власні міні-еволюційні процеси.

• Внутрішні синхронні Island моделі всередині популяції.

• Метаеволюції

• Велика кількість селекцй і операторів

1. **Процес встановлення ECJ**

Фреймворк ECJ написано на Java, це означає, що він не вимагає будь-якого додаткового програмного забезпечення. Все що необхідно, це мати інстальованим Java JDK 1.5 або більш пізню версію. Вихідний код ECJ поставляється в пакеті ecj.22.jar, який можна завантажити з наступного поситання <https://cs.gmu.edu/~eclab/projects/ecj/>.

Нижче буде представлено спосіб інсталяції ECJ використовуючи Eclipse IDE.

Після завантаження фреймворк матиме виляд JAR бібліотеки рис. 1.5.

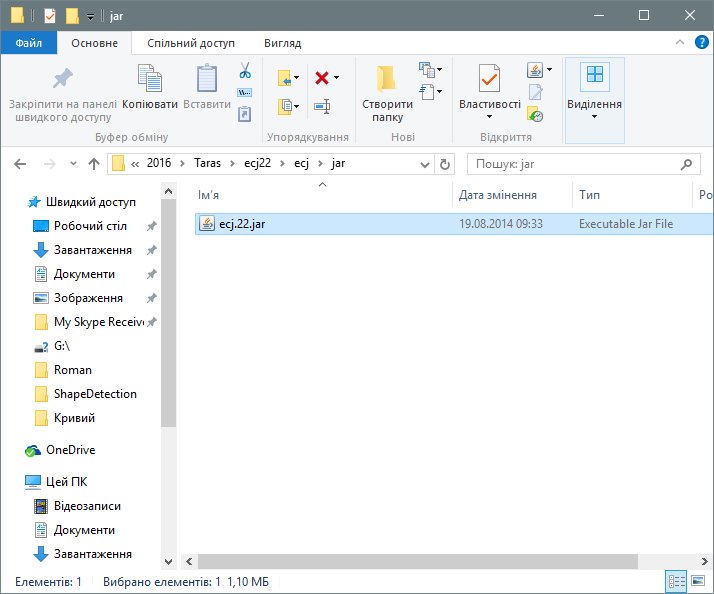


Рис. 1.5. JAR бібліотека ECJ після завантаження

Наступним кроком є підключення ECJ до середовища Eclipse IDE. Для цього необхідно виконати наступні кроки:

* Виберіть *File* → *Create* → *New Java Project*.
* Введіть ім'я проекту (наприклад, ECJproject) і натисніть на кнопку *Finish*.
* Натисніть правою кнопкою миші на папці з проектом (в провіднику Eclipse) та оберіть *Properties* (рис. 1.6).
* Оберіть розділ *Java Build Pat*.
* В меню праворуч натисніть кнопку *Add External JARs..* (рис. 1.7)
* Вкажіть шлях до папки, що містить JAR бібліотеку ECJ та натисніть відкрити (рис. 1.8).

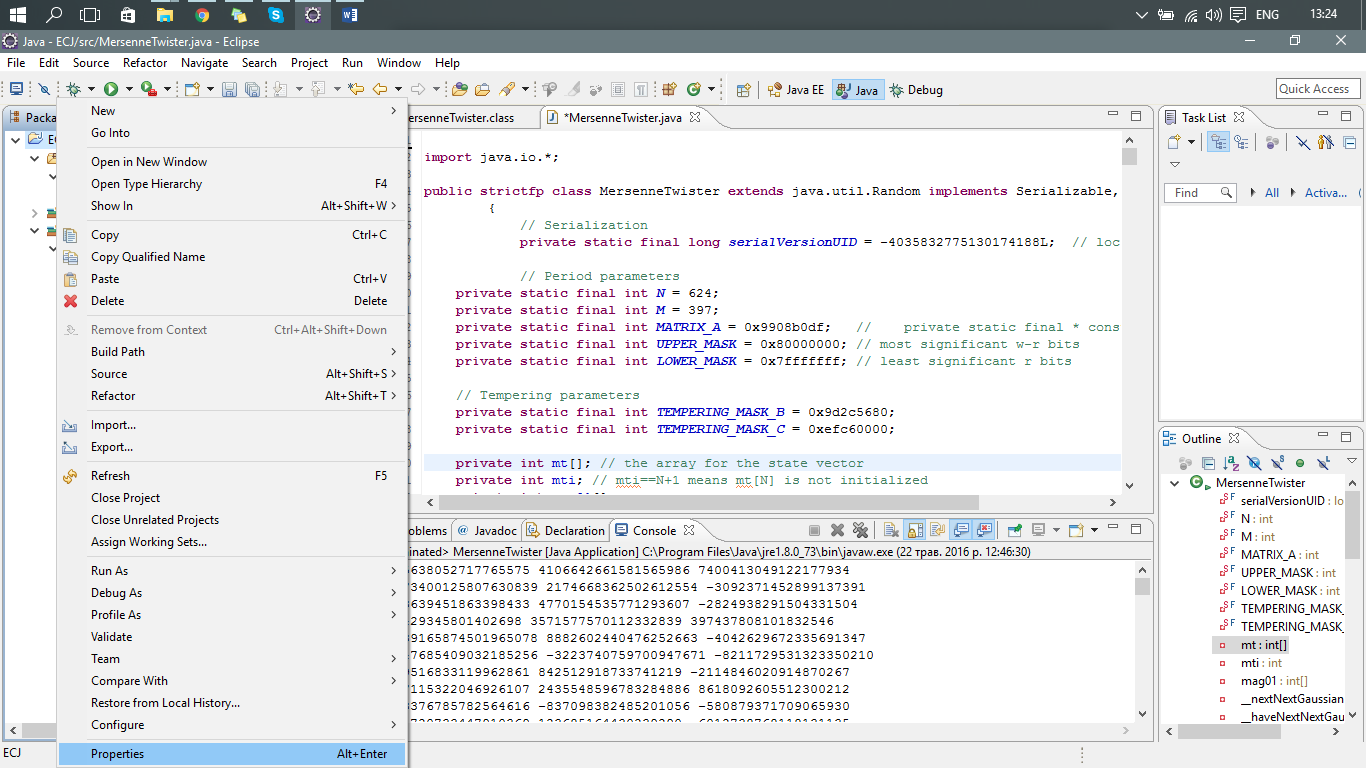


Рис. 1.6. Вікно *Project Properties*

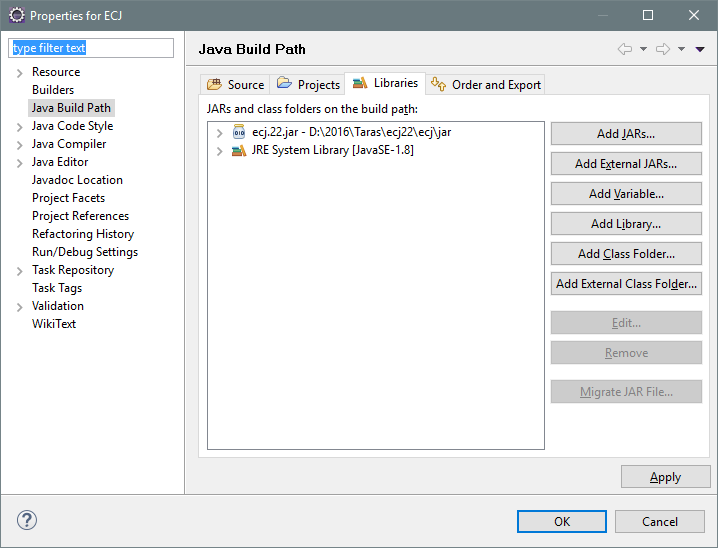


Рис. 1.7. Вікно *Java Build Pat*

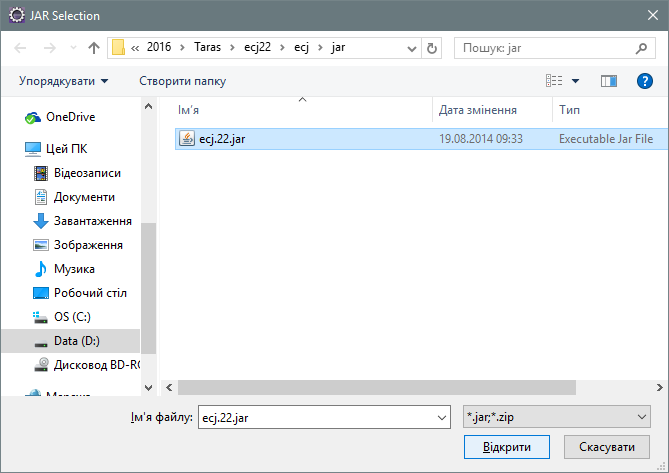


Рис. 1.8. *JAR Selection*

Мною було обране тестове завдання реалізації алгоритму вирішення проблеми MaxOnes, код програми представлений в додатку А.

Проблема MaxOnes - це дуже проста проблема, де еволюція використовується, щоб знайти конкретний «ген». Ген, по суті, являє собою фрагмент тексту заповнений випадковими двійковими значенями (0 або 1), бінарний рядок.

Приклад гену випадкових індивідів:

10111110011010110001

Ціль - ген який нам необхідно отримати:

11111111111111111111

Мета поставленої задачі - знайти бінарний рядок, який буде ідентичним до поставленої мети. Так як ми не можемо просто задати нашу рядок ідентичним до мети, тому для нашої популяції ми повинні використовувати методи, які є доступні для еволюційного алгоритму " evolve ", поки врешті-решт, ми не досягнемао поставленої мети.

Ця проблема відрізняється від більш просунутих проблем в тому, що еволюційний алгоритм (EA) намагається безпосередньо знайти ген-мішень. У більш складних завдань ЕА перетворювати гени в "фенотипів", а потім фенотип порівнюють з геномом який необхідно отримати.

Для того, щоб знайти індивідів, які потомство ми використовуємо механізми parent selection, і для створення нащадків, які ми використовуємо функцію reproduction.

У даному прикладі проблема відбору індивіда обчислюється шляхом простого порівняння індивіда що вже проеволюціонува та індивіда що поставлений як мета. Придатність (Fitness) це значення від 0,0 до 1,0, що відображає наскільки поточний індивід є схожим з шуканим. Це працює наступним чином, ми даємо одне очко за правильне значення в гені, і наступним поділом на довжину гена.

Розглянемо приклад ручної реалізації функції придатності (Fitness)

Початковий індивід:

10111110011010110001 – значення придатності для нього складатиме 0.6 бо порівняно з цільовим індивідом (11111111111111111111) він має 12 з 20 спільних генів, і 20/12=0.6

Резултати виконання програми представлені в додатку Б.

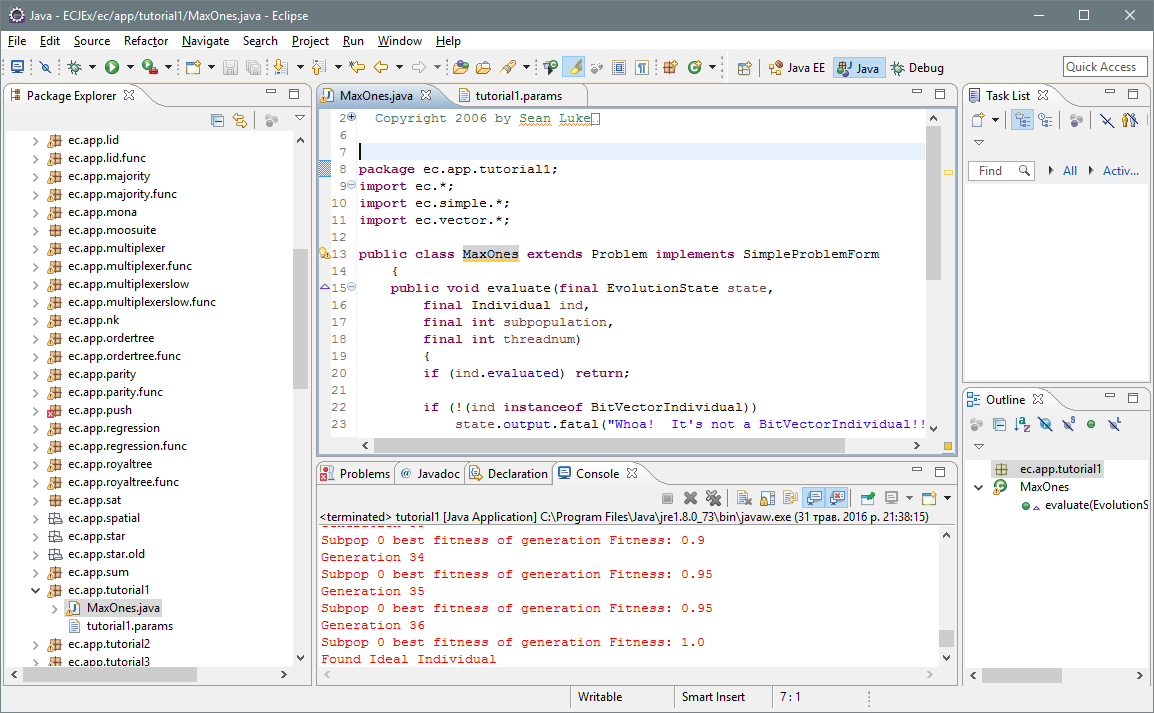


Рис.1.9 Представлення реалізації програми MaxOnes в ECJ

**Висновок:** під час виконання розрахунково-графічної роботи, я ознайомилась з програмним продуктом ECJ, якй є надзвичайно зручним та ефективним інструментом для розв’язання різного роду задач, використовуючи евристичні та генетичні алгоритми. Фреймворк містить велику кількість вже реалізованих алгоритмів.

Перевага використання цієї програми перш за все в тому, що для розв’язання задач немає необхідності реалізовувати певний алгоритм самостійно, а на томість можна використати чи модифікувати уже реалізований. Для цього фреймворк надає відкритий код своїх реалізацій алгоритмів.

**Додаток А.**

Реалізація генетичного алгоритму MaxOnes

package ec.app.tutorial1;

import ec.\*;

import ec.simple.\*;

import ec.vector.\*;

public class MaxOnes extends Problem implements SimpleProblemForm

{

public void evaluate(final EvolutionState state,

final Individual ind,

final int subpopulation,

final int threadnum)

{

if (ind.evaluated) return;

if (!(ind instanceof BitVectorIndividual))

state.output.fatal("Whoa! It's not a BitVectorIndividual!!!",null);

int sum=0;

BitVectorIndividual ind2 = (BitVectorIndividual)ind;

for(int x=0; x<ind2.genome.length; x++)

sum += (ind2.genome[x] ? 1 : 0);

if (!(ind2.fitness instanceof SimpleFitness))

state.output.fatal("Whoa! It's not a SimpleFitness!!!",null);

((SimpleFitness)ind2.fitness).setFitness(state,

/// ...the fitness...

sum/(double)ind2.genome.length,

///... is the individual ideal? Indicate here...

sum == ind2.genome.length);

ind2.evaluated = true;

}

}

**Додаток Б.**

Результати виконання алгоритму

Generation: 0

Best Individual:

Subpopulation 0:

Evaluated: T

Fitness: 0.6

10111110011010110001

Generation: 1

Best Individual:

Subpopulation 0:

Evaluated: T

Fitness: 0.6

10111110011010110001

Generation: 2

Best Individual:

Subpopulation 0:

Evaluated: T

Fitness: 0.6

10111110011010110001

Generation: 3

Best Individual:

Subpopulation 0:

Evaluated: T

Fitness: 0.7

10111110011011110110

…………………………………………………………………….

Generation: 33

Best Individual:

Subpopulation 0:

Evaluated: T

Fitness: 0.9

11111110111111111011

Generation: 34

Best Individual:

Subpopulation 0:

Evaluated: T

Fitness: 0.95

11111111111111111011

Generation: 35

Best Individual:

Subpopulation 0:

Evaluated: T

Fitness: 0.95

11111111111111111011

Generation: 36

Best Individual:

Subpopulation 0:

Evaluated: T

Fitness: 1.0

11111111111111111111

Best Individual of Run:

Subpopulation 0:

Evaluated: T

Fitness: 1.0

11111111111111111111