**Отчет лабы 3**

932003

Ву Нгок Зуй Хань

До За Бать

**1. Цель работы.**

- Генетические алгоритмы обычно используются для создания высококачественных решений задач оптимизации и поиска, полагаясь на биологически вдохновленные операторы, такие как мутация, скрещивание и отбор.

**2. Постановку задачи.**

- Проблемы, которые кажутся особенно подходящими для решения с помощью генетических алгоритмов, включают задачи составления расписания и планирования, и многие программные пакеты для планирования основаны на ГА. ГА также применялись в инженерии. Генетические алгоритмы часто применяются в качестве подхода к решению задач глобальной оптимизации.

- Как правило, генетические алгоритмы могут быть полезны в проблемных областях со сложным ландшафтом пригодности, поскольку смешивание, то есть мутация в сочетании с скрещиванием, предназначено для смещения популяции от локальных оптимумов, в которых традиционный алгоритм восхождения на холм может застрять. in. Заметим, что обычно используемые операторы кроссовера не могут изменить какую-либо однородную совокупность. Только мутация может обеспечить эргодичность всего процесса генетического алгоритма.

- Примеры задач, решаемых с помощью генетических алгоритмов, включают: зеркала, предназначенные для направления солнечного света в солнечный коллектор, антенны, предназначенные для приема радиосигналов в космосе, методы ходьбы для компьютерных фигур, оптимальное проектирование аэродинамических тел в сложных полях течения.

**3. Метод решения задачи.**

- Инициализация

Размер популяции зависит от характера проблемы, но обычно содержит несколько сотен или тысяч возможных решений. Часто начальная популяция генерируется случайным образом, допуская весь диапазон возможных решений. Иногда решения могут быть «засеяны» в областях, где оптимальные решения могут быть найдены.

- Выбор

В каждом последующем поколении часть существующей популяции отбирается для выведения нового поколения. Индивидуальные решения выбираются с помощью процесса, основанного на пригодности, где, как правило, с большей вероятностью будут выбраны более подходящие решения. Некоторые методы отбора оценивают пригодность каждого решения и предпочтительно выбирают лучшие решения. Другие методы оценивают только случайную выборку населения, поскольку первый процесс может занять очень много времени.

Функция пригодности определяется генетическим представлением и измеряет качество представленного решения. Фитнес-функция всегда зависит от проблемы. Например, в задаче о рюкзаке нужно максимизировать общую стоимость предметов, которые можно поместить в рюкзак некоторой фиксированной вместимости. Представление решения может быть массивом битов, где каждый бит представляет отдельный объект, а значение бита (0 или 1) показывает, находится ли объект в рюкзаке. Не всякое такое представление справедливо, так как размеры предметов могут превышать вместимость рюкзака. Пригодность решения - это сумма значений всех объектов в рюкзаке, если представление верно, или 0 в противном случае.

В некоторых задачах трудно или даже невозможно определить выражение пригодности; в этих случаях может использоваться моделирование для определения значения функции пригодности фенотипа (например, вычислительная гидродинамика используется для определения сопротивления воздуха транспортного средства, форма которого закодирована как фенотип), или даже используются интерактивные генетические алгоритмы.

- Генетические операторы

Следующим шагом является создание популяции решений второго поколения из выбранных с помощью комбинации генетических операторов: скрещивания и мутации.

Для каждого нового решения, которое будет получено, для размножения выбирается пара «родительских» решений из пула, выбранного ранее. При создании «дочернего» решения с использованием описанных выше методов кроссовера и мутации создается новое решение, которое обычно имеет многие характеристики своих «родителей». Для каждого нового потомка выбираются новые родители, и процесс продолжается до тех пор, пока не будет сгенерирована новая популяция решений соответствующего размера. Хотя методы воспроизводства, основанные на использовании двух родителей, больше «вдохновлены биологией», некоторые исследования показывают, что более двух «родителей» генерируют хромосомы более высокого качества.

Эти процессы в конечном итоге приводят к популяции хромосом следующего поколения, которая отличается от исходного поколения. Как правило, при этой процедуре средняя приспособленность популяции повышается, поскольку для размножения отбираются только лучшие организмы из первого поколения, а также небольшая доля менее приспособленных растворов. Эти менее подходящие решения обеспечивают генетическое разнообразие в генетическом пуле родителей и, следовательно, обеспечивают генетическое разнообразие следующего поколения детей.

Хотя скрещивание и мутация известны как основные генетические операторы, в генетических алгоритмах можно использовать и другие операторы, такие как перегруппировка, колонизация-вымирание или миграция.

Стоит настроить такие параметры, как вероятность мутации, вероятность кроссовера и размер популяции, чтобы найти разумные настройки для рассматриваемого класса задач. Очень небольшая частота мутаций может привести к генетическому дрейфу (который не является эргодичным по своей природе). Слишком высокая скорость рекомбинации может привести к преждевременной сходимости генетического алгоритма. Слишком высокая частота мутаций может привести к потере хороших решений, если только не используется элитарный отбор. Адекватный размер популяции обеспечивает достаточное генетическое разнообразие для решения поставленной задачи, но может привести к пустой трате вычислительных ресурсов, если установить значение, превышающее требуемое.

- Эвристика

В дополнение к основным операторам, указанным выше, могут использоваться другие эвристики, чтобы сделать вычисления более быстрыми или надежными. Эвристика видообразования наказывает переход между решениями-кандидатами, которые слишком похожи; это способствует разнообразию населения и помогает предотвратить преждевременную конвергенцию к менее оптимальному решению.

- Прекращение

Этот процесс генерации повторяется до тех пор, пока не будет достигнуто условие завершения. Общие условия завершения:

+ Найдено решение, удовлетворяющее минимальным критериям

+ Достигнуто фиксированное количество поколений

+ Выделенный бюджет (время вычислений/деньги) достигнут

+ Пригодность решения с наивысшим рейтингом достигает или достигла плато, так что последовательные итерации больше не дают лучших результатов.

+ Ручной осмотр

+ Комбинации вышеперечисленногоe

**4. Структурную схему алгоритма.**

Формирование

начальной

популяции

Результат

Оценивание

популяции

Селекция

Скрещивание

Мутация

**5. Листинг программы.**

Сгенерировать с помощью генетического алгоритма слово “МИР”

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

#define POPULATION\_SIZE 100

const string GENES = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOP"\

"QRSTUVWXYZ 1234567890, .-;:\_!\"#%&/()=?@${[]}";

const string TARGET = "World";

int random\_num(int start, int end)

{

int range = (end-start)+1;

int random\_int = start+(rand()%range);

return random\_int;

}

char mutated\_genes()

{

int len = GENES.size();

int r = random\_num(0, len-1);

return GENES[r];

}

string create\_gnome()

{

int len = TARGET.size();

string gnome = "";

for(int i = 0;i<len;i++)

gnome += mutated\_genes();

return gnome;

}

class Individual

{

public:

string chromosome;

int fitness;

Individual(string chromosome);

Individual mate(Individual parent2);

int cal\_fitness();

};

Individual::Individual(string chromosome)

{

this->chromosome = chromosome;

fitness = cal\_fitness();

};

Individual Individual::mate(Individual par2)

{

string child\_chromosome = "";

int len = chromosome.size();

for(int i = 0;i<len;i++)

{

float p = random\_num(0, 100)/100;

if(p < 0.45)

child\_chromosome += chromosome[i];

else if(p < 0.90)

child\_chromosome += par2.chromosome[i];

else

child\_chromosome += mutated\_genes();

}

return Individual(child\_chromosome);

};

int Individual::cal\_fitness()

{

int len = TARGET.size();

int fitness = 0;

for(int i = 0;i<len;i++)

{

if(chromosome[i] != TARGET[i])

fitness++;

}

return fitness;

};

bool operator<(const Individual &ind1, const Individual &ind2)

{

return ind1.fitness < ind2.fitness;

}

int main()

{

srand((unsigned)(time(0)));

int generation = 0;

vector<Individual> population;

bool found = false;

for(int i = 0;i<POPULATION\_SIZE;i++)

{

string gnome = create\_gnome();

population.push\_back(Individual(gnome));

}

while(! found)

{

sort(population.begin(), population.end());

if(population[0].fitness <= 0)

{

found = true;

break;

}

vector<Individual> new\_generation;

int s = (10\*POPULATION\_SIZE)/100;

for(int i = 0;i<s;i++)

new\_generation.push\_back(population[i]);

s = (90\*POPULATION\_SIZE)/100;

for(int i = 0;i<s;i++)

{

int len = population.size();

int r = random\_num(0, 50);

Individual parent1 = population[r];

r = random\_num(0, 50);

Individual parent2 = population[r];

Individual offspring = parent1.mate(parent2);

new\_generation.push\_back(offspring);

}

population = new\_generation;

cout<< "Generation: " << generation << "\t";

cout<< "String: "<< population[0].chromosome <<"\t";

cout<< "Fitness: "<< population[0].fitness << "\n";

generation++;

}

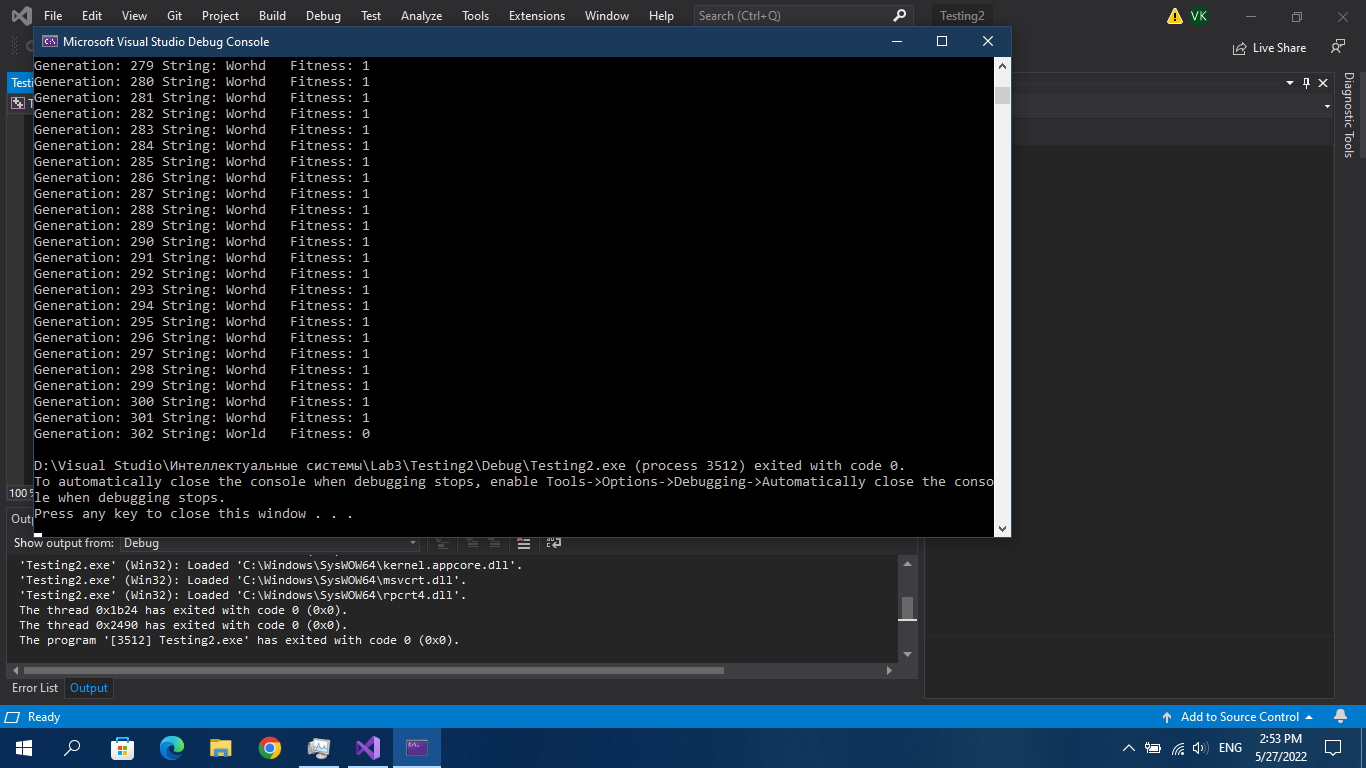
cout<< "Generation: " << generation << "\t";

cout<< "String: "<< population[0].chromosome <<"\t";

cout<< "Fitness: "<< population[0].fitness << "\n";

}

**6. Результаты работы генетического алгоритмы.**



**7. Выводы.**

- В этом отчете представлен структурированный и объясненный взгляд на генетические алгоритмы. Некоторые генетические операторы предназначены для представления. Однако они не применимы к исследовательским областям. Роль генетических операторов, таких как кроссинговер, мутация и отбор, в облегчении преждевременной конвергенции широко изучается. Есть много преимуществ использования ГА в других областях исследований и метаэвристических алгоритмов.