

РЕФЕРАТ

Дипломная работа: 41 с., 30 рис., 50 табл., 7 источников.

Ключевые слова: ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ, МАШИННОЕ ОБУЧЕНИЕ, НР-ПОЛНЫЕ ЗАДАЧИ, ЗАДАЧА О РЮКЗАКЕ, ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ, МНОГОМЕРНЫЙ РЮКЗАК

Объект исследования - задача о многомерном рюкзаке

Цель работы - исследование применения генетического алгоритма к задаче о многомерном рюкзаке.

Результатом работы является программа, реализующая генетический алгоритм на языке C# и его модификации. Разработанная программа позволяет найти точные максимумы для малых наборов предметов(<50 предметов) в 100% случаев и для больших(100 предметов) - в 43,3% случаев. В остальных случаях программа находит локальный максимум, различающийся с исходным не более чем на 0,21%.

Содержание

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	4
ВВЕДЕНИЕ	5
1 Теоретическая часть	6
1.1 Задача о многомерном рюкзаке	6
1.1.1 Постановка задачи	6
1.1.2 Доказательство NP-полноты	6
1.1.3 Вычислительная сложность задачи	8
1.2 Генетические алгоритмы	8
1.2.1 Этапы работы алгоритма	9
1.2.2 Выбор параметров алгоритма в приложении к задаче о многомерном рюкзаке	10
1.2.3 Дополнения и улучшения, внесённые в генетический алгоритм	12
2 Практическая часть	14
2.1 Наборы тестов	14
2.2 Первый набор тестов	15
2.2.1 Мутация в одной позиции	15
2.2.2 Инверсионная мутация	16
2.2.3 Выводы по первому тестированию	17
2.3 Второй набор тестов	17
ЗАКЛЮЧЕНИЕ	20
ПРИЛОЖЕНИЯ	22
А Исходный код программы	23

ОБОЗНАЧЕНИЯ

В данной работе использованы следующие обозначения и сокращения.

Класс проблем NP (nondeterministic polynomial) – множество проблем разрешимости, решение которых возможно проверить на машине Тьюринга за время, не превосходящее полинома от размера входных данных.

NP-полная проблема – проблема из класса NP, к которой можно свести любую другую проблему из этого класса за полиномиальное время.

.NET Framework - программная платформа, выпущенная фирмой Microsoft. Платформа состоит из среды CLR, позволяющей исполнять программы, написанные на разных языках и собранные специально под эту среду, и библиотек классов .NET Framework.

ВВЕДЕНИЕ

В ходе развития компьютерных наук человечеством встретилось с классом NP-полных задач. Такие задачи решаются алгоритмически за недетерминированное полиномиальное время, что существенно затрудняет поиск решения таких задач в приемлемые сроки. Одной из таких задач является задача о рюкзаке (Knapsack problem) и её модификации. Объектом исследования данной работы является задача о многомерном рюкзаке (Multidimensional knapsack problem). Предмет исследования - решение задачи о многомерном рюкзаке с использованием генетического алгоритма. Многие прикладные проблемы могут быть формализованы в виде рассматриваемой задачи. Примерами таких проблем являются размещение процессоров и баз данных в системе распределенных вычислений (см. [1]), погрузка груза и контроль бюджета (см. [2]), задачи раскройки (см. [3]) и др. Решение этих проблем обуславливает актуальность решения задачи о рюкзаке.

Цель исследования - решить задачу о многомерном рюкзаке с использованием генетического алгоритма. Для достижения цели были поставлены следующие задачи:

- Исследовать генетические алгоритмы в применении к NP-полным задачам
- Спроектировать и реализовать генетический алгоритм, решающий задачу о многомерном рюкзаке
- Провести оценку эффективности генетического алгоритма в решении поставленной задачи, используя набор готовых тестов

Для реализации алгоритма был выбран язык C# ввиду удобства и простоты работы.

1 Теоретическая часть

1.1 Задача о многомерном рюкзаке

В данной работе рассматривается задача о многомерном рюкзаке (Multidimensional 0-1 knapsack problem, МКР). Эта задача является модификацией классической задачи о рюкзаке, поставленной в 19 веке Джорджем Мэттьюсоном. (см. [4]) Данный же вариант задачи впервые был предложен Клиффордом Петерсеном в 1967 году. (см. [5])

1.1.1 Постановка задачи

Постановка задачи такова:

Пусть существует набор S , состоящий из N предметов, каждый из которых имеет стоимость $c_i \in \mathbb{R}^+$ и набор размеров S $s_{ij} \in \mathbb{R}^+$, где $i \in 1, 2, \dots, N$, $j \in 1, 2, \dots, M$, $M, N \in \mathbb{N}$. Пусть также существует рюкзак с ограничениями по вместимости по измерениям $r_j \in \mathbb{R}^+$. Требуется максимизировать сумму

$$\sum_{i=1}^N c_i x_i$$

где $x_i \in \{0, 1\}$ при условии

$$\sum_{i=1}^N s_{ij} x_i < r_j \forall j \in \{1, 2, \dots, M\} \quad (1.1)$$

Обычную задачу о рюкзаке можно рассматривать как частный случай при $M = 1$. И обычная задача, и её модификация являются NP-полными задачами.

1.1.2 Доказательство NP-полноты

Для доказательства NP-полноты рассматриваемой задачи достаточно доказать NP-полноту обычной задачи о рюкзаке. Сначала покажем, что задача принадлежит к классу NP. Входными данными является набор предметов S , процесс проверки

правильности решения заключается в вычислении сумм

$$\sum_{i \in S} s_i x_i$$

и

$$\sum_{i \in S} c_i x_i$$

Очевидно, такое вычисление требует затрат полиномиального времени в зависимости от вводимых данных.

Затем, покажем что задача о рюкзаке является NP-полной. Для этого воспользуемся задачей разбиения множества чисел, для которой доказана NP-полнота. Требуется показать, что существует такая полиномиальновременная редукция $Q(\cdot)$, что $Q(X)$ дает ответ «Да» для задачи о рюкзаке тогда, и только тогда, когда X даёт ответ «Да» для задачи разбиения.

Обозначим ёмкость рюкзака как R , максимум, для которого рассматривается задача как V . Предположим, что существует набор a_i для задачи разбиения, рассмотрим соответствующую ему задачу о рюкзаке: $c_i = a_i, s_i = a_i \forall i \in \{1, 2, \dots, N\}$, при этом

$$R = V = \frac{1}{2} \sum_{i=1}^n a_i$$

. $Q(\cdot)$ здесь - процесс, преобразующий задачу о разбиении к задаче о рюкзаке. Очевидно, что этот процесс является полиномиальным от входных данных.

Если X является тем случаем, для которого задача о разбиении имеет ответ «Да», то существуют такие множества S и T , что

$$\sum_{i \in S} a_i = \sum_{i \in T} a_i = \frac{1}{2} \sum_{i=1}^n a_i$$

. Пусть наш рюкзак содержит предметы из S , из чего следует, что

$$\sum_{i \in S} s_i = \sum_{i \in S} a_i = R$$

и

$$\sum_{i \in S} c_i = \sum_{i \in S} a_i = V$$

. Следовательно, $Q(X)$ является случаем с ответом «Да» для задачи о рюкзаке. Наоборот, если $Q(X)$ является случаем с ответом «Да» для задачи о рюкзаке, рассмотрим для выбранного набора S такое T , что $T = \{1, 2, \dots, n\} \sim S$. Имеем

$$\sum_{i \in S} s_i = \sum_{i \in S} a_i \leq R = \frac{1}{2} \sum_{i=1}^n a_i$$

и также

$$\sum_{i \in S} c_i = \sum_{i \in S} a_i \geq V = \frac{1}{2} \sum_{i=1}^n a_i$$

Отсюда получаем, что

$$\sum_{i \in S} a_i = \frac{1}{2} \sum_{i=1}^n a_i$$

и

$$\sum_{i \in T} a_i = \sum_{i=1}^n a_i - \frac{1}{2} \sum_{i=1}^n a_i = \frac{1}{2} \sum_{i=1}^n a_i$$

Следовательно, $\{S, T\}$ – искомое разбиение, и X является случаем с ответом «Да» для задачи о разбиении. Таким образом, NP-полнота задачи о рюкзаке доказана.

1.1.3 Вычислительная сложность задачи

Найдем функцию вычислительной сложности переборного решения рассматриваемой задачи в зависимости от параметров M и N . Для того, чтобы найти оптимальное решение, требуется перебрать все 2^N возможных решений, для каждого из которых требуется вычислить M сумм, содержащих N слагаемых. Таким образом, сложность переборного алгоритма $f(N, M)$ может быть оценена как $f(N, M) = O(2^N NM)$ – экспоненциальная от параметра N .

1.2 Генетические алгоритмы

Генетические алгоритмы являются семейством в множестве эвристических алгоритмов. Впервые такой алгоритм был предложен А. Фразером. (см [6]) В генетическом алгоритме рассматривается популяция возможных решений (особей, фенотипов), которая итерационно подвергается эволюционному процессу, направленному на получение лучших решений. У каждого возможного решения существует набор свойств (генотип), который может быть изменен.

Решение задачи с помощью такого алгоритма требует нескольких предварительных этапов:

- Выбор кодирования генотипа.

На этом этапе нужно выбрать способ кодирования генотипа, который будет эффективен для данной задачи. Такой генотип должен позволять однозначно восстановить соответствующий ему фенотип.

- Выбор начального приближения.

Для запуска итерационного процесса требуется создать начальное множество генотипов - нулевое поколение. В зависимости от способа задания начального поколения скорость поиска оптимального решения может меняться. Размер

поколения также оказывает влияние на эффективность алгоритма: слишком маленькое поколение подавляет разнообразие и приводит к попаданию в локальные максимумы, слишком большое - увеличивает число операций на каждой итерации, чем замедляет работу алгоритма.

- **Выбор мутации.**

На каждой итерации алгоритма заранее определенная часть пула генотипов подвергнется мутациям, то есть определенным случайным образом изменяются их составляющие. Если мутации подвергается слишком малая часть пула, то генетический алгоритм не сможет покинуть локальные максимумы, если же слишком большая - алгоритм потеряет свойство сохранения признаков фенотипа.

- **Выбор механизма скрещивания (кроссинговера).**

После мутации происходит создание новых генотипов из частей старых, что обеспечивает сохранение признаков фенотипа родителя. Алгоритм скрещивания позволяет получить из двух родительских генотипов два различных дочерних генотипа.

- **Выбор функции оценки(фитнесс-функции).**

Такая функция позволяет оценивать генотипы с точки зрения их близости к оптимальному решению и отбирать из них лучшие на каждой итерации.

1.2.1 Этапы работы алгоритма

- Создается нулевое поколение генотипов с использованием заданного алгоритма начального приближения

- Запускается итерационный процесс

Случайным образом выбирается часть поколения, которая подвергнется мутации

Выбранная часть пула генотипов мутируется с использованием заданного алгоритма мутации

Мутировавшие генотипы замещают собой исходные в поколении, не мутировавшие – остаются без изменений

Из пула генотипов выбираются пары для скрещивания

Производится скрещивание с использованием заданного алгоритма

С использованием заданной функции оценки из результатов скрещивания выбираются лучшие

Если выполнено условие останова - например, достигнут предел числа итераций или известный максимум, то итерационный процесс завершается, в противном случае начинается следующая итерация.

- Результат итерационного процесса отдается пользователю

1.2.2 Выбор параметров алгоритма в приложении к задаче о многомерном рюкзаке

Наиболее естественным кодированием возможного решения задачи о рюкзаке в генотип является бинарная последовательность длины N , состоящая из нулей и единиц. Каждый i -й элемент такой последовательности является индикатором вхождения i -го предмета в текущее решение. Такая модель требует наличия проверки корректности генотипа - соблюдения условия 1.1. Этот генотип позволяет однозначно восстановить фенотип и провести оценку.

Для генерации начального приближения был использован жадный алгоритм. Сначала создается генотип из единиц, соответствующий конфигурации рюкзака, в который положены все предметы. Затем в случайном порядке единицы заменяются на нули, пока полученный генотип не будет удовлетворять условию корректности 1.1. Если полученный генотип состоит только из нулей или только из единиц (отсутствие решения или тривиальный случай), то эволюционный процесс не запускается. В противном случае полученный генотип подвергается мутации до заполнения пула решений.

В ходе работы было реализовано несколько алгоритмов мутации и скрещивания с целью сравнения их эффективности. Были реализованы следующие алгоритмы мутации:

- Мутация в одной позиции, при которой инвертируется значение в одной случайно выбранной позиции генотипа (см. 1). Данный алгоритм наиболее прост в реализации и не требует дополнительных условий.
- Инверсионная мутация, при которой случайно выбранная половина генотипа заменяется на противоположные значения (см. 2). Данный алгоритм обеспечивает высокую изменчивость, что может ускорить процесс перебора возможных решений. С другой стороны, высокая изменчивость может стать причиной неустойчивости эволюционного процесса.

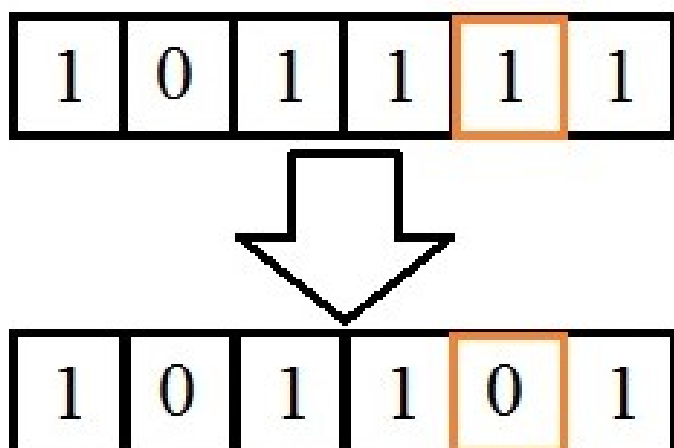


Рис. 1: Одноточечная мутация

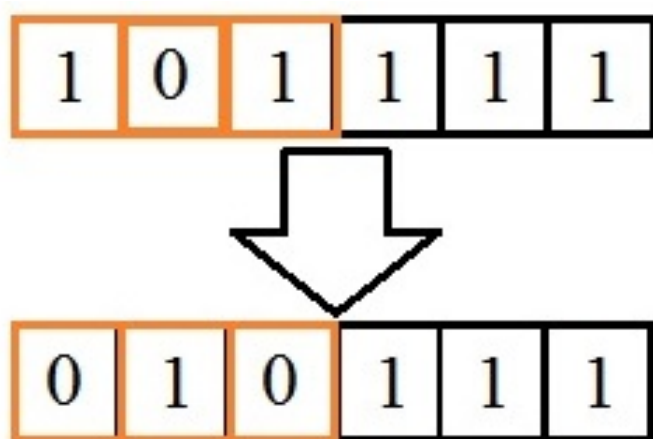


Рис. 2: Инверсионная мутация

Были реализованы следующие алгоритмы скрещивания:

- Скрещивание по одной точке, при котором выбирается произвольная позиция в последовательности генотипа, значения до позиции берутся из первого генотипа, на позиции и после - из второго (см. 3). Наиболее простое из возможных скрещиваний, позволяющее получить разнообразные особи при повторном применении для одинаковых родительских генотипов.
- Скрещивание по двум точкам, при котором выбираются две различные произвольные точки, значения внутри полученного интервала и в самих точках берутся из первого генотипа, вне интервала - из второго (см. 4). Обеспечивает ещё большую вариативность, чем одноточечное скрещивание, но требует проверки на расположение границ интервала друг относительно друга.
- Побитовое скрещивание, при котором значения на нечетных позициях берутся из первого генотипа, на четных - из второго (см. 5). Не требует проверок, но и не дает разнообразия генотипов при повторном применении для одинаковых родительских генотипов.



Рис. 3: Скрещивание по одной точке

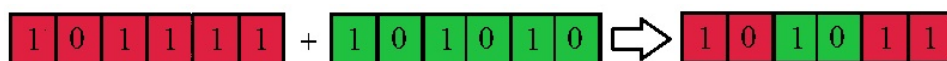


Рис. 4: Скрещивание по двум точкам

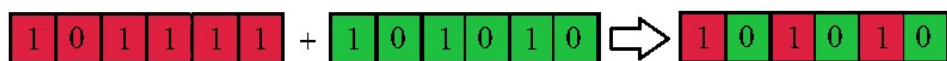


Рис. 5: Побитовое скрещивание

В качестве функции оценки используется стоимость всех предметов, содержащихся в рюкзаке-фенотипе, соответствующем генотипу.

1.2.3 Дополнения и улучшения, внесённые в генетический алгоритм

В связи со спецификой задачи и сложностями, возникшими при тестировании на практике, в алгоритм были внесены дополнения и улучшения.

Важнейшее дополнение – проверка и исправление генотипов на корректность после мутации и скрещивания. Если соответствующий генотипу рюкзак не удовлетворяет

условию корректности 1.1, то производится принудительная коррекция генотипа: значения индикаторов начинают последовательно зануляться до достижения фенотипом корректности. Такая операция позволяет эволюционному процессу функционировать, но существенно замедляет процесс работы генетического алгоритма. Также эта коррекция негативным образом влияет на свойство наследственности генетического алгоритма, так как скорректированный генотип по определению не может кодировать решение лучше решения, заданного исходным генотипом.

Были реализованы два принципиальных улучшения алгоритма: пулы лучших генотипов и механика перезапуск эволюционного процесса.

Пул лучших генотипов для текущего эволюционного процесса хранит в себе лучшие с точки зрения оценки генотипы с начала текущей эволюции. Этот пул позволяет реализовать возврат к лучшим конфигурациям при существенном ухудшении стоимости текущего поколения, что происходит, в связи с необходимостью коррекций, достаточно часто.

Эволюционный процесс часто приводит к такой ситуации, когда разнообразие внутри поколения мало, а лучшая оценка генотипа внутри поколения не меняется – попадание в локальный максимум функции оценки. В немодифицированном генетическом алгоритме эта проблема решается за счет мутации, однако из-за большой удаленности локальных максимумов (в терминах расстояния Хэмминга) и постоянных коррекций такое решение может быть недостаточно эффективным. Одним из способов покинуть локальный максимум является перезапуск алгоритма. При таком перезапуске текущее поколение заменяется новым, созданным с использованием алгоритма начального приближения, после чего эволюционный процесс запускается вновь. Текущий пул лучших конфигураций при этом сохраняется отдельно – в глобальном пуле, который не взаимодействует с эволюционным процессом. При множественных перезапусках глобальный пул хранит информацию о лучших решениях за все время работы генетического алгоритма, что позволяет при необходимости получить неоптимальные решения, близкие к искомому.

2 Практическая часть

2.1 Наборы тестов

Для оценки эффективности работы алгоритма было проведено его тестирование на различных наборах тестов. Первый набор тестов взят из книги Петерсена(см. [5]) и содержит 7 задач. Условия этих задач(см таблицу 1) позволяют проверить эффективность простейшей версии генетического алгоритма без модификаций(см. 1.2.3) и оценить её эффективность.

№ задачи	Размерность	Количество предметов
1	6	10
2	10	10
3	15	10
4	20	10
5	28	10
6	39	5
7	50	5

Таблица 1: Параметры первого набора тестов

Второй набор тестов взят из статьи Чу (см. [7]) и содержит в себе 30 задач с одинаковыми параметрами: размерность рюкзака равна 5, рассматривается 100 различных предметов. Для каждого набора тестов известно лучшее решение, эффективность алгоритма оценивалась по скорости поиска решения и числу итераций. Этот набор тестов использовался со вс

Для тестирования использовался компьютер с следующей аппаратно-программной базой:

- Процессор: Intel®Core™i5-3230M CPU @ 2.60GHz (4 CPUs), 2.6GHz
- Количество оперативной памяти: 6144MB RAM
- Материнская плата: SAMSUNG NP370-R5E
- Операционная система: Windows 10 Домашняя для одного языка 64-bit (10.0, Build 15063) (15063.rs2_release.170317-1834)

Для сборки использовалась программная платформа .NET Framework 4.5.2.

2.2 Первый набор тестов

Рассмотрим результаты работы алгоритма без модификаций на первом наборе тестов. Экспериментальным образом были подобраны количество конфигураций в пуле(10) и процент мутирующих особей(20)

2.2.1 Мутация в одной позиции

Данная мутация показала себя эффективной с точки зрения решения задачи - для всех возможных вариантов скрещивания во всех тестах найден оптимальный результат(см таблицы 2, 3, 4) При сравнении результатов скрещивания можно заметить, что алгоритм побитового скрещивания в большинстве случаев несколько уступает в скорости другим алгоритмам.

№ теста	Итерации	Время, мс
1	6	8
2	114340	2694
3	712	18
4	4022	92
5	3049	97
6	400974	13481
7	9343432	396644

Таблица 2: Одноточечная мутация, одноточечное скрещивание

№ теста	Итерации	Время, мс
1	21	9
2	70930	1697
3	901	22
4	2092	67
5	2672	95
6	23261	8181
7	4770456	199461

Таблица 3: Одноточечная мутация, двуточечное скрещивание

№ теста	Итерации	Время. мс
1	3	9
2	182633	3301
3	1455	49
4	7869	219
5	2198	74
6	289379	100133
7	348826	139403

Таблица 4: Одноточечная мутация, побитовое скрещивание

2.2.2 Инверсионная мутация

Инверсионная мутация в контексте данной задачи оказалась менее эффективной, для тестов 5-7 решения не были найдены за 20 минут для всех вариантов скрещивания (см табл 5, 6, 7). Неэффективность такого варианта мутации может быть обусловлена применением алгоритма коррекции после мутации, что сводит на нет какой-либо положительный эффект от её применения. Также можно отметить существенно более высокую скорость решения в сравнении с алгоритмом одноточечной мутации для теста 2.

№ теста	Итерации	Время. мс
1	3	6
2	10	2
3	58	2
4	9538	560
5	-	-
6	-	-
7	-	-

Таблица 5: Инверсионная мутация, одноточечное скрещивание

№ теста	Итерации	Время. мс
1	2	10
2	16	1
3	356	19
4	7161	422
5	-	-
6	-	-
7	-	-

Таблица 6: Инверсионная мутация, двуточечное скрещивание

№ теста	Итерации	Время. мс
1	10	10
2	23	1
3	1409	52
4	246001	12189
5	-	-
6	-	-
7	-	-

Таблица 7: Инверсионная мутация, побитовое скрещивание

2.2.3 Выводы по первому тестированию

Исходя из приведенных тестов, можно считать подтвержденной принципиальную возможность реализованного алгоритма быть использованным для решения задачи о многомерном рюкзаке. Наиболее эффективным себя показало двуточечное скрещивание, что может быть обусловлено большим разнообразием порождаемых им особей. Инверсионная мутация обеспечила более высокую скорость в сравнении с одноточечной, но показала себя неэффективной на больших наборах тестов, что может быть обусловлено нарушением функции наследственности при её использовании.

2.3 Второй набор тестов

Для проверки эффективности второго набора тестов были выбраны параметры запуска алгоритма, указанные в таблице 8. Выбор алгоритмов мутации и скрещивания был обусловлен результатами предыдущего тестирования, максимальное число итераций было подобрано таким образом, чтобы алгоритм завершал работу за 15 минут. Остальные параметры были подобраны экспериментально.

Вид скрещивания	Двуточечное
Вид мутации	Одноточечная
Размер пула генотипов	10
Процент мутирующих особей	20%
Максимальное число итераций	2000000
Величина отклонения для сброса алгоритма	0.01
Количество итераций до полного перезапуска	200000

Таблица 8: Параметры запуска второго набора тестов

Максимальное число итераций было введено с целью ограничить время работы алгоритма. В случае, если лучшее решение не найдено, программа выдает на печать расхождение лучшего из найденных решений с текущим.

Рассмотрим результаты работы программы на втором наборе тестов. Для успешных решений среднее время составило 315 с, среднее число итераций - 652 007 (см. табл. [BigResults1])

№ теста	Количество итераций	Время, с
1	-	888
2	197915	44
3	-	915
4	-	812
5	-	860
6	-	970
7	1253381	619
8	83207	39
9	262645	255
10	-	963
11	189635	83
12	-	912
13	-	912
14	-	825
15	-	881
16	942021	484
17	-	864
18	4386	5
19	-	880
20	-	865
21	-	734
22	1735194	752
23	-	779
24	845324	428
25	-	812
26	908217	454
27	594220	281
28	36434	35
29	-	895
30	1423522	625

Таблица 9: Временная эффективность алгоритма

В 43.3% случаев генетический алгоритм решил задачу и во всех случаях расхождение найденного и действительного максимумов расходится не более чем на 0,0021 с оптимальным результатом в относительных значениях(см [BigResults2]). Такой результат можно считать успешным и применимым в практической деятельности.

№ теста	Относительное расхождение с максимумом
1	0,0021
2	0
3	0,0005
4	0.0022
5	0.0018
6	0.0014
7	0
8	0
9	0
10	0,0031
11	0
12	0,0020
13	0,0008
14	0,0001
15	0,0017
16	0
17	0,0006
18	0
19	0,0018
20	0,0005
21	0,0016
22	0
23	0,0008
24	0
25	0,0007
26	0
27	0
28	0
29	0,0006
30	0

Таблица 10: Эффективность в достижении максимума задачи

ЗАКЛЮЧЕНИЕ

В ходе работы были выполнены все поставленные задачи. Была создана программа, реализующая генетический алгоритм на языке C# с требуемыми для решения задачи модификациями и улучшениями. Было проведено тестирование, показавшее высокую эффективность программы на различных наборах тестов. Для малых наборов предметов (50 предметов и менее) оптимальное решение было достигнуто в 100. Для больших наборов (100 предметов) оптимум был получен в 43,3. При этом среднее время поиска решения составляет 5 минут 25 секунд. Таким образом, алгоритм может эффективно применяться в решении практических проблем.

Список

ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Gavish B., Pirkul H.* Allocation of Databases and Processors in a Distributed Computing System // Management of Distributed Data Processing. – 1982. – № 31. – С. 215–231.
2. *Shih W.* A Branch and Bound Method for the Multiconstraint Zero-One Knapsack Problem // Journal of the Operational Research Society. – 1979. – № 30. – С. 369–378.
3. *Gilmore P., Gomory R.* The Theory and Computation of Knapsack Functions // Operations Research. – 1966. – № 14.
4. *Mathews G.* On the partition of numbers // Proceedings of the London Mathematical Society. – 1897. – № 28. – С. 486–490.
5. *C.C.Petersen* Computational experience with variants of the Balas algorithm applied to the selection of R&D projects // Management Science. – 1967. – 13(9). – С. 736–750.
6. *Alex F.* Computer Models in Genetics. – McGraw-Hill, 1970.
7. *P.C.Chu, J.E.Beasley* A genetic algorithm for the multidimensional knapsack problem // Journal of Heuristics. – 1998. – Т. 4. – С. 63–86.

ПРИЛОЖЕНИЯ

A Исходный код программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GAMultidimKnapsack
{
    class KnapsackConfig:IEquatable<KnapsackConfig>
    {
        private int[] CurrentConfiguration;

        public KnapsackConfig(int elementsAmount)
        {
            CurrentConfiguration = Enumerable.Repeat(-1, elementsAmount).ToArray();
        }

        public KnapsackConfig(int[] initConfig)
        {
            CurrentConfiguration = initConfig;
        }

        public KnapsackConfig(KnapsackConfig conf)//memberwise clone coud replace it or not
        {
            try
            {
                this.CurrentConfiguration = new int[conf.Length()];
                for (int i = 0; i < conf.Length(); i++)
                {
                    this.CurrentConfiguration[i] = conf.valueAt(i);
                }
            }
            catch (NullReferenceException ex)
            {
                Console.WriteLine("Empty configuration");
                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)
        {
            Console.WriteLine("Empty configuration");
            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;
    }

    public override bool Equals(object obj)
    {
        if (obj is KnapsackConfig)
            return Equals((KnapsackConfig) obj);
        return false;
    }

    public override int GetHashCode()
    {
        return ToString().GetHashCode();
    }

    public override string ToString()
    {
        return String.Join(",", CurrentConfiguration);
    }
}

using System;
using System.Collections.Generic;
using System.Linq;

```

```

using System.Text;

namespace GAMultidimKnapsack
{
    class GeneticalAlgorithm
    {
        private static int itemsAmount, dimensions;
        private static double[,] itemsSet; //amount of items*their dimensions
        private static double[] restrictions;
        private double[] itemsCosts;

        private int configsInPoolAmount;
        private int bestConfigsAmount;
        private KnapsackConfig[] configsPool;
        private KnapsackConfig[] currentBestConfigs; //however, this pool is resetted
            over and over again. We should create a permanent pool for practical use.
            Possibliy
        private KnapsackConfig[] bestConfigsAllTime;
        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 = configsInPoolAmount;

            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);
    }
    emptyBestConfigs(ref currentBestConfigs);
    emptyBestConfigs(ref bestConfigsAllTime);

    //int[] emptyConfig = (new int[itemsAmount]).Select(x => 0).ToArray();
    ;
    //bestConfigs = (new KnapsackConfig[bestConfigsAmount]).Select(x =>
        new KnapsackConfig(emptyConfig)).ToArray(); //HACK
}
catch (Exception ex)
{
    Console.WriteLine("Bugs in initialization");
    return;
}
}

void emptyBestConfigs(ref KnapsackConfig[] targetConfig)
{
    int[] emptyConfig = (new int[itemsAmount]).Select(x => 0).ToArray();
    targetConfig = (new KnapsackConfig[bestConfigsAmount]).Select(x => new
        KnapsackConfig(emptyConfig)).ToArray();
}

public void RestartAlgorithm(double flushPercent)
{
    try
    {
        IndexOutOfRangeException ex = new IndexOutOfRangeException(); //TODO:
        write exceptions class;
        if (flushPercent < 0 || flushPercent > 1)
            throw ex;
        else
        {
            //int[] emptyConfig = (new int[itemsAmount]).Select(x => 0).
            ToArray();
            //bestConfigs = (new KnapsackConfig[bestConfigsAmount]).Select(x
            => new KnapsackConfig(emptyConfig)).ToArray(); //HACK
            emptyBestConfigs(ref currentBestConfigs);
            int startFlushpoint = rand.Next(0, itemsAmount), endPoint,
            itemsToFlush = (int)(itemsAmount * flushPercent);
            if (startFlushpoint + itemsToFlush >= itemsAmount)
            {
                endPoint = itemsToFlush - (itemsAmount - startFlushpoint);
                foreach (var conf in configsPool)
                {
                    for (int i = startFlushpoint; i < itemsAmount; i++)
                        conf.SetValueToPassive(i);
                    for (int i = 0; i < endPoint; i++)
                        conf.SetValueToPassive(i);
                }
            }
            else

```

```

        {
            endPoint = startFlushpoint + itemsToFlush;
            foreach (var conf in configsPool)
            {
                for (int i = startFlushpoint; i < endPoint; i++)
                    conf.SetValueToPassive(i);
            }
        }
        //Сбрасываем целиком BestConfigs
        //Выносим flushpercent для текущих конфигураций 0.
        Конфигурации корректны .
    }
}
catch (Exception ex)
{
    Console.WriteLine("Flush_percent_is_in_[0;1]");
}
}
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();
    }
    for (int i = 0; i < configsInPoolAmount; i++)
    {
        configsPool[i] = activeMutation(configsPool[i], rand);
    }
    KnapsackConfig[] CrossoverPool = new KnapsackConfig[configsInPoolAmount *
        2 - 2]; //not very well, if i want to customize Crossover ,but works
    for (int 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))
        .Take(Convert.ToInt32(configsInPoolAmount))
        .ToArray();
    configsPool = tempConfigs;

    var tuningCoeff = 0.01;
    updateConfigs(ref currentBestConfigs, tuningCoeff);
    updateConfigs(ref bestConfigsAllTime, tuningCoeff);
}

private void updateConfigs(ref KnapsackConfig[] currentPool, double
    tuningCoeff)
{
    if (currentPool.Length >= configsPool.Length &&
        GetKnapsackCost(currentPool[0]) * (1 - tuningCoeff) > GetKnapsackCost(configsPool
            [0]))
    {
        for (int i = 0; i < configsInPoolAmount; i++)
            configsPool[i] = new KnapsackConfig(currentPool[i]);
    }
}

```

```

        return;
    }
    currentPool = currentPool
        .Concat(configsPool)
        .OrderByDescending(config => GetKnapsackCost(config))
        .Distinct()
        .Take(bestConfigsAmount)
        .ToArray();
}
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;
    var iterationsToResque = 100000;
    while (mutatedSack.Equals(sack) && count < iterationsToResque) //TODO -
        not mutate empty sack
    {
        mutatedSack.swapValue(mutationPosition);
        if (!IsValid(mutatedSack)) //somehow unrealistic
        {
            mutatedSack.swapValue(mutationPosition);
            mutationPosition = rand.Next(itemsAmount);
        }
        count++;
    }
    if (count == iterationsToResque)
    {
        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;
            }
        }
        //Amount of items is much bigger than number of dimensions, so we can
        //do checks on every turn.
    }
    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)
{
    for (var i = 0; i < sack.Length() && !IsValid(sack); i++)
    {
        sack.setValueToPassive(i);
    }
    return sack;
}

//methods for current active pool
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;
    }

    public List<double> GetBestConfigsCosts()
    {
        return currentBestConfigs.Select(x => GetKnapsackCost(x)).ToList();
    }

    //methods for best all-time configs
    public double GetAbsoluteMaximalCostAllTime()
    {
        return GetKnapsackCost(bestConfigsAllTime[0]); //does it work is uncertain
    }
}

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;

namespace GAMultidimKnapsack
{
    public partial class SetPartition
    {
        static Random rand = new Random();
        static ConcurrentQueue<double> averageValuations = new ConcurrentQueue<double>();
        static ConcurrentQueue<double> maxValuations = new ConcurrentQueue<double>();
        static ConcurrentQueue<double> ages = new ConcurrentQueue<double>();
    }
}

```

```

static List<string> algorithmWithRestart(int itemsAmount, int dimensions,
    double maxCost, double[] restrictions, double[] costs, double[,] itemsSet)
{
    int ConfigsAmount = 10, restartTime = 2*100000, currentMaxValueLiveLength
        = 0;
    double PrevCost = 0;
    double mutationPercent = 0.20; //FROM 0 TO 1
    GeneticalAlgorithm ga = new GeneticalAlgorithm(itemsAmount, dimensions,
        restrictions, costs, itemsSet, ConfigsAmount, GeneticalAlgorithm.
        TwoPointCrossover, GeneticalAlgorithm.SinglePointMutation,
        mutationPercent);
    int iterationNumber = 0, endIteration = 2*1000000;
    List<double> resetPoints = new List<double>();
    var workTime = new Stopwatch();
    workTime.Start();

    // string logFileName = "plotwithreset.txt";
    // List<string> values = new List<string>();
    while (ga.GetAbsoluteMaximalCostAllTime() != maxCost && iterationNumber <
        endIteration)
    {
        var watch = new Stopwatch();
        watch.Start();
        ga.MakeIteration();
        iterationNumber++;
        double tmp = ga.GetBestConfigsCosts()[0];
        // values.Add(tmp.ToString());
        if (tmp != PrevCost)
        {
            PrevCost = tmp;
            currentMaxValueLiveLength = 0;
        }
        else
        {
            currentMaxValueLiveLength++;
        }
        /*
        if (iterationNumber % 10000 == 0) Отрисовка //
        {
            Console.Write(iterationNumber + " "); // delta with avg is " + (
                maxCost - ga.GetAbsoluteAverageKnapsackCost()) + "\n delta
                with max is " + (maxCost - ga.GetAbsoluteMaximalKnapsackCost()
                ));
            var bestCosts = ga.GetBestConfigsCosts();
            Console.WriteLine("Top 3 of the best configs pool are {0}, {1},
                {2}, {3}, {4}",
                (maxCost - bestCosts[0]),
                (maxCost - bestCosts[1]),
                (maxCost - bestCosts[2]),
                (maxCost - bestCosts[3]),
                (maxCost - bestCosts[4]));
        }
        */
        if (currentMaxValueLiveLength == restartTime)
        {
            var restartPercent = 0.4;
            resetPoints.Add(maxCost - ga.GetBestConfigsCosts()[0]);
            ga.RestartAlgorithm(restartPercent);
            PrevCost = 0;
            currentMaxValueLiveLength = 0;
        }
    }
}

```



```

        // Console.WriteLine("Restart");
    }
    watch.Stop();
}
workTime.Stop();
Console.WriteLine(workTime.Elapsed.TotalSeconds.ToString());
//Can use resetPoints for Something.
//File.WriteAllLines(logFileName, values);
if (ga.GetAbsoluteMaximalCostAllTime() == maxCost) //problem solved
    return transformResults(iterationNumber, workTime);

    else return transformResults(maxCost - ga.GetAbsoluteMaximalCostAllTime()
        , workTime);
}

static List<string> transformResults(int finishingIteration, Stopwatch
workTime)
{
    List<string> results = new List<string>();
    results.Add("Succeeded_in_" + finishingIteration + "_iterations_" +
        workTime.Elapsed.Seconds.ToString() + "_seconds");
    //string tmpString = "";
    //foreach (var x in resetPoints)
    //    tmpString += x.ToString() + ",";
    //results.Add(tmpString);
    //results.Add(resetPoints.Count.ToString());
    return results;
}

static List<string> transformResults(double difference, Stopwatch workTime)
{
    List<string> results = new List<string>();
    results.Add("Unsucceeded!_It_took_" + workTime.Elapsed.Seconds.ToString()
        + "_seconds_and_the_difference_between_best_of_found_configurations_
        and_known_maximum_is_" + difference.ToString());
    return results;
}

static void WriteResutls(int experimentNumber, List<string> results, string
filename)
{
    var fs = new FileStream(filename, FileMode.Append);
    using (StreamWriter sw = new StreamWriter(fs))
    {
        sw.WriteLine("Experiment_" + experimentNumber.ToString());
        foreach (var result in results)
            sw.WriteLine(result);
        sw.WriteLine();
    }
}

static void ProcessTestSet(string inputFileData, string inputFileResults) //
WORK WITH IT!
{
    using (StreamReader dataReader = new StreamReader(inputFileData))
    {
        string[] resultsArray = File.ReadAllLines(inputFileResults);
        var resultsStringNumber = 12;
        int experimentsAmount = Convert.ToInt32(dataReader.ReadLine());
    }
}

```

```

for (int experimentNumber = 0; experimentNumber < experimentsAmount;
    experimentNumber++, resultsStringNumber++)
{
    string[] initializationSequence;
    string firstString = dataReader.ReadLine();
    if (firstString.Trim() == "")
        initializationSequence = dataReader.ReadLine().Split("\u000A".
            ToCharArray(), StringSplitOptions.RemoveEmptyEntries);
    else initializationSequence = firstString.Split("\u000A".ToCharArray(),
        StringSplitOptions.RemoveEmptyEntries); ;
    int itemsAmount = Convert.ToInt32(initializationSequence[0]),
    dimensions = Convert.ToInt32(initializationSequence[1]);
    double maxCost = Convert.ToDouble(resultsArray[
        resultsStringNumber].Substring(25)); //Convert.ToDouble(temp);
    List<double> tempCosts = new List<double>();
    while (tempCosts.Count() != itemsAmount)
        tempCosts.AddRange(dataReader
            .ReadLine()
            .Split("\u000A".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 = dataReader.ReadLine()
                .Split("\u000A".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(dataReader
            .ReadLine()
            .Split("\u000A".ToCharArray(), StringSplitOptions.
                RemoveEmptyEntries)
            .Select(x => Convert.ToDouble(x))
            .ToList());
    double[] restrictions = tempRestrictions.ToArray();
    //some silly work with reading from file.
    //написать перегрузку выбора алгоритма
    List<string> resultsList = algorithmWithRestart(itemsAmount,
        dimensions, maxCost, restrictions, costs, itemsSet);
    WriteResutls(experimentNumber, resultsList, "results.txt");

    Thread.Sleep(3000);
    maxValuations.Enqueue(0);
    averageValuations.Enqueue(0);
}
}

```

```

    }

    static void Main()
    {
        //new Thread(TestAlgorithm) { IsBackground = true }.Start();
        //new Thread(() => ProcessTestSet(@"C:\Users\black_000\Source\Repos\
        GeneticKnapsack\GAMultidimKnapsack\3.txt", @"C:\Users\black_000\Source
        \Repos\GeneticKnapsack\GAMultidimKnapsack\_res.txt")) { IsBackground =
        true }.Start();
        ProcessTestSet(@"C:\Users\black_000\Source\Repos\GeneticKnapsack\
        GAMultidimKnapsack\3.txt", @"C:\Users\black_000\Source\Repos\
        GeneticKnapsack\GAMultidimKnapsack\_res.txt");
    }
}

using GAMultidimKnapsack;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace GAMultidimKnapsack
{
    public partial class SetPartition
    {
        static void Algorithm(int itemsAmount, int dimensions, double maxCost, double
            [] restrictions, double[] costs, double[,] itemsSet)//version w/o restarts
        {
            int ConfigsAmount = 10;
            double mutationPercent = 0.20;//FROM 0 TO 1
            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();
                ga.MakeIteration();
                iterationNumber++;
                if (iterationNumber % 10000 == 0)
                {
                    Console.WriteLine(iterationNumber + ")_delta_with_avg_is_" + (
                        maxCost - ga.GetAbsoluteAverageKnapsackCost()) + "\n_delta_
                        with_max_is_" + (maxCost - ga.GetAbsoluteMaximalKnapsackCost()
                    ));
                    var bestCosts = ga.GetBestConfigsCosts();
                    Console.WriteLine("Top_3_of_the_best_configs_pool_are_{0},{1},{2}",
                        (maxCost - bestCosts[0]),
                        (maxCost - bestCosts[1]),

```

```

        (maxCost - bestCosts[2]));
    }
    // watch.Stop();
}
Console.WriteLine("Finished in {0}", iterationNumber);
Console.ReadKey();
}

static void TestAlgorithm()//First 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 = 6;
    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();
    }
}

static void ProcessTestSet(string file) //worked with first files. proven its
    efficiency.TODO - check, if that true
{
    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 List<string> multiThreadAlgorithms(int itemsAmount, int dimensions,
    double maxCost, double[] restrictions, double[] costs, double[,] itemsSet)
    //launches multiple algorithms with different start approximations. Does
    not work
{
    int ConfigsAmount = 10, algorithmsNumber = 3;
    double mutationPercent = 0.20;

    GeneticalAlgorithm[] gas = new GeneticalAlgorithm[algorithmsNumber];
    for (int i = 0; i < algorithmsNumber; i++)//пока предположим,
        что первые приближения различны
        gas[i] = new GeneticalAlgorithm(itemsAmount, dimensions, restrictions
            , costs, itemsSet, ConfigsAmount, GeneticalAlgorithm.

```

```

                TwoPointCrossover, GeneticalAlgorithm.SinglePointMutation,
                mutationPercent);
    int iterationNumber = 1;
    var controlWatch = new Stopwatch();
    controlWatch.Start();
    while (!gas.Select(x => x.GetAbsoluteMaximalKnapsackCost()).ToArray().
        Contains(maxCost))
    {
        Parallel.ForEach(gas, ga =>
            //foreach(GeneticalAlgorithm ga in gas)
            {
                ga.MakeIteration();
            });

        iterationNumber++;
        if (iterationNumber%10==0)
            Console.WriteLine(iterationNumber);
    }

    controlWatch.Stop();
    List<string> t=new List<string>();
    t.Add(iterationNumber.ToString());
    return t;
}

// transformResults
//     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");
// }

}

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
{
    //this whole file processes 1st file with fixed settings.
    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("\n".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 Main2(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();
    //}
}
}

```