МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра ИАНИ**

Направление подготовки: «Прикладная информатика»

Профиль подготовки: «Прикладная информатика в области обработки данных»

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

на тему:

**«Сравнительный анализ подходов к решению задачи об оптимальном распределении инвестиций»**

**Выполнил(а):** студент группы 381407в

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Иванеженков Д. В.

Подпись

**Научный руководитель:**

Старший преподаватель, кандидат технических наук

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Неймарк Е. А.

Подпись

Нижний Новгород  
2018

Оглавление

[Введение 3](#_Toc535490553)

[Формальная постановка задачи 4](#_Toc535490554)

[Обзор методов решения 5](#_Toc535490555)

[Алгоритм полного перебора 5](#_Toc535490556)

[Метод ветвей и границ 5](#_Toc535490557)

[Метод динамического программирования 5](#_Toc535490558)

[Жадные алгоритмы 6](#_Toc535490559)

[Генетические алгоритмы 6](#_Toc535490560)

[Алгоритмическое описание выбранных алгоритмов 8](#_Toc535490561)

[Метод динамического программирования 8](#_Toc535490562)

[Генетические алгоритмы 8](#_Toc535490563)

[Жадный алгоритм 10](#_Toc535490564)

[Эксперимент 12](#_Toc535490565)

[Заключение 14](#_Toc535490566)

[Литература 16](#_Toc535490567)

[Приложение 17](#_Toc535490568)

## Введение

Задача о ранце является широко известной задачей дискретной оптимизации, что обусловлено большим количеством её приложений.

Задача о ранце и её различные вариации находят широкое практическое применение в областях планирования и управления производственными и транспортными системами, прикладной математике, криптографии, и т.д.

Формулируется задача таким образом:

Пусть есть N разных предметов, каждый предмет имеет вес  и полезность , так же имеется максимальный вес , который можно положить в рюкзак. Требуется собрать такой набор предметов , чтобы полезность их была наибольшей, а суммарный вес не превышал .

В данной работе рассматривается задача оптимального распределения инвестиций, что является частным случаем задачи о ранце, а именно ранец с мультивыбором (англ. *Multiple-choice Knapsack Problem*) в приложении к экономике. В этой вариации предметы разделены на группы, и из каждой группы требуется выбрать только один предмет.

Задача является NP – трудной, т.е. на данный момент не существует алгоритмов, способных найти оптимальное решение за полиномиальное время. Отсюда вытекает проблема выбора метода для поиска решения: использовать точный алгоритм, дающий оптимальное решение, но не работающий за приемлемое время для больших объемов входных данных; либо использовать быстрый алгоритм, не гарантирующий нахождения оптимума.

Цель данной работы – провести сравнительный анализ методов решения задачи оптимального распределения инвестиций, выявить диапазоны наборов входных данных, на которых оправданно применение этих методов.

Для достижения поставленной цели необходимо решение следующих задач:

* Описание методов
* Реализация конкретных алгоритмов
* Проведение экспериментов с замерами времени работы и оценкой приближенности решения к оптимуму реализованных алгоритмов на различных наборах данных
* Сравнение и анализ собранных результатов
* Формирование заключения на основе полученных данных

## Формальная постановка задачи

Требуется вложить *T* имеющихся средств в *m* предприятий. Для каждого предприятия предусмотрены n сумм вложений. Прибыль , для каждого предприятия определяется в зависимости от количества вложенных средств , . Необходимо распределить средства так, чтобы прибыль со всех предприятий была максимальной. При этом, в одно предприятие можно вложиться только один раз.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | *…* |  | *…* |  |
|  |  |  | *…* |  | *…* |  |
|  |  |  | *…* |  | *…* |  |
| *…* | *…* | *…* | *…* | *…* | *…* | *…* |
|  |  |  | *…* |  | *…* |  |
| *…* | *…* | *…* | *…* | *…* | *…* | *…* |
|  |  |  | *…* |  | *…* |  |

Здесь – прибыль *j*-го предприятия при вложении в него средств.

Математическая модель задачи:

Определить вектор , где , ∀ , где – количество средств, вложенных в *j*-е предприятие; удовлетворяющий условиям:

и обеспечивающий максимум целевой функции

## Обзор методов решения

# **Алгоритм полного перебора**

Полный перебор – метод решения, относящийся к классу методов поиска путем исчерпывания возможных вариантов. Любая задача из класса NP может быть решена полным перебором, но сложность вычислений зависит от количества всех решений задачи. В данном случае, для каждой группы из предметов (для каждого предприятия) существует n вариантов выбора предмета (вложения определенной суммы в предприятие). Тогда, если всего предприятий , перебор всех возможных вариантов имеет временную сложность *O*(), что позволяет использовать его лишь для небольшого количества предметов. С ростом числа предметов задача становится неразрешимой данным методом за приемлемое время.

# Метод ветвей и границ

Метод ветвей и границ – общий алгоритмический метод нахождения оптимальных решений различных задач оптимизации. Является вариацией метода полного перебора; отличается исключением заведомо неоптимальных ветвей дерева полного перебора на основе оценок верхних и нижних границ ожидаемого решения для каждой ветви.

В начале берется какое-либо допустимое решение, называемое «рекордом», для него считается значение целевой функции. Далее, в процессе построения дерева перебора, для каждого узла считается верхняя оценка возможного решения. Если эта оценка не больше значения целевой функции у «рекорда», то в рассматриваемом подмножестве не содержится решения лучше «рекорда», и оно может быть отброшено. Если значение целевой функции на очередном решении больше рекордного, то происходит смена рекорда. Алгоритм заканчивает свою работу, когда будут просмотрены все возможные подмножества. Метод ветвей и границ в среднем работает быстрее алгоритма полного перебора, но всегда можно подобрать такие входные данные, для которых оценка по времени работы этих алгоритмов будет совпадать.

# Метод динамического программирования

Динамическое программирование – подход, позволяющий решать задачи оптимизации, которые могут быть сформулированы как задачи многошагового оптимального управления некоторой системой. Метод основан на уравнении Беллмана и являет собой сведение сложной оптимизационной задачи к упорядоченной последовательности более простых задач (задач меньшей размерности) и их решения. При этом, на каждом шаге для решения задачи используются результаты решения предыдущих задач.

В общем случае, можно решить задачу, в которой присутствует оптимальная подструктура, проделав следующие шаги:

1. Разбиение задачи на подзадачи меньшего размера.
2. Нахождение оптимального решения подзадач рекурсивно, проделывая такой же трехшаговый алгоритм.
3. Использование полученного решения подзадач для конструирования решения исходной задачи.

Алгоритм является псевдополиномиальным, т.е. это алгоритм, проявляющий экспоненциальный характер только при относительно больших значениях числовых параметров.

# Жадные алгоритмы

Жадный алгоритм – метод решения оптимизационных задач, основанный на том, что процесс принятия решения можно разбить на элементарные шаги, на каждом из которых принимается отдельное решение. Решение, принимаемое на каждом шаге, оптимально только на текущем шаге и принимается без учета предыдущих или последующих решений; т.е. производится локально оптимальный выбор в надежде, что он приведет к оптимальному решению глобальной задачи.

Генетические алгоритмы

Генетический алгоритм – это эвристический алгоритм поиска, применяемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров, основываясь на моделировании биологических механизмов эволюции и популяционной генетике, а не на математических свойствах целевой функции. Генетические алгоритмы не гарантируют нахождения оптимального решения за полиномиальное время и не дают оценку близости решения к оптимальному, но обладают хорошими временными показателями, позволяя найти *достаточно хорошее* решение быстрее других известных детерминированных или эвристических методов.

Для нахождения решения при помощи генетического алгоритма, задача формализуется таким образом, чтобы её решение возможно было закодировать в виде вектора генов (генотипа). Некоторым образом, создается множество генотипов начальной популяции. Каждый генотип оценивается функцией приспособленности, в результате чего каждому генотипу ставится в соответствие определённое значение (приспособленность), определяющее насколько хорошо описываемый им фенотип решает поставленную задачу. Из полученного множества решений (поколения) путём применения генетических операторов скрещивания (кроссовера) и мутации строится множество новых решений (потомки). Для них также вычисляется значение функции приспособленности, и затем, производится отбор (селекция) решений в следующее поколение, основываясь на значении приспособленности.

Эти действия повторяются итеративно, моделируя эволюционный процесс, продолжающийся несколько циклов. Критерием остановки алгоритма могут быть: исчерпание числа поколений, исчерпание времени или нахождение субоптимального решения.

В качестве исследуемых алгоритмов, рассмотрим метод динамического программирования, несколько реализаций жадного и генетический алгоритм. Для метода динамического программирования можно приближенно оценить, начиная с какого объёма входных данных, он проявляет экспоненциальный характер. Также, этот алгоритм является точным, что позволяет оценить приближенность решений, получаемых в ходе работы генетического и жадных алгоритмов.

## Алгоритмическое описание выбранных алгоритмов

# Метод динамического программирования

Пусть есть предприятий. Для каждого предприятия есть возможных размеров вложений . Всего имеется средств. Прибыль от вложения средств в -е предприятие обозначим . Обозначим – количество средств, оставшихся после вложения в -е предприятие; – количество средств, вложенных в -е предприятие. Рассматриваем задачу с конца, т.е. допустим, что средства уже вложены в предприятие. Тогда, максимальная прибыль от вложения в -е предприятие находится по формуле: .

Отсюда, максимальная суммарная прибыль от вложения в и -е предприятия:

Аналогичным образом строим соотношения для оставшихся шагов. На последнем шаге получаем: . На каждом шаге запоминаем управление , при котором достигается максимум. Наконец, посчитав все соотношения, получаем – вектор решения, на котором достигается максимум целевой функции .

# Генетические алгоритмы

Генетические алгоритмы включают в себя множество различных операторов, работающих последовательно, и влияющих на формирование популяций решений. Все операторы, использующиеся в данной работе описаны ниже.

**Кодирование:**

Генотип представляет из себя вектор , где , ∀ .

- индекс размера инвестиций в -е предприятие ( – размер вложений в предприятие);

~ нулевое вложение;

– количество предложений;

– количество компаний;

**Формирование начальной популяции:**

Начальная популяция формируется одним из четырёх различных случайных операторов:

1. Каждому гену случайным образом присваивается значение в диапазоне от 0 до .
2. Каждому гену, начиная с первого случайным образом присваивается значение в диапазоне от 0 до , где – сумма всех предыдущих генов. Этот оператор формирования начальной популяции гарантирует допустимость сгенерированного решения относительно условий задачи.
3. Каждому гену, начиная с последнего случайным образом присваивается значение в диапазоне от 0 до , где – сумма всех последующих генов. Способ аналогичен предыдущему, с той только разницей, что заполнение начинается с конца.
4. Каждому гену случайным образом присваивается значение в диапазоне от 0 до [], где [] – целая часть числа. Данный оператор более или менее равномерно заполняет генотип, что может быть полезно для решения задач со входными данными определенного вида.

**Выбор родительской пары:**

ПАНМИКСИЯ Родительская пара выбирается случайным образом из всего поколения.

**Оператор скрещивания:**

В качестве оператора скрещивания выступает одноточечный кроссовер. Генотип родителей разрывается в одной и той же точке; потомок получается путем соединения части генотипа до точки разрыва первого родителя и части генотипа после точки разрыва второго родителя. Так, если – первый родитель, а , и – точка разрыва, то их потомком будет .

**Оператор мутации:**

Оператор мутации применяется к потомкам и заменяет значение гена, стоящего на случайной позиции, случайным другим значением из множества допустимых значений.

**Оператор селекции:**

В результате работы операторов поиска родительской пары и скрещивания генерируется множество потомков, количеством в два раза превосходящее множество родителей. Два этих множества объединяются в одно репродукционное множество и происходит отбор особей в новое поколение двумя операторами селекции (в зависимости от параметров запуска последовательно двумя, либо отдельно одним из них):

1. **Схема пропорциональной селекции. Особь репродукционного множества переходит в поколение в том случае, если приспособленность этой особи будет больше средней приспособленности множества . Поскольку размер каждого поколения фиксирован и одинаков, могут возникнуть случаи, когда в результате работы данного оператора следующее поколение формируется не полностью. В этом случае, доформирование производится либо случайным выбором из репродукционного множества, что дает возможность добора особей с малым значением функции приспособленности, в генотипах которых может содержаться часть генотипа оптимального решения; либо оператором вращения рулеточного колеса, что также сохраняет такую возможность.**
2. **Метод отбора путём вращения рулеточного колеса (стохастический выбор с возвращением). В первую очередь, вычисляется суммарная приспособленность особей, входящих в репродукционное множество : , где - мощность репродукционного множества, – приспособленность -й особи. Затем, для каждой особи находится вероятность по**падания в следующее поколение, пропорциональная приспособленности этой особи и обратно пропорциональная суммарной приспособленности: . Далее, вычисляется совокупная вероятность для каждой особи: . Случайным образом выбирается действительное число ∈ [0,], где . Если , то в популяцию копируется особь , иначе для копирования в популяцию выбирается особь , для которой выполняется условие . При использовании этого метода для формирования популяции выбор очередной копии производится каждый раз из всего множества решений, образующих репродукционное множество . При этом, одно и то же решение может быть репродуцировано несколько раз.

# Жадный алгоритм

Пусть есть предприятий. Для каждого предприятия есть возможных размеров вложений . Всего имеется средств, причем . Прибыль от вложения средств в -е предприятие обозначим .

Для сравнения с генетическим алгоритмом реализованы четыре варианта жадного алгоритма:

1. **Начиная с первого предприятия, на каждом шаге алгоритма определяется максимальная возможная прибыль от вложения оставшихся средств в** -е предприятие по формуле: , ∀, где .
2. **Аналогично второму варианту, однако просмотр начинается с конца.**
3. Начиная с первого предприятия, на каждом шаге находим наиболее выгодный размер вложений для предприятия. Для этого сравниваем удельные прибыли в диапазоне допустимого решения , ∀, где . Записываем индекс размера наиболее выгодного вложения в компоненту вектора решения, соответствующую данному предприятию:. На последнем шаге вкладываем все оставшиеся средства, т.е. .
4. **Аналогично третьему варианту, за исключением того, что просмотр и нахождение наиболее выгодного размера вложений начинается с последнего предприятия.**

В результате работы каждого из алгоритмов, получаем вектор решения, удовлетворяющий ограничению средств, представляющий какое-то решение.

## Эксперимент

Для генетического алгоритма установлены ограничения по времени и памяти: памяти дается приблизительно столько же, сколько нужно методу динамического программирования; ограничение по времени – половина времени работы динамического программирования, либо, если время очень мало, задается минимальное число итераций (10).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер матрицы предложений; максимальное вложение | Прямой жадный по чистой прибыли | Обратный жадный по чистой прибыли | Прямой жадный по удельной прибыли | Обратный жадный по удельной прибыли | Динамическое программирование | Генетический алгоритм |
| 4х8; 280 | Решение:308; Время: 00.00 | Решение:300 | Решение:310 | Решение:311 | Решение:312 | Решение:310 |
| 15х21; 1000 | Решение:1068; Время: 00.00 | Решение:1104; Время: 00.00 | Решение:1130; Время: 00.00 | Решение:1083; Время: 00.00 | Решение:1158; Время: 00.00 | Решение:1104; Время: 00.00 |
| 89x7; 300 | Решение:342; Время: 00.01 | Решение:330; Время: 00.01 | Решение:342; Время: 00.00 | Решение:337; Время: 00.00 | Решение:381; Время: 00.00 | Решение:346; Время: 00.00 |
| 10x11; 1000 | Решение:1129; Время: 00.00 | Решение:1134; Время: 00.00 | Решение:1124; Время: 00.00 | Решение:1121; Время: 00.00 | Решение:1174; Время: 00.00 | Решение:1176; Время: 00.00 |
| 25х48; 4700 | Решение:5160; Время: 00.00 | Решение:5134; Время: 00.00 | Решение:5180; Время: 00.00 | Решение:5263; Время: 00.00 | Решение:5407; Время: 00.00 | Решение:5164; Время: 00.00 |
| 100х41; 4000 | Решение:4431; Время: 00.00 | Решение:4351; Время: 00.00 | Решение:4491; Время: 00.00 | Решение:4491; Время: 00.00 | Решение:4835; Время: 00.01 | Решение:4381; Время: 00.01 |
| 100x51; 5000 | Решение:5437; Время: 00.01 | Решение:5302; Время: 00.01 | Решение:5627; Время: 00.00 | Решение:5598; Время: 00.00 | Решение:6059; Время: 00.01 | Решение:5344; Время: 00.01 |
| 100х401; 40000 | Решение:43901; Время: 00.00 | Решение:43813; Время: 00.00 | Решение:44604; Время: 00.00 | Решение:44752; Время: 00.00 | Решение:45251; Время: 00.12 | Решение:43761; Время: 00.07 |
| 500х601; 30000 | Решение:32932; Время: 00.04 | Решение:32975; Время: 00.04 | Решение:33617; Время: 00.04 | Решение:33464; Время: 00.04 | Решение:35178; Время: 02.38 | Решение:33074; Время: 01.42 |
| 1000х1301; 65000 | Решение:71374; Время: 00.11 | Решение:71295; Время: 00.11 | Решение:71989; Время: 00.10 | Решение:71998; Время: 00.10 | Решение:75805; Время: 20.20 | Решение:69982; Время: 11.22 |
| 1333х1501; 150000 | Решение:164432; Время: 00.15 | Решение:164423; Время: 00.15 | Решение:167257; Время: 00.15 | Решение:167176; Время: 00.15 | Решение:176501;  Время: 37.77 | Решение:165183; Время: 20.94 |
| 100х10001; 1000000 | Решение:1099204; Время: 00.01 | Решение:1098980; Время: 00.01 | Решение:1003350; Время: 00.02 | Решение:1031813; Время: 00.02 | Решение:1101649; Время: 49.99 | Решение:1045294; Время: 26.17 |
| 10000х50001; 50000000 | Решение:549870621; Время: 03.70 | Решение:549533010; Время: 03.70 | Решение:556564108; Время: 05.64 | Решение:556030243; Время: 05.64 | Не вычислялось | Не вычислялось |

## Заключение

В результате изучения различных подходов к решению задачи оптимального распределения инвестиций, реализации конкретных алгоритмов и сравнения их на практике, можно сделать вывод, что на данный момент безоговорочным лидером среди остальных алгоритмов является метод динамического программирования. К такому заключению ведут несколько причин:

* Динамическое программирование – точный алгоритм, т.е. всегда найдется глобальный оптимум решения, что позволяет получить максимальную выгоду.
* Работа за приемлемое время даже на достаточно больших наборах входных данных (порядка 40с. для совокупного количества вложений ~ 2 000 000); что не является пределом в свете постоянной тенденции роста вычислительных мощностей.
* Низкая сложность реализации.

В свою очередь, генетический алгоритм должен хорошо показывать себя там, где динамическое программирование не может дать ответ за приемлемое время, т.е. на очень больших наборах входных данных. Но, по результатам эксперимента, он ведет себя не стабильно, в связи с большой длиной генотипа и размером алфавита, необходимого для кодирования гена. Поскольку при кроссовере и мутации очень большой шанс выйти за предел допустимости (по суммарному весу), генетический алгоритм лишается своей главной особенности – наследования частей генотипов наиболее приспособленных особей. Также необходимо учитывать, что сложность реализации и настройки работы данного метода в значительной степени превышает сложность остальных алгоритмов. Кроме того, имеется зависимость работоспособности от входных данных, т.к. при различных комбинациях длины генотипа и мощности алфавита изменяются и наиболее подходящие для данного случая операторы (схемы формирования начальной популяции, селекции, мутации и т.д.). На малых наборах данных применение генетического алгоритма теряет смысл на фоне сравнения с динамическим программированием.

Жадные алгоритмы показали себя несколько лучше генетического алгоритма. Разумеется, этот класс методов также не является точным, но у него есть и весомые плюсы:

* Работоспособность сохраняется даже при очень больших объемах входных данных (матрица 10000x50000, время работы ~ 3 с.)
* Жадный алгоритм и не гарантирует близости к оптимуму, но в среднем, на это можно рассчитывать, поскольку на каждой итерации выбирается оптимальное управление, хоть и на данном конкретном этапе.
* При условии выбора лучшего решения после результатов работы четырех представленных алгоритмов, в среднем, получаем решение, достаточно близкое к оптимуму
* Максимальная простота реализации среди представленных алгоритмов.

В приложении к данной задаче, рационально применение метода динамического программирования, в случаях, когда важна точность решения и когда количество вложений в совокупности по всем предприятиям не превышает 10 000 000.

В случае, когда объем данных значительно больше, и в угоду скорости нахождения решения возможно пожертвовать точностью, оптимальным будет реализация одного или нескольких вариантов жадного алгоритма, с последующим выбором наилучшего результата.

## Литература

1. Курейчик В. М., Лебедев Б. К., Лебедев О. К. Поисковая адаптация: теория и практика. — М: Физматлит, 2006. — С. 272. — ISBN 5-9221-0749-6
2. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы. — 2-е изд.. — М: Горячая линия-Телеком, 2008. — С. 452. — ISBN 5-93517-103-1
3. Динамическое программирование / С. М. Окулов, О. А. Пестов. —М. : БИНОМ. Лаборатория знаний, 2012.--296 с.
4. Методы решения задачи о рюкзаке <https://nauchforum.ru/studconf/tech/xx/6006>
5. Додонова М.М., научный руководитель:канд. физ.-мат. наук, доцент Баранова И.В. Изучение различных постановок задачи о рюкзаке и методов их решения.
6. NP-полнота задачи <https://ru.wikipedia.org/wiki/NP-полная_задача>
7. С.А. Баркалов, О.Н. Бакунец, И.В. Гуреева, В.Н. Колпачев И.Б. Руссман Оптимизационный модели распределения инвестиций на предприятии по видам деятельности
8. Батищев Д.И., Неймарк Е.А., Старостин Н.В. Применение генетических алгоритмов к решению задач дискретной оптимизации: Учебное пособие. – Н.Новгород, изд-во ННГУ им. Н.И.Лобачевского, 2006. – 136с
9. Генетические алгоритмы в задаче оптимизации действительных параметров <http://mirznanii.com/a/8821-2/geneticheskie-algoritmy-v-zadache-optimizatsii-deystvitelnykh-parametrov>
10. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982, .- 416с.

## Приложение

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Backpack\_plus

{

class Program

{

static void Main(string[] args)

{

DataCase example;

example = new DataCase(4, 8, 280);

example.PrintData();

Console.WriteLine("Прямой жадный алгоритм: ");

System.Diagnostics.Stopwatch stopwatchGreed = new System.Diagnostics.Stopwatch();

stopwatchGreed.Start();

example.GreedyAlg();

stopwatchGreed.Stop();

TimeSpan tsGreed = stopwatchGreed.Elapsed;

string elapsedTimeGreed = String.Format("{0:00}:{1:00}:{2:00}.{3:00}\n", tsGreed.Hours, tsGreed.Minutes, tsGreed.Seconds, tsGreed.Milliseconds / 10);

Console.WriteLine("Greed RunTime " + elapsedTimeGreed);

Console.WriteLine("Пропорциональный жадный алгоритм: ");

System.Diagnostics.Stopwatch stopwatchProportionalGreed = new System.Diagnostics.Stopwatch();

stopwatchProportionalGreed.Start();

example.GreedProportional();

stopwatchProportionalGreed.Stop();

TimeSpan tsProportionalGreed = stopwatchProportionalGreed.Elapsed;

string elapsedTimeProportionalGreed = String.Format("{0:00}:{1:00}:{2:00}.{3:00}\n", tsProportionalGreed.Hours, tsProportionalGreed.Minutes, tsProportionalGreed.Seconds, tsProportionalGreed.Milliseconds / 10);

Console.WriteLine("ProportionalGreed RunTime " + elapsedTimeProportionalGreed);

Console.WriteLine("Динамическое программирование:");

System.Diagnostics.Stopwatch stopwatchDyn = new System.Diagnostics.Stopwatch();

stopwatchDyn.Start();

example.DynamicProg();

stopwatchDyn.Stop();

TimeSpan tsDyn = stopwatchDyn.Elapsed;

string elapsedTimeDyn = String.Format("{0:00}:{1:00}:{2:00}.{3:00}\n", tsDyn.Hours, tsDyn.Minutes, tsDyn.Seconds, tsDyn.Milliseconds / 10);

Console.WriteLine("Dynamic RunTime " + elapsedTimeDyn);

Console.WriteLine("Генетический алгоритм: ");

System.Diagnostics.Stopwatch stopwatchGen = new System.Diagnostics.Stopwatch();

stopwatchGen.Start();

example.GeneticAlg(stopwatchDyn);

stopwatchGen.Stop();

TimeSpan tsGen = stopwatchGen.Elapsed;

string elapsedTimeGen = String.Format("{0:00}:{1:00}:{2:00}.{3:00}\n", tsGen.Hours, tsGen.Minutes, tsGen.Seconds, tsGen.Milliseconds / 10);

Console.WriteLine("Genetic RunTime " + elapsedTimeGen);

//Console.WriteLine("Полный допустимый перебор: ");

//System.Diagnostics.Stopwatch stopwatchBrute = new System.Diagnostics.Stopwatch();

//stopwatchBrute.Start();

//example.BruteForce();

//stopwatchBrute.Stop();

//TimeSpan tsBrute = stopwatchBrute.Elapsed;

//string elapsedTimeBrute = String.Format("{0:00}:{1:00}:{2:00}.{3:00}\n", tsBrute.Hours, tsBrute.Minutes, tsBrute.Seconds, tsBrute.Milliseconds / 10);

//Console.WriteLine("Brute RunTime " + elapsedTimeBrute);

}

}

class Management

{

public int[] stratedy;

public int profit;

static int n;

public Management()

{

stratedy = new int[n];

}

public Management(int pos, int prof)

{

stratedy = new int[n];

stratedy[0] = pos;

profit = prof;

}

public Management(Management a)

{

stratedy = new int[n];

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

stratedy[i] = a.stratedy[i];

profit = a.profit;

}

public static void SetN(int a)

{

n = a;

}

public void Copy(Management a)

{

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

stratedy[i] = a.stratedy[i];

profit = a.profit;

}

public void Print()

{

Console.Write("(");

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

Console.Write(stratedy[i] + ", ");

Console.WriteLine("); Profit = " + profit);

}

}

class Individ

{

int[] genotype;

int resources;

int profit;

double fitness;

static int NoC;

static int NoAS;

static int[,] tableOfOffers;

static int[] sizeOfAttachments;

static int maxResources;

public static void StaticInit(int noc, int noas, int r, int[,] table, int[] attachments)

{

NoC = noc;

NoAS = noas;

maxResources = r;

tableOfOffers = table;

sizeOfAttachments = attachments;

}

public Individ()

{

genotype = new int[NoC];

}

public Individ(Random rnd, int flag=0) //RandInit

{

genotype = new int[NoC];

int rangeRandom = NoAS;

switch (flag)

{

case 1:

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

{

int rand = rnd.Next(rangeRandom);

genotype[i] = rand;

rangeRandom -= rand;

}

break;

case 2:

for (int i = NoC - 1; i >= 0; i--)

{

int rand = rnd.Next(rangeRandom);

genotype[i] = rand;

rangeRandom -= rand;

}

break;

case 3:

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

genotype[i] = rnd.Next(NoAS/2);

break;

default:

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

genotype[i] = rnd.Next(NoAS);

break;

}

UpdateFitness();

}

public Individ(Individ a, Individ b, int pos) //Crossover

{

genotype = new int[NoC];

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

genotype[i] = a.genotype[i];

for (int i = pos; i < NoC; i++)

genotype[i] = b.genotype[i];

UpdateFitness();

}

public Individ(Individ a)

{

genotype = new int[NoC];

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

genotype[i] = a.genotype[i];

UpdateFitness();

}

public void Set(int pos, int value)

{

genotype[pos] = value;

UpdateFitness();

}

public double Mutation(Random rnd)

{

double oldFitness = fitness;

genotype[rnd.Next(NoC)] = rnd.Next(NoAS);

UpdateFitness();

return (fitness - oldFitness);

}

public double GetFitness()

{

return fitness;

}

public int GetProfit()

{

return profit;

}

public bool ResourcesCheck()

{

if (resources <= maxResources)

return true;

else

return false;

}

public void Print()

{

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

{

Console.Write("{0:000} ", sizeOfAttachments[genotype[i]]);

}

Console.WriteLine(" Resources: " + resources + "; Profit: " + profit + "; Fitness: {0:0.0000}", fitness);

}

private void UpdateFitness()

{

int p = 0;

int r = 0;

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

{

p += tableOfOffers[i, genotype[i]];

r += sizeOfAttachments[genotype[i]];

}

profit = p;

resources = r;

if (resources > maxResources)

{

double r2 = Math.Pow(resources, 2);

double max2 = Math.Pow(maxResources, 2);

double temp = r2 - max2;

double sq = Math.Pow(temp, 0.5);

fitness = (double)(profit - sq); //Если перерасход ресурсов приспосеобленность уменьшается

if (fitness < 0)

fitness = profit/10;

}

else

fitness = (double)(profit + (maxResources - resources) / 2); //Если ресурсы использованы не все, то приспособленность

//fitness = (double)(profit - Math.Pow(Math.Pow(resources, 3) - Math.Pow(maxResources, 3), 1 / 3.0));

//увеличивается на половину разницы (теоретическая прибыль)

}

}

class DataCase

{

int[,] tableOfOffers;

int numberOfCompany;

int numberOfAttachmentSizes;

int[] sizeOfAttachment;

int resources;

int[] U;

Random rnd;

private void BaseInitValues()

{

tableOfOffers[0,0] = 0; tableOfOffers[0,1] = 50; tableOfOffers[0,2] = 120; tableOfOffers[0,3] = 140; tableOfOffers[0,4] = 150; tableOfOffers[0,5] = 200; tableOfOffers[0,6] = 250;

tableOfOffers[1,0] = 0; tableOfOffers[1,1] = 60; tableOfOffers[1,2] = 130; tableOfOffers[1,3] = 140; tableOfOffers[1,4] = 130; tableOfOffers[1,5] = 160; tableOfOffers[1,6] = 200;

tableOfOffers[2,0] = 0; tableOfOffers[2,1] = 30; tableOfOffers[2,2] = 60; tableOfOffers[2,3] = 100; tableOfOffers[2,4] = 130; tableOfOffers[2,5] = 200; tableOfOffers[2,6] = 250;

tableOfOffers[3,0] = 0; tableOfOffers[3,1] = 40; tableOfOffers[3,2] = 100; tableOfOffers[3,3] = 110; tableOfOffers[3,4] = 120; tableOfOffers[3,5] = 160; tableOfOffers[3,6] = 220;

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

sizeOfAttachment[i] = i \* 50;

}

private void RandInitValues()

{

int step = resources / (numberOfAttachmentSizes - 1);

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

sizeOfAttachment[i] = i \* step;

Random rnd = new Random();

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

for (int j = 1; j < numberOfAttachmentSizes; j++)

tableOfOffers[i, j] = sizeOfAttachment[j] \* 9 / 10 + rnd.Next((step + sizeOfAttachment[j]) \* 2 / 10);

}

private int Method(int r, out int[] U, int depth = 0)

{

U = new int[numberOfCompany];

if (depth < numberOfCompany)

{

int maxResult = 0;

int maxU = 0;

int lim = r / sizeOfAttachment[1];

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

{

int[] tempU = new int[numberOfCompany];

int temp = tableOfOffers[depth, i] + Method(r - sizeOfAttachment[i], out tempU, depth + 1);

if (temp > maxResult)

{

maxResult = temp;

maxU = sizeOfAttachment[i];

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

U[j] = tempU[j];

U[depth] = maxU;

}

}

return maxResult;

}

else

return 0;

}

public void PrintData()

{

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

Console.Write(sizeOfAttachment[i] + " ");

Console.WriteLine();

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

{

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

Console.Write(tableOfOffers[i,j] + " ");

Console.WriteLine();

}

}

private void MemoryAllocation(int a = 4, int b = 7, int r = 300)

{

numberOfCompany = a;

numberOfAttachmentSizes = b;

U = new int[numberOfCompany];

sizeOfAttachment = new int[numberOfAttachmentSizes];

tableOfOffers = new int[numberOfCompany, numberOfAttachmentSizes];

resources = r;

}

public DataCase(int a, int b, int r)

{

rnd = new Random();

if (r % (b - 1) == 0)

{

MemoryAllocation(a, b, r);

RandInitValues();

}

else

{

Console.WriteLine("Недопустимые значения, инициализация по умолчанию!");

MemoryAllocation();

BaseInitValues(); //Метод инициализации проверочными значениями

}

Individ.StaticInit(numberOfCompany, numberOfAttachmentSizes, resources, tableOfOffers, sizeOfAttachment);

}

public void BruteForce()

{

int maxProfit = Method(resources, out U);

Console.Write("Максимальная прибыль в размере " + maxProfit + " достигается на наборе (");

for (int i = 0; i < numberOfCompany - 1; i++)

{

Console.Write(U[i] + ", ");

}

Console.Write(U[numberOfCompany - 1] + ")");

Console.WriteLine();

}

public void DynamicProg()

{

Management result = DynamicProgramming();

Console.Write("Максимальная прибыль в размере " + result.profit + " достигается на наборе (");

for (int i = 0; i < numberOfCompany - 1; i++)

{

Console.Write(result.stratedy[i] + ", ");

}

Console.Write(result.stratedy[numberOfCompany - 1] + ")");

Console.WriteLine();

}

private Management DynamicProgramming()

{

Management.SetN(numberOfCompany);

Management[,] dynamicTable = new Management[2,numberOfAttachmentSizes];

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

dynamicTable[0, i] = new Management(i, tableOfOffers[0, i]);

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

dynamicTable[1, i] = new Management();

for (int i = 1; i < numberOfCompany - 1; i++) //i - номер рассматриваемой компании

{

int curr = i % 2;

int prev = (i + 1) % 2;

for (int j = 0; j < numberOfAttachmentSizes; j++) //j - номер заполняемой ячейки (по доступному размеру вложений)

{

Management max = new Management(dynamicTable[prev, j]);

for (int k = 1; k <= j; k++)

{

int newProfit = dynamicTable[prev, j - k].profit + tableOfOffers[i, k];

if (max.profit < newProfit)

{

max.Copy(dynamicTable[prev, j - k]);

max.stratedy[i] = k;

max.profit = newProfit;

}

}

dynamicTable[curr, j].Copy(max);

}

}

int last = numberOfCompany % 2;

Management result = new Management(dynamicTable[last, numberOfAttachmentSizes - 1]);

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

{

int newProfit = tableOfOffers[numberOfCompany - 1, i] + dynamicTable[last, numberOfAttachmentSizes - i - 1].profit;

if (result.profit < newProfit)

{

result.Copy(dynamicTable[last, numberOfAttachmentSizes - i - 1]);

result.stratedy[numberOfCompany - 1] = i;

result.profit = newProfit;

}

}

return result;

}

public void GeneticAlg(System.Diagnostics.Stopwatch record) //Отвратительно работает, когда количество компаний больше количества предложений, т.е. когда в оптимальном генотипе есть множество нулей

{

int populationSize = numberOfAttachmentSizes;

Individ[] population = new Individ[populationSize];

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

population[i] = new Individ(rnd, rnd.Next(3)+1);

//for (int i = 0; i < populationSize; i++)

// population[i].Print();

System.Diagnostics.Stopwatch stopwatch = new System.Diagnostics.Stopwatch();

stopwatch.Start();

//int testcount = 0;

while (stopwatch.ElapsedMilliseconds < record.ElapsedMilliseconds / 2/\*testcount++ < 10\*/)

{

double sumFitness = 0;

Individ[] child = new Individ[populationSize \* 3];

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

{

child[i] = new Individ(population[i]);

sumFitness += child[i].GetFitness();

}

for (int i = populationSize; i < populationSize \* 3; i++) //Создание потомков путем одноточечного кроссовера

{

child[i] = Crossover(population, populationSize);

sumFitness += child[i].GetFitness();

}

for (int i = 0; i < populationSize / 10; i++)

sumFitness += child[rnd.Next(populationSize, populationSize \* 3)].Mutation(rnd); //Мутации производятся только среди потомков

double averageFitness = sumFitness / (populationSize \* 3);

//Console.WriteLine(sumFitness);

//population = WheelRotation(child, populationSize, sumFitness);

//population = WheelRotation(child, populationSize, Convert.ToInt32(Math.Ceiling(sumFitness))); //Старая версия

//Console.WriteLine("Потомки:");

//for (int i = 0; i < populationSize \* 2; i++)

// child[i].Print();

population = ProportionalSelection(population, child, populationSize, averageFitness);

//Console.WriteLine("Суммарная приспособленность равна: " + sumFitness);

//Console.WriteLine("Средняя приспособленность: " + averageFitness); //Округляет вверх всегда

//for (int i = 0; i < populationSize; i++)

// population[i].Print();

}

stopwatch.Stop();

Console.WriteLine("Наилучшее найденное решение: ");

population[FindBest(population, populationSize)].Print();

//for (int i = 0; i < populationSize; i++)

// Console.WriteLine(population[i].GetFitness());

}

private Individ Crossover(Individ[] population, int populationSize)

{

Individ a = new Individ(population[rnd.Next(populationSize)], population[rnd.Next(populationSize)], rnd.Next(1, numberOfCompany));

return a;

}

private int FindBest(Individ[] population, int populationSize)

{

int max = 0;

int maxPos = 0;

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

if (max < population[i].GetProfit() && population[i].ResourcesCheck())

{

max = population[i].GetProfit();

maxPos = i;

}

return maxPos;

}

private Individ[] ProportionalSelection(Individ[] population, Individ[] child, int populationSize, double averageFitness) //Работает не очень, быстро сходится

{

Individ[] newPopulation = new Individ[populationSize];

int counter = 0;

for (int i = 0; i < populationSize && counter < populationSize; i++)

if (averageFitness < population[i].GetFitness())

newPopulation[counter++] = new Individ(population[i]);

for (int i = 0; i < populationSize\*2 && counter < populationSize; i++)

if(averageFitness < child[i].GetFitness())

newPopulation[counter++] = new Individ(child[i]);

while (counter < populationSize)

{

newPopulation[counter++] = new Individ(child[rnd.Next(populationSize \* 2)]);

}

return newPopulation;

}

private Individ[] WheelRotation(Individ[] population, int populationSize, int sumFitness)

{

Individ[] newPopulation = new Individ[populationSize];

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

{

int rangePoint = rnd.Next(sumFitness);

double topRange = 0;

for (int j = 0; j < populationSize \* 3; j++)

{

topRange += population[j].GetFitness();

if (rangePoint < topRange)

{

newPopulation[i] = new Individ(population[j]);

break;

}

}

}

return newPopulation;

}

public void GreedProportional()

{

Console.WriteLine();

Individ[] solutionPair = new Individ[2];

for (int i = 0; i < 2; i++)

solutionPair[i] = new Individ();

int counter = numberOfAttachmentSizes;

for (int i = 0; i < numberOfCompany - 1; i++)

{

double maxGain = 0;

int pos = 0;

for (int j = 1; j < counter; j++)

{

double gain = (double)tableOfOffers[i, j] / sizeOfAttachment[j];

if (gain >= maxGain)

{

maxGain = gain;

pos = j;

}

}

solutionPair[0].Set(i, pos);

counter -= pos;

}

solutionPair[0].Set(numberOfCompany - 1, counter - 1);

counter = numberOfAttachmentSizes;

for (int i = numberOfCompany - 1; i > 0; i--)

{

double maxGain = 0;

int pos = 0;

for (int j = 1; j < counter; j++)

{

double gain = (double)tableOfOffers[i, j] / sizeOfAttachment[j];

if (gain >= maxGain)

{

maxGain = gain;

pos = j;

}

}

solutionPair[1].Set(i, pos);

counter -= pos;

}

solutionPair[1].Set(0, counter - 1);

Console.WriteLine("Proportional Greed: ");

solutionPair[0].Print();

solutionPair[1].Print();

}

public void GreedyAlg()

{

Individ[] solutionPair = new Individ[2];

for (int i = 0; i < 2; i++)

solutionPair[i] = new Individ();

int counter = numberOfAttachmentSizes;

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

{

int maxGain = -sizeOfAttachment[numberOfAttachmentSizes-1];

int pos = -1;

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

{

int gain = tableOfOffers[i, j] - sizeOfAttachment[j];

if(gain >= maxGain)

{

maxGain = gain;

pos = j;

}

}

solutionPair[0].Set(i, pos);

counter -= pos;

}

counter = numberOfAttachmentSizes;

for (int i = numberOfCompany-1; i >= 0; i--)

{

int maxGain = -sizeOfAttachment[numberOfAttachmentSizes - 1];

int pos = -1;

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

{

int gain = tableOfOffers[i, j] - sizeOfAttachment[j];

if (gain >= maxGain)

{

maxGain = gain;

pos = j;

}

}

solutionPair[1].Set(i, pos);

counter -= pos;

}

Console.WriteLine("Жадные решения: ");

solutionPair[0].Print();

solutionPair[1].Print();

}

public void RandSolution()

{

int numberOfSolutions = 5;

Individ[] RandSolution = new Individ[numberOfSolutions];

Console.WriteLine("Случайные решения: ");

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

{

RandSolution[i] = new Individ(rnd, rnd.Next(1, 2));

RandSolution[i].Print();

}

Console.WriteLine("Конец случайных решений!");

}

}

}