**Министерство образования и науки Российской Федерации**

**ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина»**

**Кафедра вычислительных методов и уравнений математической физики**

Оценка работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель от УрФУ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Раскраска графа с помощью генетического алгоритма.

ОТЧЕТ

по учебной практике

Руководитель от УрФУ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Лукач Ю.С.

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Островский С.В.

Направление подготовки 02.03.03 Математическое обеспечение и

администрирование информационных систем

Группа РИ-330901

Екатеринбург 2016

Оглавление

[**Введение** 3](#_Toc460612157)

[**Генетический алгоритм** 5](#_Toc460612158)

[**Описание работы программы** 7](#_Toc460612159)

[**Результаты работы** 9](#_Toc460612160)

[**Вывод** 11](#_Toc460612161)

[**Список литературы** 12](#_Toc460612162)

[**Приложение** 13](#_Toc460612163)

# **Введение**

В данной работе рассматривается задача о многомерном рюкзаке. Пусть существует N предметов, каждый из которых имеет стоимость и размеры , где , . Пусть также существует рюкзак с ограничениями по вместимости по измерениям . Требуется максимизировать сумму

где при условии

для всех .

Данная задача является модификацией классической задачи о рюкзаке, поставленной ещё в 19 веке Джорджем Мэттьюсом. Семейство задач об упаковке рюкзака является одним из наиболее популярных задач в силу простоты формулировки и сложности решения. И классическая задача, и её модификации являются NP-полными.

Решение задач такого рода классическими методами невозможно без существенных затрат времени и/или машинных ресурсов, поэтому для решения часто применяются эвристические алгоритмы. Одним из наиболее перспективных семейств таких алгоритмов являются методы машинного обучения, в том числе - эволюционные методы оптимизации. Одним из таких методов являются генетические алгоритмы, использование которых для решения данной задачи является эффективным за счёт простоты формализации. Целью работы является применение генетического алгоритма к заполнению многомерного рюкзака.

Во время выполнения работы были решены следующие задачи:

1. Ознакомление с проблемой заполнения многомерного рюкзака.
2. Ознакомление с генетическими алгоритмами.
3. Реализация алгоритма с помощью одного из языков программирования (был выбран C#).
4. Тестирование алгоритма на наборе примеров, позволяющее определить его эффективность.

Объектом исследования является генетический алгоритм.

Предметом исследования является применение генетического алгоритма к задаче о многомерном рюкзаке.

Данная проблема актуальна, так как задача о многомерном рюкзаке находит свое применение в различных отраслях. Так, например, эта задача может использоваться как модель для решения проблемы размещения грузов в замкнутом пространстве или оптимальной раскройки ткани. Также решение задачи о рюкзаке находит применение в криптографии как база для алгоритма асимметричного шифрования.

# **Генетический алгоритм**

Генетические алгоритмы являются семейством в множестве эвристических алгоритмов. Эвристические алгоритмы – алгоритмы, для которых их корректность строго не доказана, либо наоборот, доказана их некорректность в некоторых случаях.

Генетический алгоритм моделирует естественные процессы эволюции популяции.

Решение задачи с помощью такого алгоритма требует нескольких предварительных этапов:

1. Выбор кодирования генотипа.

На этом этапе нужно выбрать способ кодирования генотипа, который будет удобен для данной задачи. Такой генотип однозначно определяет фенотип, однозначно моделируя сущность, рассматриваемую в задаче.

1. Выбор мутации.

Генотипы подвергаются каким-либо мутациям, то есть каким-то образом изменяются их составляющие.

1. Выбор механизма скрещивания (кроссинговера).

На данном этапе происходит создание новых генотипов из частей старых.

1. Выбор функции оценки(фитнесс-функции)

Такая функция позволяет оценивать генотипы и отбирать из них лучшие.

Мною были выбраны следующие реализации данных этапов:

1. В качестве генотипа, описывающего конфигурацию решения, используется бинарная последовательность длины N, в которой каждое i-е значение является индикатором вхождения i элемента в рюкзак. Такой ген однозначно описывает состояние рюкзака и позволяет легко проводить мутацию или скрещивание, но требует дополнительной проверки на верность(валидность) при создании или изменении.
2. Было выбрано два классических алгоритма мутации. Одноточечная мутация - случайным образом выбирается точка, для которой производится инверсия значения в конфигурации, в случае, если полученная конфигурация невалидна - изменение отменяется, и точка выбирается повторно. Половинная мутация - случайным образом выбирается точка в конфигурации, для всех значений до этой точки производится побитовая инверсия, после чего конфигурация валидируется.
3. Были выбраны три классических алгоритма скрещивания. Точечное скрещивание, когда случайным образом выбирается точка в конфигурации, и все значения слева от неё берутся из одной родительской конфигурации, а в точке и справа - из другой родительской конфигурации. Двухточечное скрещивание обеспечивает скрещивание по случайно выбранному интервалу, все значения внутри которого берутся из одной конфигурации, а вне его - из другой. Побитовое скрещивание - значения берутся попеременно из двух конфигураций - 1,3,5... из первой, 2, 4, 6… из второй
4. Для оценки генотипа вычисляется сумма стоимостей всех предметов, входящих в описанный генотипом рюкзак. Опционально такая сумма может быть нормирована к интервалу, где 1 – стоимость конфигурации, в которую помещены все предметы.

Для избегания использования неверных генотипов используется простой алгоритм. Сначала вычисляется сумма размеров предметов, попавших в рюкзак по каждому из размеров. Затем, если эта сумма превышает какое-либо ограничение, то из генотипа в случайно порядке удаляются предметы, пока он не будет описывать верную конфигурацию.

# **Описание работы программы**

В начале работы алгоритма задаются исходные данные: количество предметов в рюкзаке, количество измерений, ограничения по вместимости, стоимости предметов и их размеры, также здесь задаётся количество конфигураций в пуле алгоритма, процент мутаций и выбранные алгоритмы мутации и скрещивания. Было реализовано два метода ввода исходных данных: псевдослучайная генерация с фиксированными параметрами без условия остановки алгоритма и генерация путем чтения из файла с введением условия остановки – достижения оптимальной конфигурации. Первый из способов использовался для проверки работоспособности программы, второй – для тестирования алгоритма. В качестве тестовых данных использовался набор проблем из публикации Карла Петерсена.

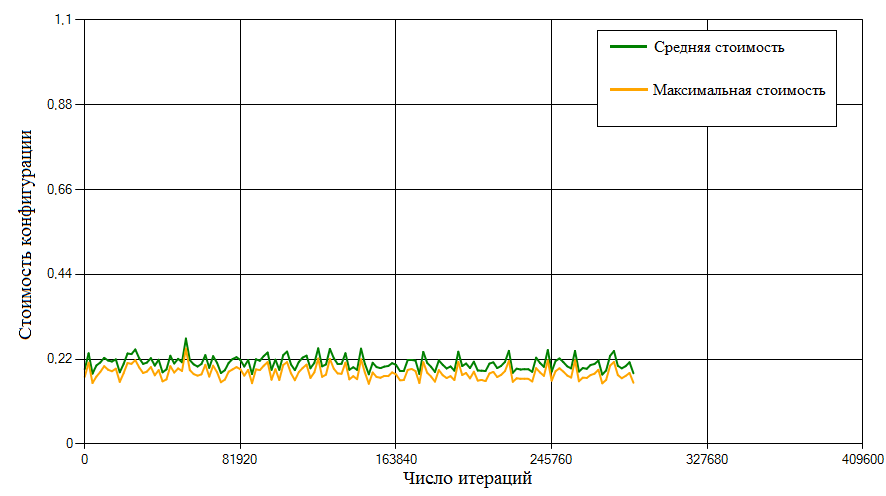
Для запуска алгоритма создаётся начальное приближение следующим образом: используется конфигурация, в которой добавлены все предметы, из которой случайным образом “выкидываются” значения вплоть до того момента, когда она станет верной. В случае, когда такая конфигурация описывает рюкзак со всеми предметами или рюкзак без предметов, алгоритм останавливается. В противном случае пул конфигураций заполняется мутациями первого приближения.

После этого в цикле начинает выполняться метод эволюции.

Для слежения были реализованы графический и консольный интерфейсы.

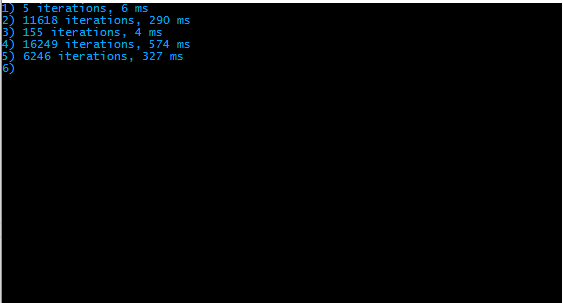
Пример работы графического интерфейса можно увидеть на рисунке 1.

Рисунок 1. Пример работы алгоритма со случайными вводными данными



Консольный интерфейс использовался только для тестирования по готовым данным, он предоставляет информацию о пройденных тестах, затраченном на них времени и количестве итераций. Пример работы консольного интерфейса можно увидеть на рисунке 2

Рисунок 2.Пример работы алгоритма на заданном наборе тестов



# **Результаты работы**

После тестирования были получены следующие результаты работы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Точечная мутация | | | | | | |
|  | Одноточечное скрещивание | | Двухточечное скрещивание | | Побитовое скрещивание | |
| № теста | Итерации | Время, мс | Итерации | Время, мс | Итерации | Время, мс |
| 1 | 6 | 8 | 21 | 9 | 3 | 9 |
| 2 | 114340 | 2694 | 70930 | 1697 | 1826334 | 33017 |
| 3 | 712 | 18 | 901 | 22 | 1455 | 49 |
| 4 | 4022 | 92 | 2092 | 67 | 7869 | 219 |
| 5 | 3049 | 97 | 2672 | 95 | 2198 | 74 |
| 6 | 400974 | 13481 | 23261 | 818 | 289379 | 10013 |
| 7 | 9343432 | 396644 | 4770456 | 199461 | 348826 | 13940 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Половинная мутация | | | | | | |
|  | Одноточечное скрещивание | | Двухточечное скрещивание | | Побитовое скрещивание | |
| № теста | Итерации | Время, мс | Итерации | Время, мс | Итерации | Время, мс |
| 1 | 3 | 6 | 2 | 10 | 10 | 10 |
| 2 | 10 | 2 | 16 | 1 | 23 | 1 |
| 3 | 58 | 2 | 356 | 19 | 1409 | 52 |
| 4 | 9538 | 560 | 7161 | 422 | 246001 | 12189 |
| 5 | - | - | - | - | - | - |
| 6 | - | - | - | - | - | - |
| 7 | - | - | - | - | - | - |

Входные данные тестов

|  |  |  |
| --- | --- | --- |
| № теста | Число предметов | Число измерений |
| 1 | 6 | 10 |
| 2 | 10 | 10 |
| 3 | 15 | 10 |
| 4 | 20 | 10 |
| 5 | 28 | 10 |
| 6 | 39 | 5 |
| 7 | 50 | 5 |

Можно заметить, что в зависимости от выбора мутации и скрещивания алгоритм может как улучшать свои результаты в отдельных тестах(например, в тесте 2, где был получен существенный выигрыш по времени при использовании половинной мутации), так и ухудшать(тесты 5-7, на которых алгоритм при использовании половинной мутации не находил оптимальное решение в течение получаса). Также можно заметить, что двухточечное скрещивание оказалось существенно эффективнее иных на различных наборах тестов.

# **Вывод**

В данной работе было изучено применение генетического алгоритма к задаче о многомерном рюкзаке. Полученные результаты показывают, что генетический алгоритм является эффективным в решении данной задачи. Эффективность алгоритма существенно зависит от выбранных схемы мутаций и скрещивания.

В дальнейшем данный алгоритм может быть улучшен с использованием средств параллельного программирования и добавлением дополнительных методов оптимизации, таких как поиск Табу.

# **Список литературы**

1. G. B. Mathews On the partition of numbers — 1897. — P. 486-490.
2. C.C.Petersen "Computational experience  
   with variants of the Balas algorithm applied to the selection  
   of R&D projects" Management Science 13(9) (1967) 736-750.
3. Mitchell, Melanie (1996). An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press.
4. Искусственный интеллект и машенное обучение (видеокурс) - https://ulearn.azurewebsites.net/Course/AIML/

# **Приложение**

Файл GeneticalAlgorithm.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace GAMultidimKnapsack

{

class GeneticalAlgorithm

{

private static int itemsAmount, dimensions;

private static double[,] itemsSet;

private static double[] restrictions;

private double[] itemsCosts;

private int configsInPoolAmount;

private int bestConfigsAmount;

private KnapsackConfig[] configsPool;

private KnapsackConfig[] bestConfigs;

private double maximalKnapsackCost;

private Crossover activeCrossover;

private Mutation activeMutation;

private static Random rand;

private double mutationPercentage;

public GeneticalAlgorithm(int itemsAm, int dim, double[] rest, double[] costs, double[,] myItemsSet, int confAm, Crossover myCrs, Mutation myMt, double mutationPercentage)

{

itemsAmount = itemsAm;

restrictions = rest;

dimensions = dim;

itemsSet = new double[itemsAm, dim];

rand = new Random();

itemsSet = myItemsSet;

itemsCosts = costs;

configsInPoolAmount = confAm;

activeCrossover = myCrs;

activeMutation = myMt;

bestConfigsAmount = 10;

int[] emptyConfig = (new int[itemsAmount]).Select(x => 0).ToArray();

bestConfigs = (new KnapsackConfig[bestConfigsAmount]).Select(x => new KnapsackConfig(emptyConfig)).ToArray();//HACK

configsPool = new KnapsackConfig[configsInPoolAmount];

this.mutationPercentage = mutationPercentage;

maximalKnapsackCost = itemsCosts.Sum();

StartCycling();

}

private void StartCycling()

{

try

{

configsPool[0] = FirstApproachGenerate();

int active = 0, passive = 0;

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

{

if (configsPool[0].isValueActive(i))

active++;

else passive++;

}

if (active == itemsAmount || passive == itemsAmount)

return;

for (int i = 1; i < configsInPoolAmount; i++)

{

configsPool[i] = activeMutation(configsPool[0], rand);

}

}

catch (Exception ex)

{

MessageBox.Show("Bugs in initialization", "ERROR", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

}

public void MakeIteration()

{

if (GetKnapsackCost(configsPool[0]) == maximalKnapsackCost) return;

List<int> positions = new List<int>();

while (positions.Count < mutationPercentage \* configsInPoolAmount)

{

positions.Add(rand.Next(configsInPoolAmount));

positions.Distinct();

}

foreach (var pos in positions)

{

configsPool[pos] = activeMutation(configsPool[pos], rand);

}

KnapsackConfig[] CrossoverPool = new KnapsackConfig[configsInPoolAmount \* 2 - 2];//not very well, if i want to customize Crossover ,but works

for (var j = 0; j < (configsInPoolAmount - 1); j++)

{

CrossoverPool[j] = activeCrossover(configsPool[j], configsPool[j + 1], true);

CrossoverPool[(CrossoverPool.Length - 1) - j] = activeCrossover(configsPool[j], configsPool[j + 1], false);

}

var tempConfigs = CrossoverPool.OrderByDescending(config => GetKnapsackCost(config))

.Distinct()

.Take(Convert.ToInt32(configsInPoolAmount))

.ToArray();

configsPool = tempConfigs;

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

{

if (GetKnapsackCost(bestConfigs[i]) < GetKnapsackCost(configsPool[0]))

{

for (int j = i; i < bestConfigsAmount && j < configsInPoolAmount; j++, i++)

bestConfigs[i] = configsPool[j];

break;

}

}

}

private KnapsackConfig FirstApproachGenerate()

{

KnapsackConfig result = new KnapsackConfig(itemsAmount);

for (var i = 0; i < itemsAmount; i++)

{

result.setValueToActive(i);

}

Random rand = new Random();

while (!IsValid(result))

{

int positionNumber = rand.Next(itemsAmount);

while (!result.isValueActive(positionNumber))

{

positionNumber = rand.Next(itemsAmount);

}

result.setValueToPassive(positionNumber);

}

return result;

}

public delegate KnapsackConfig Crossover(KnapsackConfig sack1, KnapsackConfig sack2, bool isLeft);

public static KnapsackConfig FixedSinglePointCrossover(KnapsackConfig sack1, KnapsackConfig sack2, bool isLeft)

{

int[] crossItems = new int[itemsAmount];

if (isLeft)

{

for (var i = 0; i < itemsAmount/2; i++)

crossItems[i] = sack2.valueAt(i);

for (var i=itemsAmount/2;i<itemsAmount;i++)

crossItems[i] = sack1.valueAt(i);

}

else

{

for (var i = 0; i < itemsAmount / 2; i++)

crossItems[i] = sack1.valueAt(i);

for (var i = itemsAmount / 2; i < itemsAmount; i++)

crossItems[i] = sack2.valueAt(i);

}

KnapsackConfig CrossoverResult = new KnapsackConfig(crossItems);

if (!IsValid(CrossoverResult))

CrossoverResult = MakeValid(CrossoverResult);

return CrossoverResult;

}

public static KnapsackConfig SinglePointCrossover(KnapsackConfig sack1, KnapsackConfig sack2, bool isLeft)

{

int[] crossItems = new int[itemsAmount];

int CrossoverPoint = rand.Next(itemsAmount);

if (isLeft)

{

for (var i = 0; i < CrossoverPoint; i++)

crossItems[i] = sack2.valueAt(i);

for (var i = CrossoverPoint; i < itemsAmount; i++)

crossItems[i] = sack1.valueAt(i);

}

else

{

{

for (var i = 0; i < CrossoverPoint; i++)

crossItems[i] = sack1.valueAt(i);

for (var i = CrossoverPoint; i < itemsAmount; i++)

crossItems[i] = sack2.valueAt(i);

}

}

KnapsackConfig CrossoverResult = new KnapsackConfig(crossItems);

if (!IsValid(CrossoverResult))

CrossoverResult = MakeValid(CrossoverResult);

return CrossoverResult;

}

public static KnapsackConfig BitByBitCrossover(KnapsackConfig sack1, KnapsackConfig sack2, bool isLeft)

{

int[] crossItems = new int[itemsAmount];

if (isLeft)

{

for (var i = 0; i < itemsAmount; i++)

{

if (i % 2 == 0)

crossItems[i] = sack2.valueAt(i);

else

crossItems[i] = sack1.valueAt(i);

}

}

else

{

for (var i = 0; i < itemsAmount; i++)

{

if (i % 2 == 0)

crossItems[i] = sack1.valueAt(i);

else

crossItems[i] = sack2.valueAt(i);

}

}

KnapsackConfig CrossoverResult = new KnapsackConfig(crossItems);

if (!IsValid(CrossoverResult))

CrossoverResult = MakeValid(CrossoverResult);

return CrossoverResult;

}

public static KnapsackConfig TwoPointCrossover(KnapsackConfig sack1, KnapsackConfig sack2, bool isLeft)

{

int firstPoint = rand.Next(itemsAmount-1), secondPoint = rand.Next(firstPoint + 1,itemsAmount);

int[] crossItems = new int[itemsAmount];

if (isLeft)

{

for (var i = 0; i < firstPoint; i++)

crossItems[i] = sack1.valueAt(i);

for (var i = firstPoint; i < secondPoint; i++)

crossItems[i] = sack2.valueAt(i);

for (var i = secondPoint; i < itemsAmount; i++)

crossItems[i] = sack1.valueAt(i);

}

else

{

for (var i = 0; i < firstPoint; i++)

crossItems[i] = sack2.valueAt(i);

for (var i = firstPoint; i < secondPoint; i++)

crossItems[i] = sack1.valueAt(i);

for (var i = secondPoint; i < itemsAmount; i++)

crossItems[i] = sack2.valueAt(i);

}

KnapsackConfig sack = new KnapsackConfig(crossItems);

if (!IsValid(sack))

return (MakeValid(sack));

return sack;

}

public delegate KnapsackConfig Mutation(KnapsackConfig sack, Random rand);

public static KnapsackConfig SinglePointMutation(KnapsackConfig sack, Random rand)

{

KnapsackConfig mutatedSack = new KnapsackConfig(sack);//copy constructor

int mutationPosition = rand.Next(itemsAmount);

var count = 0;

while (mutatedSack.Equals(sack) && count < 1000000)//TODO - not mutate empty sack

{

mutatedSack.swapValue(mutationPosition);

if (!IsValid(mutatedSack))

{

mutatedSack.swapValue(mutationPosition);

mutationPosition = rand.Next(itemsAmount);

count++;

}

}

if (count == 1000000)

{

return MakeValid(mutatedSack);

}

return mutatedSack;

}

public static KnapsackConfig MutateHalf(KnapsackConfig sack, Random rand)

{

KnapsackConfig mutatedSack = new KnapsackConfig(sack);

int mutationPosition = rand.Next(itemsAmount);

if (rand.Next() % 2 == 0)

{

for (var i = 0; i < itemsAmount / 2; i++)

{

mutatedSack.swapValue(i);

}

}

else

{

for (var i = itemsAmount / 2; i < itemsAmount; i++)

{

mutatedSack.swapValue(i);

}

}

if (!IsValid(mutatedSack))

return (MakeValid(mutatedSack));

return (mutatedSack);

}

private static bool IsValid(KnapsackConfig config)

{

double[] summ = new double[dimensions];

for (var i = 0; i < itemsAmount; i++)

{

if (config.isValueActive(i))

{

for (var j = 0; j < dimensions; j++)

{

summ[j] += itemsSet[i, j];

if (summ[j] > restrictions[j]) return false;

}

}

}

return true;

}

private double GetKnapsackCost(KnapsackConfig sack)

{

double count = 0;

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

if (sack.isValueActive(i))

count += itemsCosts[i];

return count;

}

private static KnapsackConfig MakeValid(KnapsackConfig sack)

{

var position = rand.Next(itemsAmount);

while(!IsValid(sack))

{

sack.setValueToPassive(position);

position = rand.Next(itemsAmount);

}

return sack;

}

public double GetNormalizedMaximalKnapsackCost()

{

return GetAbsoluteMaximalKnapsackCost() / maximalKnapsackCost;

}

public double GetNormaizedAveragePoolCost()

{

return GetAbsoluteAverageKnapsackCost() / maximalKnapsackCost;

}

public double GetAbsoluteMaximalKnapsackCost()

{

return GetKnapsackCost(configsPool[0]);

}

public double GetAbsoluteAverageKnapsackCost()

{

if (GetKnapsackCost(configsPool[0]) == maximalKnapsackCost) return GetKnapsackCost(configsPool[0]);

double averagePoolCost = 0;

foreach (var config in configsPool)

{

averagePoolCost += GetKnapsackCost(config);

}

averagePoolCost /= configsInPoolAmount;

return averagePoolCost;

}

}

}

Файл KnapsackConfig.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GAMultidimKnapsack

{

class KnapsackConfig

{

private int[] CurrentConfiguration;

public KnapsackConfig(int elementsAmount)

{

CurrentConfiguration = new int[elementsAmount];

}

public KnapsackConfig(int[] initConfig)

{

CurrentConfiguration = initConfig;

}

public KnapsackConfig(KnapsackConfig conf)

{

try

{

this.CurrentConfiguration = new int[conf.Length()];

for (int i = 0; i < conf.Length(); i++)

this.CurrentConfiguration[i] = conf.valueAt(i);

}

catch (NullReferenceException ex)

{

MessageBox.Show("Empty configuration", "Null Reference Exception", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return;

}

}

public void setValueToActive(int position)

{

CurrentConfiguration[position] = 1;

}

public void setValueToPassive(int position)

{

CurrentConfiguration[position] = -1;

}

public void swapValue(int position)

{

CurrentConfiguration[position] = -CurrentConfiguration[position];

}

public bool isValueActive(int position)

{

return (CurrentConfiguration[position] > 0);

}

public int valueAt(int position)

{

return (CurrentConfiguration[position]);

}

public int Length()

{

try

{

return CurrentConfiguration.Length;

}

catch (NullReferenceException ex)

{

MessageBox.Show("Empty configuration", "Null Reference Exception", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

return 0;

}

}

public bool Equals(KnapsackConfig sack)

{

if (this.Length() != sack.Length()) return false;

for (int i = 0; i < this.Length(); i++)

{

if (this.valueAt(i) != sack.valueAt(i))

return false;

}

return true;

}

}

}

Файл ConsoleProgram.cs

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Linq;

using System.Threading;

using static System.Net.WebRequestMethods;

namespace GAMultidimKnapsack

{

class Program

{

private static List<long>[] averageTime=new List<long>[7];

private static List<long>[] averageIterations = new List<long>[7];

static void Algorithm(int itemsAmount, int dimensions, double maxCost, double[] restrictions, double[] costs, double[,] itemsSet, int testNumber)

{

int ConfigsAmount = 8;

double mutationPercent = 0.75;

GeneticalAlgorithm ga = new GeneticalAlgorithm(itemsAmount, dimensions, restrictions, costs, itemsSet, ConfigsAmount, GeneticalAlgorithm.BitByBitCrossover, GeneticalAlgorithm.MutateHalf, mutationPercent);

int iterationNumber = 1;

var controlWatch = new Stopwatch();

controlWatch.Start();

while (ga.GetAbsoluteMaximalKnapsackCost() != maxCost)

{

ga.MakeIteration();

iterationNumber++;

}

controlWatch.Stop();

using (StreamWriter file1 = new StreamWriter(@"C:\Users\black\_000\Documents\visual studio 2015\Projects\ConsoleKnapsack\ConsoleKnapsack\out.txt", true))

file1.WriteLine(iterationNumber + " iterations, " + controlWatch.ElapsedMilliseconds + " ms");

//averageTime[testNumber].Add(controlWatch.ElapsedMilliseconds);

//averageIterations[testNumber].Add(iterationNumber);

}

static void ProcessTestSet(string file)

{

using (StreamReader sr = new StreamReader(file))

{

int experimentsAmount = Convert.ToInt32(sr.ReadLine());

for (int experiment = 0; experiment < experimentsAmount; experiment++)

{

string[] initializationSequence;

string firstString = sr.ReadLine();

if (firstString.Trim() == "")

initializationSequence = sr.ReadLine().Split(" ".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);

else initializationSequence = firstString.Split(" ".ToCharArray(), StringSplitOptions.RemoveEmptyEntries); ;

int itemsAmount = Convert.ToInt32(initializationSequence[0]),

dimensions = Convert.ToInt32(initializationSequence[1]);

double maxCost = Convert.ToDouble(initializationSequence[2]);

List<double> tempCosts = new List<double>();

while (tempCosts.Count() != itemsAmount)

tempCosts.AddRange(sr

.ReadLine()

.Split(" ".ToCharArray(), StringSplitOptions.RemoveEmptyEntries)

.Select(x => Convert.ToDouble(x))

.ToList());

double[] costs = tempCosts.ToArray();

double[,] itemsSet = new double[itemsAmount, dimensions];

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

{

int itemsReaden = 0;

while (itemsReaden != itemsAmount)

{

double[] currentString = sr.ReadLine()

.Split(" ".ToCharArray(), StringSplitOptions.RemoveEmptyEntries).

Select(x => Convert.ToDouble(x)).

ToArray();

for (int j = itemsReaden, k = 0; j < currentString.Count() + itemsReaden; j++, k++)

itemsSet[j, i] = currentString[k];

itemsReaden += currentString.Count();

}

}

List<double> tempRestrictions = new List<double>();

while (tempRestrictions.Count() != dimensions)

tempRestrictions.AddRange(sr

.ReadLine()

.Split(" ".ToCharArray(), StringSplitOptions.RemoveEmptyEntries)

.Select(x => Convert.ToDouble(x))

.ToList());

double[] restrictions = tempRestrictions.ToArray();

//using (StreamWriter file1 = new StreamWriter(@"C:\Users\black\_000\Documents\visual studio 2015\Projects\ConsoleKnapsack\ConsoleKnapsack\out.txt", true))

//{

// file1.Write(experiment + 1 + ") ");

//}

Algorithm(itemsAmount, dimensions, maxCost, restrictions, costs, itemsSet, experiment);

// Thread.Sleep(3000);

}

}

}

static void Main(string[] args)

{

ProcessTestSet(@"C:\Users\black\_000\Source\Repos\GeneticKnapsack\GAMultidimKnapsack\1.txt");

//for (var i = 0; i < 7; i++)

//{

// resultsTime[i] = averageTime[i].Sum() / testsAmount;

// resultsIterations[i] = averageIterations[i].Sum() / testsAmount;

//}

//using (StreamWriter file = new StreamWriter(@"C:\Users\black\_000\Documents\visual studio 2015\Projects\ConsoleKnapsack\ConsoleKnapsack\out.txt", true))

//{

// file.WriteLine("Iterations:");

// for (var i = 0; i < resultsTime.Length; i++)

// file.WriteLine(resultsTime[i]);

// file.WriteLine("Time:");

// for (var i = 0; i < resultsTime.Length; i++)

// file.WriteLine(resultsIterations[i]);

// file.Close();

//}

}

}

}

Файл SetPartition.cs

//using AIRLab.GeneticAlgorithms;

using Common;

using GAMultidimKnapsack;

using System;

using System.Collections.Concurrent;

using System.Collections.Generic;

using System.Diagnostics;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace SetPartition

{

internal static class ConcurrentQueueExtensions

{

public static void Clear<T>(this ConcurrentQueue<T> queue)

{

T item;

while (queue.TryDequeue(out item))

{

// do nothing

}

}

}

static class SetPartition

{

static Random rand = new Random(42);

static Form form;

static ConcurrentQueue<double> averageValuations = new ConcurrentQueue<double>();

static ConcurrentQueue<double> maxValuations = new ConcurrentQueue<double>();

static ConcurrentQueue<double> ages = new ConcurrentQueue<double>();

static HistoryChart valuationsChart = new HistoryChart();

static HistoryChart agesChart = new HistoryChart();

static void Algorithm(int itemsAmount, int dimensions, double maxCost, double[] restrictions, double[] costs, double[,] itemsSet)

{

int ConfigsAmount = 8;

double mutationPercent = 0.75;

GeneticalAlgorithm ga = new GeneticalAlgorithm(itemsAmount, dimensions, restrictions, costs, itemsSet, ConfigsAmount, GeneticalAlgorithm.TwoPointCrossover, GeneticalAlgorithm.SinglePointMutation, mutationPercent);

int iterationNumber = 0;

while (ga.GetAbsoluteMaximalKnapsackCost() != maxCost)

{

var watch = new Stopwatch();

watch.Start();

while (watch.ElapsedMilliseconds < 1)

{

ga.MakeIteration();

iterationNumber++;

averageValuations.Enqueue(ga.GetNormaizedAveragePoolCost());

maxValuations.Enqueue(ga.GetNormalizedMaximalKnapsackCost());

ages.Enqueue(rand.NextDouble() \* 100);

}

watch.Stop();

if (!form.IsDisposed)

form.BeginInvoke(new Action(UpdateCharts));

}

}

static void TestAlgorithm()//Proof of concept

{

int itemsAmount = 500, dimensions = 6;

double[] restrictions = new double[] { 100, 600, 1200, 2400, 500, 2000 }, costs = new double[itemsAmount];

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

costs[i] = rand.NextDouble() \* 30;

double[,] itemsSet = new double[itemsAmount, dimensions];

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

for (int j = 0; j < dimensions; j++)

itemsSet[i, j] = rand.NextDouble() \* 50;

int ConfigsAmount = 8;

GeneticalAlgorithm ga = new GeneticalAlgorithm(itemsAmount, dimensions, restrictions, costs, itemsSet, ConfigsAmount, GeneticalAlgorithm.FixedSinglePointCrossover, GeneticalAlgorithm.SinglePointMutation, 0.75);

int iterationNumber = 0;

while (true)

{

var watch = new Stopwatch();

watch.Start();

while (watch.ElapsedMilliseconds < 200)

{

ga.MakeIteration();

iterationNumber++;

averageValuations.Enqueue(ga.GetNormaizedAveragePoolCost());

maxValuations.Enqueue(ga.GetNormalizedMaximalKnapsackCost());

}

watch.Stop();

if (!form.IsDisposed)

form.BeginInvoke(new Action(UpdateCharts));

}

}

static void ProcessTestSet(string file)

{

using (StreamReader sr = new StreamReader(file))

{

int experimentsAmount = Convert.ToInt32(sr.ReadLine());

for (int experiment = 0; experiment < experimentsAmount; experiment++)

{

string[] initializationSequence;

string firstString = sr.ReadLine();

if (firstString.Trim() == "")

initializationSequence = sr.ReadLine().Split(" ".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);

else initializationSequence = firstString.Split(" ".ToCharArray(), StringSplitOptions.RemoveEmptyEntries); ;

int itemsAmount = Convert.ToInt32(initializationSequence[0]),

dimensions = Convert.ToInt32(initializationSequence[1]);

double maxCost = Convert.ToDouble(initializationSequence[2]);

List<double> tempCosts = new List<double>();

while (tempCosts.Count() != itemsAmount)

tempCosts.AddRange(sr

.ReadLine()

.Split(" ".ToCharArray(), StringSplitOptions.RemoveEmptyEntries)

.Select(x => Convert.ToDouble(x))

.ToList());

double[] costs = tempCosts.ToArray();

double[,] itemsSet = new double[itemsAmount, dimensions];

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

{

int itemsReaden = 0;

while (itemsReaden != itemsAmount)

{

double[] currentString = sr.ReadLine()

.Split(" ".ToCharArray(), StringSplitOptions.RemoveEmptyEntries).

Select(x => Convert.ToDouble(x)).

ToArray();

for (int j = itemsReaden, k = 0; j < currentString.Count() + itemsReaden; j++, k++)

itemsSet[j, i] = currentString[k];

itemsReaden += currentString.Count();

}

}

List<double> tempRestrictions = new List<double>();

while (tempRestrictions.Count() != dimensions)

tempRestrictions.AddRange(sr

.ReadLine()

.Split(" ".ToCharArray(), StringSplitOptions.RemoveEmptyEntries)

.Select(x => Convert.ToDouble(x))

.ToList());

double[] restrictions = tempRestrictions.ToArray();

Algorithm(itemsAmount, dimensions, maxCost, restrictions, costs, itemsSet);

Thread.Sleep(3000);

maxValuations.Enqueue(0);

averageValuations.Enqueue(0);

}

}

}

static void UpdateCharts()

{

valuationsChart.AddRange(maxValuations, averageValuations);

maxValuations.Clear();

averageValuations.Clear();

}

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

form = new Form();

var table = new TableLayoutPanel() { Dock = DockStyle.Fill, RowCount = 2, ColumnCount = 1 };

valuationsChart = new HistoryChart

{

Lines =

{

new HistoryChartValueLine { DataFunction = { Color = Color.Green, BorderWidth=2 }},

new HistoryChartValueLine { DataFunction = { Color = Color.Orange, BorderWidth=2 }},

},

Max = 1,

Dock = DockStyle.Fill

};

table.Controls.Add(valuationsChart, 0, 0);

table.RowStyles.Add(new RowStyle { SizeType = SizeType.Percent, Height = 50 });

form.Controls.Add(table);

new Thread(TestAlgorithm) { IsBackground = true }.Start();

// new Thread(() => ProcessTestSet(@"C:\Users\black\_000\Source\Repos\GeneticKnapsack\GAMultidimKnapsack\1.txt")) { IsBackground = true }.Start();

Application.Run(form);

}

}

}

Файл HistoryChart.cs

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms.DataVisualization.Charting;

namespace Common

{

public class HistoryChartValueLine

{

public int DotsCount = 200;

public int AverageCount { get; private set; }

public int ShrinkCount { get; private set; }

List<double> values = new List<double>();

Queue<double> temp = new Queue<double>();

public HistoryChartValueLine()

{

DotsCount = 200;

AverageCount = 1;

ShrinkCount = 0;

DataFunction = new Series()

{

ChartType = SeriesChartType.FastLine

};

}

public void Shrink()

{

for (int j = 0; j < values.Count - 1; j++)

{

values[j] = (values[j] + values[j + 1]) / 2;

values.RemoveAt(j + 1);

}

AverageCount \*= 2;

ShrinkCount++;

}

public void Take(IEnumerable<double> data)

{

foreach (var e in data) temp.Enqueue(e);

}

public int GetTotalShrinksAfterPull()

{

var totalPoints = (temp.Count / AverageCount) + values.Count;

int result = 0;

while (totalPoints > DotsCount) { totalPoints /= 2; result++; }

return result + ShrinkCount;

}

public void Pull(double maximum)

{

while (temp.Count > AverageCount)

{

double sum = 0;

for (int i = 0; i < AverageCount; i++) sum += temp.Dequeue();

sum /= AverageCount;

values.Add(sum);

}

DataFunction.Points.Clear();

for (int i = 0; i < values.Count; i++)

DataFunction.Points.Add(new DataPoint(AverageCount \* i, Math.Min(values[i], maximum)));

}

public Series DataFunction { get; set; }

}

public class HistoryChart : Chart

{

public int DotsCount = 200;

public List<HistoryChartValueLine> Lines { get; private set; }

double max = 1;

public double Max

{

get { return max; }

set

{

area.AxisY.Maximum = value\*1.1;

max = value\*1.1;

}

}

ChartArea area;

int averageCount = 1;

bool initialized = false;

public HistoryChart()

{

Lines = new List<HistoryChartValueLine>();

area = new ChartArea()

{

AxisY = { Minimum = 0 },

AxisX = { Minimum = 0 }

};

ChartAreas.Add(area);

}

public void AddRange(params IEnumerable<double>[] ranges)

{

if (!initialized)

{

foreach (var s in Lines)

{

Series.Add(s.DataFunction);

s.DotsCount = DotsCount;

}

initialized = true;

}

if (Lines.Count != ranges.Length) throw new Exception();

for (int i = 0; i < ranges.Length; i++)

Lines[i].Take(ranges[i]);

var maxShrink = Lines.Select(z => z.GetTotalShrinksAfterPull()).Max();

for (int i = 0; i < Lines.Count; i++)

{

while (Lines[i].ShrinkCount < maxShrink) Lines[i].Shrink();

Lines[i].Pull(max);

}

area.AxisX.Maximum = Math.Pow(2,maxShrink)\*DotsCount;

}

}

}