1. **Постановка задачи о ранце**

**Описание**

Задача о ранце (рюкзаке) — одна из NP-полных задач комбинаторной оптимизации. Своё название задача получила от оптимизационной задачи укладки как можно большего числа ценных вещей в рюкзак при условии, что общий объём (или вес) всех предметов, способных поместиться в рюкзак, ограничен. Задача о рюкзаке является актуальной и достаточно востребованной с точки зрения ее приложения в реальной жизни. Задача о загрузке (о рюкзаке) и её модификации часто возникают в экономике, прикладной математике, криптографии, генетике и логистике для нахождения оптимальной загрузки транспорта (самолёта, поезда, трюма корабля) или склада.

**Классическая постановка задачи**

**Словесная:**

Пусть имеется набор предметов, каждый из которых имеет два параметра – вес и ценность. Имеется рюкзак c некоторым заданным значением вместимости. Задача заключается в том, чтобы собрать рюкзак с максимальной ценностью предметов внутри, соблюдая при этом весовое ограничение рюкзака.

**Математическая:**

Пусть имеется n предметов. Для каждого i-го предмета задан его вес > 0 и стоимость (ценность) > 0, i =1,.., n., Задано ограничение на максимальный вес рюкзака ‒ P . Каждый может принимать только одно из двух значений: xi=1 , если i-й предмет упаковывают в рюкзак, или , в противном случае. Требуется выбрать из заданного множества предметов набор с максимальной суммарной стоимостью при одновременном соблюдении ограничения на суммарный вес найденного набора .

**Варианты постановки задачи о ранце:**

Существует множество разновидностей задачи о ранце, отличия заключаются в условиях, наложенных на рюкзак, предметы или их выбор.

1. **Рюкзак 0-1** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) 0-1 Knapsack Problem): не более одного экземпляра каждого предмета
2. **Ограниченный рюкзак** (англ. Bounded Knapsack Problem) - обобщение классической задачи, когда любой предмет может быть взят некоторое количество раз.
3. **Неограниченный рюкзак** (целочисленный рюкзак)([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Unbounded Knapsack Problem (integer knapsack)): произвольное количество экземпляров каждого предмета.
4. **Непрерывный рюкзак** (англ. Continuous knapsack problem) - вариант задачи, в котором возможно брать любою дробную часть от предмета, при этом удельная стоимость сохраняется.
5. **Рюкзак с мультивыбором** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Multiple-choice Knapsack Problem)
6. **Мультипликативный рюкзак** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Multiple Knapsack Problem): есть несколько рюкзаков, каждый со своим максимальным весом. Каждый предмет можно положить в любой рюкзак или оставить.
7. **Многомерный рюкзак** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Multy-dimensional knapsack problem) : вместо веса дано несколько разных ресурсов (например, вес, объём и время укладки). Каждый предмет тратит заданное количество каждого ресурса. Надо выбрать подмножество предметов так, чтобы общие затраты каждого ресурса не превышали максимума по этому ресурсу, и при этом общая ценность предметов была максимальна.

**Области применения задачи о ранце:**

Задача о ранце имеет применение в различных областях знаний: в математике, информатике и на стыке этих наук — в криптографии. В вычислительной лингвистике в одной из работ предложена формулировка задачи [автоматического реферирования текстов](https://en.wikipedia.org/wiki/Automatic_summarization) (англ.)[русск.](https://ru.wikipedia.org/w/index.php?title=%D0%90%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D1%80%D0%B5%D1%84%D0%B5%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D1%82%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2&action=edit&redlink=1), вырожденный (более простой) случай которой соответствует постановке задачи о ранце.

### Изучение в математике.

Одно из первых упоминаний о задаче о ранце можно найти в статье [Джорджа Балларда Мэтьюса](https://en.wikipedia.org/wiki/George_Ballard_Mathews) (англ.)[русск.](https://ru.wikipedia.org/w/index.php?title=%D0%9C%D1%8D%D1%82%D1%8C%D1%8E%D1%81,_%D0%94%D0%B6%D0%BE%D1%80%D0%B4%D0%B6_%D0%91%D0%B0%D0%BB%D0%BB%D0%B0%D1%80%D0%B4&action=edit&redlink=1), датированной 1897 годом. Интенсивное изучение данной проблемы началось после публикации [Д. Б Данцигом](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D1%86%D0%B8%D0%B3,_%D0%94%D0%B6%D0%BE%D1%80%D0%B4%D0%B6) в 1957 году книги «[англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Discrete Variable Extremum Problem», особенно в 70-90-е годы 20-го века, как теоретиками, так и практиками. Во многом, данный интерес вызван достаточно простой формулировкой задачи, большим числом её разновидностей и свойств и в то же время сложностью их решения. В 1972 году данная задача вошла в список [К. Мэннига](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D0%BF,_%D0%A0%D0%B8%D1%87%D0%B0%D1%80%D0%B4_%D0%9C%D1%8D%D0%BD%D0%BD%D0%B8%D0%BD%D0%B3) [NP-полных задач](https://ru.wikipedia.org/wiki/NP-%D0%BF%D0%BE%D0%BB%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0) (статья [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) «Reducibility Among Combinatorial Problems»).

С практической точки зрения задача о рюкзаке может служить моделью для большого числа промышленных ситуаций:

* Размещение грузов в помещении минимального объёма.
* Раскройка ткани — для заданного куска материала получить максимальное число выкроек определенной формы.
* Расчет оптимальных капиталовложений.

С задачей о ранце сталкивается любой человек, собирающий рюкзак.

### Изучение в криптографии

Проблема рюкзака лежит в основе первого алгоритма [асимметричного шифрования](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC) (шифрования с открытым ключом). Идея [криптографии](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F) с открытыми ключами была выдвинута [Уитфилдом Диффи](https://ru.wikipedia.org/wiki/%D0%A3%D0%B8%D1%82%D1%84%D0%B8%D0%BB%D0%B4_%D0%94%D0%B8%D1%84%D1%84%D0%B8), [Мартином Хеллманом](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D0%BB%D0%BB%D0%BC%D0%B0%D0%BD,_%D0%9C%D0%B0%D1%80%D1%82%D0%B8%D0%BD) и независимо — [Ральфом Мерклом](https://ru.wikipedia.org/w/index.php?title=Ralph_Merkle&action=edit&redlink=1) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Ralph Merkle). Впервые она была представлена Диффи и Хеллманом на Национальной компьютерной конференции ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) National Computer Conference). Новизна по отношению к [симметричным криптосистемам](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D0%B5_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B) заключалась в использовании парных ключей — секретного ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) private key, secret key, SK) и открытого ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) public key, PK), создаваемых пользователем. Из названия понятно, что секретный ключ пользователь должен скрывать, а открытый может быть общедоступным. Открытый ключ нужен для шифрования, а секретный для расшифровки. Часто из секретного ключа получают открытый ключ.

**Криптосистема Меркла-Хеллмана** — первый, основанный на задаче о ранце, алгоритм для обобщённого шифрования с [открытым ключом](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC). Разработан Ральфом Мерклом и Мартином Хеллманом в 1978 году. Был опубликован одностадийный ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) singly-iterated) и мультистадийный варианты ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) multiply-iterated). Алгоритм мог быть использован только для шифрования, но [Ади Шамир](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%BC%D0%B8%D1%80,_%D0%90%D0%B4%D0%B8) адаптировал его для использования в [цифровых подписях](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%86%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C).

В дальнейшем было предложено как множество модификаций криптосистемы Меркла-Хеллмана, так и совершенно новых криптосистем на основе задачи о ранце. Среди них:

1. Рюкзак Грэм-Шамира
2. Рюкзак Гудмана-Макколи
3. Рюкзак Накаше-Штерна
4. Рюкзак Шора-Ривеста

#### Шифрование с помощью задачи о рюкзаке:

Сообщение шифруется как решение набора задач о ранце.

Рюкзачным вектором  A = (a_1,...,a_n) назовём упорядоченный набор из n предметов.

Для шифрования [открытого текста](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%B9_%D1%82%D0%B5%D0%BA%D1%81%D1%82) в двоичном представлении его разбивают на блоки длины n (например, (1 1 1 0 0) соответствует 5-ти предметам в рюкзаке). Считается, что единица указывает на наличие предмета в рюкзаке, а ноль на его отсутствие.

**Пример** шифротекста, полученного по данному алгоритму.

Пусть задан рюкзачный вектор Α = (3 4 6 7 10 11) с длинной n = 6.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| открытый текст | 1 1 1 1 1 0 | 0 0 1 1 0 0 | 0 0 0 0 0 0 | 0 0 0 0 0 1 |
| вещи в рюкзаке | 3 4 6 7 10 11 | 3 4 6 7 10 11 | 3 4 6 7 10 11 | 3 4 6 7 10 11 |
| шифротекст | 3 + 4 + 6 + 7 + 10 = 30 | 6 + 7 = 13 | 0 | 11 |

Для заданного Α - все криптосистемы - есть числа, не превышающие 41, суммарный вес всех предметов в рюкзачном векторе. Для каждого исходного текста существует единственный криптотекст. В указанном примере можно получить одинаковый шифротекст для векторов 100010 и 001100. Ошибка?

Верно, ошибка. Для шифрования необходим сверхвозрастающий рюкзак, то есть элементы упорядоченного рюкзачного вектора должны являться [сверхвозрастающей последовательностью.](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%B5%D1%80%D1%85%D0%B2%D0%BE%D0%B7%D1%80%D0%B0%D1%81%D1%82%D0%B0%D1%8E%D1%89%D0%B0%D1%8F_%D0%BF%D0%BE%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C)

### Изучение в информатике

Как было сказано выше, задача о ранце относится к классу NP-полных, для неё нет полиномиального алгоритма, решающего её за разумное время. Поэтому при решении задачи о ранце всегда нужно выбирать между точными алгоритмами, которые не применимы для «больших» рюкзаков, и приближенными, которые работают быстро, но не обеспечивают оптимального решения задачи. Естественно, создание быстрого и достаточно точного алгоритма представляет большой интерес.

## 

## NP-трудность и вычислительная сложность

**Задача о ранце – NP – трудная задача.**

**Доказательство:**

Рассмотрим специальную задачу о Ранце:

(\*)

Докажем, используя утверждение о том, что задача о камнях - NP - полная.

Задача о камнях – задача о разбиении кучи камней на две, вес которых будет одинаков.

Если оптимум в задаче о Ранце строго меньше < P, то задача о камнях не имеет решение. Если в (\*) оптимум <P , то => , если (\*) оптимум >P , то => . Если

L->P за полиномиальное время, то есть задача о Ранце - NP – трудная.

**Задачу о рюкзаке можно решить несколькими способами:**

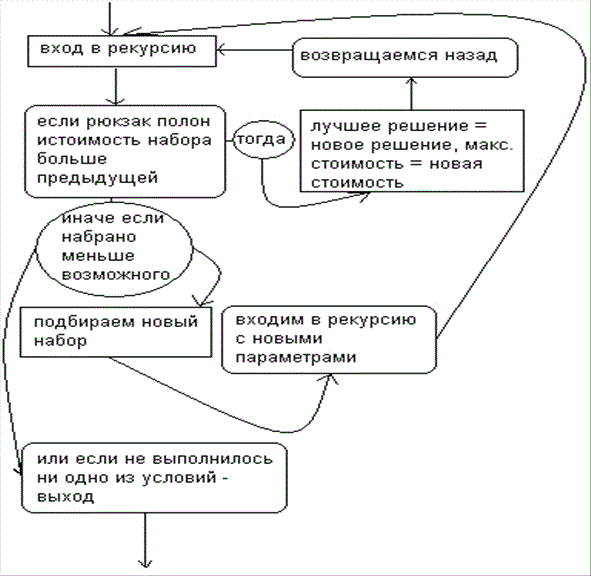
* Перебирать все подмножества набора из N предметов. Сложность такого решения  O(2N ).
* Методом [Meet-in-the-middle](http://neerc.ifmo.ru/wiki/index.php?title=Meet-in-the-middle). Сложность решения  O(2N/2 x N).
* Метод динамического программирования. Сложность -  O(N x P).
* Жадным алгоритмом. Сложность - O(N\*Log(N)).
* И другие( о них далее).

**2. Известные алгоритмы решения**

**1) Точные алгоритмы**

**1.1) Полный перебор**

Пусть в рюкзак загружаются предметы  N  разных типов. Рассмотрим задачу, когда количество предметов каждого типа не ограничено. Нужно определить максимальную стоимость груза, вес которого равен  P . Для получения решения алгоритмом [полного перебора](https://ru.wikipedia.org/wiki/NP-%D0%BF%D0%BE%D0%BB%D0%BD%D0%B0%D1%8F_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0) осуществляется перебор всех вариантов загрузки рюкзака.

Временная сложность алгоритма  O(N!) , т.е он работоспособен для небольших значений  N . С ростом  N  задача становится неразрешимой данным методом за приемлемое время.



Дерево полного перебора

На рисунке показано четырёхуровневое дерево перебора. Корень дерева соответствует нулевому весу (рюкзак пуст), в кружках показан вес предмета. Первый предмет возможно выбрать четырьмя способами, второй тремя и т. д.

**1.2) Метод ветвей и границ**



Дерево, упрощенное методом ветвей и границ

[Метод ветвей и границ](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B2%D0%B5%D1%82%D0%B2%D0%B5%D0%B9_%D0%B8_%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%86) является вариацией метода полного перебора с той разницей, что мы сразу исключаем заведомо неоптимальные решения.

Пусть есть оптимальное решение  R . Попытаемся его улучшить, рассмотрев решение на другой ветви. Если на рассматриваемой в данной момент ветви решение становится хуже (с какого-то шага), чем  R , то прекращаем его исследование и выбираем другую ветвь дерева.

Пусть для предыдущего четырёхуровневого дерева есть ограничение P=5. Тогда, применяя метод ветвей и границ, можно сократить количество вариантов для перебора с 24-х до 8-ми. Однако метод ветвей и границ работает не для всех наборов данных. Можно привести примеры, в которых время выполнения будет таким же, как и для простого перебора.

##### Применение метода ветвей и границ:

При использовании метода ветвей и границ строится сеть. По оси  X  откладываем количество предметов, по оси  Y  — их вес. На первом шаге из начала координат строятся две линии: горизонтальная, соответствующая тому, что первый предмет не был взят, и наклонная, соответствующая взятому первому предмету. Их проекции на ось  Y  равны весу предмета. На втором шаге опять строим 2 линии, горизонтальная (второй предмет не был взят) или наклонная (второй предмет взят). Положим длину горизонтальных дуг равной нулю, а наклонных — ценности предмета.

Таким образом, любому решению задачи соответствует некоторый путь в сети. Наша задача свелась к нахождению пути максимальной длины.

Пример: Пусть вместимость рюкзака  P=14 .

|  |  |  |
| --- | --- | --- |
| **i** | **Ценность** | **Вес** |
| 1 | 3 | 5 |
| 2 | 5 | 10 |
| 3 | 4 | 6 |
| 4 | 2 | 5 |

На рисунке, в квадратных скобках [] стоит суммарная ценность на каждом шаге алгоритма. Видно, что для конкретного примера она равна 17 .

****

**3) Метод динамического программирования**

В основе метода динамического программирования лежит принцип оптимальности Беллмана: ”Каково бы ни было состояние системы перед очередным шагом, надо выбирать управление на этом шаге так, чтобы выигрыш на этом шаге плюс оптимальный выигрыш на всех последующих шагах был оптимальным”. Проще говоря, оптимальное решение на i шаге находится исходя из найденных ранее оптимальных решений на предшествующих шагах. Из этого следует, что для того чтобы найти оптимальное решение на последнем шаге надо сначала найти оптимальное решения для первого, затем для второго и так далее пока не пройдем все шаги до последнего.

Имеется набор из N предметов. Пусть P- объем рюкзака, ci – стоимость i-го предмета, pi – вес i-го предмета. Value [p, i] – максимальная сумма, которую надо найти. Суть метода динамического программирования – на каждом шаге по весу 1<pi<p находим максимальную загрузку Value[pi, i], для веса pi. Допустим мы уже нашли Value[1..p, 1..i-1], то есть для веса меньше либо равного p и с предметами, взятыми из 1..N-1. Рассмотрим предмет N, если его вес pNменьше p проверим стоит ли его брать.

Если его взять то вес станет p-pi , тогда Value[p, i] = Value[p – pi , i-1] + ci (для Value[p – pi , i-1]) решение уже найдено остается только прибавить ci.

Если его не брать то вес останется тем же и Value[p, i] = Value[p – pi , i-1]. =Из двух вариантов выбирается тот, который дает наибольший результат. Рассмотрим алгоритм подробнее.

****

Динамическое программирование для задачи о рюкзаке дает точное решение, причем одновременно вычисляются решения для всех размеров рюкзака от 1 до P, но какой ценой? Для хранения таблицы стоимости и запоминания того, брался каждый предмет или нет, требуется порядка O(N\*P) памяти, временная сложность равна O(N\*P).

**1) Приближенные алгоритмы**

**1.1) Жадный алгоритм(алгоритм Данцига)**

Согласно [жадному алгоритму](https://ru.wikipedia.org/wiki/%D0%96%D0%B0%D0%B4%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) предметы сортируются по убыванию стоимости единицы каждого. Помещаем в рюкзак то, что помещается и одновременно и самое дорогое, т.е с максимальным отношением цены к весу.

Для сортировки предметов потребуется  O(Nlog(N)) . Далее организуется проход по всем  N  элементам цикла.

Точное решение можно получить не всегда.

Пример. Пусть вместимость рюкзака  P=80 . Предметы уже отсортированы. Применяем к ним жадный алгоритм.

|  |  |  |  |
| --- | --- | --- | --- |
| **i** | **вес** | **цена** | **цена/вес** |
| 1 | 20 | 60 | 3 |
| 2 | 30 | 90 | 3 |
| 3 | 50 | 100 | 2 |

Кладём в рюкзак первый, а за ним второй предметы. Третий предмет в рюкзак не влезет. Суммарная ценность поместившегося равна 150. Если бы были взяты второй и третий предметы, то суммарная ценность составила бы 190. Видно, что жадный алгоритм не обеспечивает оптимального решения, поэтому относится к приближенным.

Рассмотрим непрерывную задачу о ранце, условия для нее те же самые, отличие лишь в том, что мы можем взять часть предмета. То есть предметы можно делить. Пусть у нас есть тот же набор, тогда следуя жадному алгоритму, берем первый и второй предметы, полностью третий предмет не помещается т.к места осталось всего на 30кг, но мы можем брать части предметов, тогда возьмем веса третьего предмета, соответственно и его стоимости, таким образом мы нагрузили рюкзак полностью, стоимость груза стала равна 210у.е**. Для непрерывной задачи о рюкзаке жадный алгоритм будет давать оптимальное решение.**

**1.2) Генетический алгоритм.**

[Генетические алгоритмы](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) были предложены [Джоном Генри Холландом](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D0%BB%D0%BB%D0%B0%D0%BD%D0%B4,_%D0%94%D0%B6%D0%BE%D0%BD_%D0%93%D0%B5%D0%BD%D1%80%D0%B8) в 1970 годуи относятся к так называемым [метаалгоритмам](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%B0%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC). Идея — составление алгоритмов поиска на основе биологической модели механизмов [естественного отбора](https://ru.wikipedia.org/wiki/%D0%95%D1%81%D1%82%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BE%D1%82%D0%B1%D0%BE%D1%80). Базовыми понятиями являются: популяция, отбор, мутация, скрещивание.

[Популяция](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BF%D1%83%D0%BB%D1%8F%D1%86%D0%B8%D1%8F). Составляется набор бинарных строк ([хромосом](https://ru.wikipedia.org/wiki/%D0%A5%D1%80%D0%BE%D0%BC%D0%BE%D1%81%D0%BE%D0%BC%D0%B0)), возможных решений. На основе первой («старой») популяции строится вторая («новая») популяция решений, которая служит «старой» для третьей популяции и т.д

[Отбор](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%B1%D0%BE%D1%80). Задается функция выбора, согласно которой, лучшие представители «старой» популяции выбираются для воспроизводства «новой». Следовательно, алгоритм выбирает наилучшее решение.

[https://upload.wikimedia.org/wikipedia/commons/thumb/c/c9/%D0%A1%D0%BA%D1%80%D0%B5%D1%89%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5.png/220px-%D0%A1%D0%BA%D1%80%D0%B5%D1%89%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5.png](https://commons.wikimedia.org/wiki/File:%D0%A1%D0%BA%D1%80%D0%B5%D1%89%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5.png?uselang=ru)

Скрещивание хромосом. «Родители» обмениваются последними пятью битами и образуют новые хромосомы — «потомки».

[Скрещивание](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B5%D1%89%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Для пары строк («родителей») с определенной длиной r выбирается произвольное число 1 \le s \le r. «Родители» обмениваются между собой битами с s+1-го по r-й и получаются две новые строки («потомки»).

[Мутация](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D1%82%D0%B0%D1%86%D0%B8%D1%8F). Изменение, происходящее с определенной [вероятностью](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C).

Содержимое рюкзака представляется в виде хромосом или бинарных строк, i-й бит которых равен единице в случае наличия предмета в рюкзаке, нулю — в случае его отсутствия. Задается целевая функция S — вместимость рюкзака.

Отбор осуществляется следующим образом.

Выбирается произвольная хромосома. Пусть  L_{max}= max(S,S''-S) — максимальное расхождение между целевой функцией и хромосомой. S'' суммарный вес всех предметов, входящих в рюкзачный вектор. S' — вес рюкзака при выбранной хромосоме.

Если S' \le S, то хромосома оценивается числом q = 1 - \sqrt{|S'-S|/S}.

Если S' > S, то хромосома оценивается числом q = 1 - \sqrt{|S'-S|/L_{max}}.



Алгоритм прерывается после заданного числа итераций.

Генетический алгоритм не гарантирует нахождение оптимального решения, однако показывает хорошие результаты за меньшее время по сравнению с другими алгоритмами.

**Классификация алгоритмов:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Метод** | **Тип алгоритма** | **Сложность** | **Плюсы** | **Минусы** |
| Полный перебор | точный | O(N!) | простота реализации  точное решение | входные данные не велики  временная сложность |
| Метод ветвей и границ | точный | <= O(N!) | возможно значительное сокращение времени работы  простота реализации | в худшем случае работает как полный перебор. |
| Динамическое программирование | точный | O(P\*N) | независимость от вида исходных данных  точное решение  Высокая скорость работы по сравнению с другими алгоритмами (для не больших значений N<50)  Имеем оптимальные загрузки рюкзака для всех его весов от 1 до P | большой объём вычислительной работы  Веса предметов целые, если брать вещественные значения, ДП - алгоритм неприменим! |
| Жадный алгоритм | приближенный | O(N\*Log(N)) | Возможно значительное сокращение времени работы.  Простота реализации. | Всегда можно предоставить такой набор, при котором решение будет не точным. |
| Генетический алгоритм | приближенный |  | высокая скорость  может работать с большими значениями N  независимость от вида исходных данных | не гарантирует нахождение оптимального решения |

**3.Предлагаемый метод решения задачи о ранце**

Для решения задачи комбинаторной оптимизации я выбрала «Генетический алгорим».

**Генети́ческий алгори́тм** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) genetic algorithm) — это [эвристический алгоритм](https://ru.wikipedia.org/wiki/%D0%AD%D0%B2%D1%80%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных [естественному отбору](https://ru.wikipedia.org/wiki/%D0%95%D1%81%D1%82%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BE%D1%82%D0%B1%D0%BE%D1%80) в природе. Является разновидностью [эволюционных вычислений](https://ru.wikipedia.org/wiki/%D0%AD%D0%B2%D0%BE%D0%BB%D1%8E%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F), с помощью которых решаются оптимизационные задачи с использованием методов естественной эволюции, таких как: [наследование](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%B1%D0%B8%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F)), [мутации](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D1%82%D0%B0%D1%86%D0%B8%D1%8F), [отбор](https://ru.wikipedia.org/wiki/%D0%95%D1%81%D1%82%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BE%D1%82%D0%B1%D0%BE%D1%80) и [кроссинговер](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%B8%D0%BD%D0%B3%D0%BE%D0%B2%D0%B5%D1%80). Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.

Интерпретация задачи о ранце и операторов генетического алгоритма с помощью понятий популяционной генетики:

Наименьшей неделимой единицей биологического вида, подверженной действию факторов эволюции, является особь , (индекс k обозначает номер особи, а индекс t – некоторый момент времени эволюционного процесса). В качестве аналога особи , примем произвольное допустимое решение, которому присвоено имя . Действительно, вектор управляющих переменных – это наименьшая неделимая единица, характеризующая в задаче о ранце внутренние параметры на каждом t-м шаге поиска оптимального решения, которые изменяют свои значения в процессе максимизации критерия оптимальности Q. Как уже было отмечено, символьная модель экстремальной задачи о ранце может быть представлена в виде множества двоичных кодировок, которые описывают конечное множество допустимых решений принадлежащих области поиска . Для описания особей введем два типа вариабельных признаков, отражающих качественные и количественные различия между особями по степени их выраженности:

• качественные признаки – признаки, которые позволяют однозначно разделять совокупность особей на четко различимые группы;

• количественные признаки – признаки, проявляющие непрерывную изменчивость, в связи с чем степень их выраженности можно охарактеризовать числом. Качественные признаки особи , определяются из символьной модели задачи - как кодировка s(x), соответствующая точке с именем и составляющие ее . Приведем интерпретацию этих признаков в терминах хромосомной теории наследственности. В качестве гена – единицы наследственного материала, ответственного за формирование альтернативных признаков особи, примем комбинацию , которая определяет фиксированное значение целочисленного кода управляющей переменной . Каждая особь характеризуется n генами, а структуру строки можно интерпретировать хромосомой, содержащей n сцепленных между собой генов, которые следуют друг за другом в строго определенной последовательности. Хромосому особи будем обозначать , т.е.

Согласно хромосомной теории наследственности передача генетической информации будет осуществляться через хромосомы от «родителей» к «потомкам». Местоположение определенного гена в хромосоме называется локусом, а альтернативные формы одного и того же гена, расположенные в одинаковых локусах хромосомы, называются аллелями (аллелеформами). В задаче поиска i-й локус соответствует i-й позиции в строковой кодировке s(x), а аллели – это аналоги множества значений управляющих переменных.

Задача формализуется таким образом, чтобы её решение могло быть закодировано в виде [вектора](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%BA%D1%82%D0%BE%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) («[генотипа](https://ru.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%BE%D1%82%D0%B8%D0%BF)») генов, где каждый ген может быть [битом](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82), числом или неким другим объектом. В классических реализациях генетического алгоритма (ГА) предполагается, что генотип имеет фиксированную длину. Однако существуют вариации ГА, свободные от этого ограничения.

Некоторым, обычно случайным, образом создаётся множество генотипов начальной популяции. Они оцениваются с использованием «функции приспособленности», в результате чего с каждым генотипом ассоциируется определённое значение («приспособленность»), которое определяет насколько хорошо [фенотип](https://ru.wikipedia.org/wiki/%D0%A4%D0%B5%D0%BD%D0%BE%D1%82%D0%B8%D0%BF), им описываемый, решает поставленную задачу. При выборе «функции приспособленности» (или fitness function в англоязычной литературе) важно следить, чтобы её «рельеф» был «гладким».

**В своей программе я реализовала несколько способов создания начальной популяции:**

**1)Алгоритм Данцига для линейной одномерной задачи о ранце**

Теорема (правило Данцига).

Вектор x – вектор решений. R - ограничение по весу. Пусть переменные перенумерованы так, что где – удельная стоимость(вес/стоимость). Тогда оптимальное решение имеет вид

, где s определяется из условия

.

Смысл этого правила очевиден. Единичные значения следует последовательно назначать переменным, начиная с наибольшей удельной стоимости (ценности на единицу веса), при этом .

Я генерирую начальную популяцию в зависимости от удельной стоимости. Удельная стоимость (стоимость/вес) сортирую в порядке невозрастания, а уже в соответствии с ней кодируется особь, по правилу выше.

Что бы получить несколько решения, ввела элемент случайности – выбрасывание одного из предметов, в соответствии с чем, другой предмет имеет возможность попасть в рюкзак. Предметы с вероятностью 50% могут попасть в рюкзак.(см. Приложение).

**2) Жадный алгоритм**

Он не является оптимальным для решения задачи о рюкзаке.

«Иногда может возникнуть искушение использовать жадину везде, где только это возможно, но на некоторых задачах это неприемлемо. К примеру, [задача о рюкзаке](http://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BE_%D1%80%D0%B0%D0%BD%D1%86%D0%B5): вор пробрался на склад, в котором хранятся три вещи весом 10 кг, 20 кг и 30 кг и стоимостью 60, 100 и 120 деревянных вечнозеленых нано-рублей соответственно. Вор максимум может унести 50 кг. Нужно максимизировать прибыль вора. Если поступать здесь жадно и выбирать самую ценную вещь(то есть, 6 нано-рублей за кг первой штуки, 5 нанорублей за кг второй и 4 нанорубля за кг третьей), то вор по-любому должен взять первую вещь, потом останется место для второй вещи, однако оптимальное решение составляет вторая и третья вещь.»

Но он хорошо подходит для создания начальной популяции, если ввести в него элемент случайности. Я генерирую начальную популяцию в зависимости от стоимости. Стоимость сортируется в порядке не убывания - в соответствии с ней кодируется особь. Элемент случайности – либо берем предмет, либо нет, т. е. предметы с вероятностью 50% могут попасть в рюкзак.(см. Приложение).

**3) Случайный алгоритм.**

Полностью рандомно генерируются случайные особи с учётом функции приспособленности.

Из полученного множества решений («поколения») с учётом значения «приспособленности» выбираются решения (обычно лучшие особи имеют большую вероятность быть выбранными), к которым применяются «генетические операторы» (в большинстве случаев «[скрещивание](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B5%D1%89%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)» — crossover и «[мутация](https://ru.wikipedia.org/wiki/%D0%9C%D1%83%D1%82%D0%B0%D1%86%D0%B8%D1%8F)» — mutation), результатом чего является получение новых решений. Для них также вычисляется значение приспособленности, и затем производится отбор («селекция») лучших решений в следующее поколение.

Оператор селекции включает в себя два этапа.

**Этап 1**. Здесь определяется ожидаемое число копий каждой i-ой кодировки t Rtв популяции следующего поколения: .

**Этап 2.** Здесь генерируются новые решения популяции путём указания конкретных кодировок t Rt

, копируемых в на основании соответствующего числа ожидаемых копий .

Этот набор действий повторяется итеративно, так моделируется «эволюционный процесс», продолжающийся несколько жизненных циклов (поколений), пока не будет выполнен критерий остановки алгоритма. Таким критерием может быть:

* нахождение глобального, либо субоптимального решения;
* исчерпание числа поколений, отпущенных на эволюцию;
* исчерпание времени, отпущенного на эволюцию.

Генетические алгоритмы служат, главным образом, для поиска решений в многомерных пространствах поиска.

Таким образом, можно выделить следующие этапы генетического алгоритма:

1. Задать целевую функцию (приспособленности) для особей популяции
2. Создать начальную популяцию

* (Начало цикла)

1. Размножение (скрещивание)
2. Мутирование
3. Вычислить значение целевой функции для всех особей
4. Формирование нового поколения (селекция)
5. Если выполняются условия остановки, то (конец цикла), иначе (начало цикла)

**Реализованные операторы скрещивания:**

Механизм размножения с помощью операторов кроссовера прост, т. к. содержит следующие простейшие операции:

1) генерация случайных чисел;

2) копирование строк фиксированной длины L, являющихся «родительскими» кодировками;

3)обмен кусками «родительских» кодировок, разрываемых в одной и той же точке хромосомы.

Применение этих операторов приводит к тому, что кодировки-потомки будут содержать новые сочетания аллелей генов, принадлежащих кодировкам родителей, т.е. происходит только перераспределение суще-ствующих аллелей родителей по хромосомам потомков.

**1)Одноточечный кроссовер**

Алгоритм одноточечного кроссовера заключается в следующем:

1. Согласно выбранной системе скрещивания формируется «брачная

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| пара» из допустимых родительских кодировок |  '  (1' ,..., L' ) и | | |  |
|  ''  (1'' ,..., L'' ) . |  |  |  |  |
| 2. Случайным образом с равной вероятностью | 1 |  | выбирается |  |
| L 1 |  |  |

точка разрыва r{1,2,...,L1}.

1. Кодировки 'и ''разрываются в одной и той же точке r .
2. Из полученных четырех кусков родительских кодировок:

( ' ,...,  ' ), ( '  ,...,  ' ), ( '' ,..., '' ), ( '' ,...,  '' )

1 r r 1 L 1 r r 1 L

формируются кодировки двух потомков:

|  |  |  |
| --- | --- | --- |
| 1 |  (1',..., r' ), (r''1,...,  L'' | ) , |
| 2 |  (1'' ,...,r'' ), (r'1,...,  L' | ) . |

Если гены в родительских хромосомах имеют одинаковые аллели ( i'  i''  x ) в одном и том же i-м локусе, то и воспроизведенные ими потомки будут иметь в i-м локусе ту же самую аллель (i1i2x).(см. приложение)

**2) Двуточечный кроссовер**

«Родительские» кодировки  ' ,  '' разрываются в двух точках r1 и r2 (r1 < r2), случайным образом выбранных с равной вероятностью без возвращения из интервала [1, (L–1)].(см. приложение)

**3) Однородный кроссовер**

Оператор кроссовера, в котором единственный потомок воспроизводится таким образом, что аллель каждого гена копируется в хромосому потомка либо из хромосомы одного родителя, либо из хромосомы другого

родителя с вероятностью 12.(см. приложение)

**Для метода мутации я реализовала следующие операторы:**

**1)Точечная мутация**

Кодировка мутанта получается путем однократной перестановки аллелей в случайно выбранном локусе и соседнем с ним.(см. приложение)

**2)Инверсия**

Кодировкаразрывается в двух точкахr1иr2(r1<r2),

|  |  |
| --- | --- |
| Случайно | выбранных с равной вероятностью из интервала |
| [1, (L 1)]; значения аллелей во всех генах куска изменяются на | |
| Обратные | («1» на «0» или «0» на «1»); кодировка мутанта  M |

образуется путем соединения вновь двух старых кусков и нового куска

( r1 1 ,..., r2 ) :  M = (1,...,r1 ,  r1 1,..., r2 , r2 1,...,L ) .

(см. приложение)

**3) Сальтация**

В кодировкеслучайным образом инвертируются одновременнозначения аллелей вk(kL)генах родительской хромосомы.(см. приложение)

**4)Транслакация**

В кодировкевыделяется не один кусок[r1,r2],а сразу несколько непересекающихся между собой участков «родительской» хромосомы  ; значения аллелей в мутанте внутри этих участков инвертируются.(см. приложение)

Далее я вычислила значение целевой функции для всех особей.(см. приложение)

И реализовала две **схемы селекции:**

**1)Линейно-ранговая схема селекции**

Одним из способов предотвращения преждевременной сходимости является использование схемы линейной ранговой селекции (scheme of linear rank selection), основанной на том, что в ней вместо значений функции приспособленности i(ait)используются рангиrit,i1,v:

|  |  |  |  |
| --- | --- | --- | --- |
| nit |  n  n  n  | rit | 1 |
|  | 1 |

(1)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **где** rt |  i – ранг | i-**й** кодировки, | когда все особи | | | | множества Rt |  |
| i |  |  | |  |  |  |  |  |

отсортированы в порядке монотонного неубывания значений функции приспособленности;

n – нижняя граница ожидаемого числа копий для «наихудшей» особи с наименьшим значением функции приспособленности, имеющей ранг r1t  1;

n – верхняя граница ожидаемого числа копий для «наилучшей» кодировки с наибольшим значением функции приспособленности,

имеющей рангrvtv.

Для того чтобы определить численные значения граничных оценок ожидаемого числа копийnиn, требуется, чтобы значенияnit,определяемые формулой (1)

удовлетворяли следующим условиям:

1. значенияnitдолжны монотонно увеличиваться0n1tn2t...nvtпо

отношению к возрастанию значений функции приспособленности

1  2  ...  v ;

1. общая сумма ожидаемого числа копий решений, репродуцируемых в популяцию Pt1, должна равняться численности n этой популяции

**n** выбирается из интервала **1,2** случайным образом или детерминированно, а **n** вычисляется по формуле: n  2  n

Для данной схемы селекции вторым шагом я реализовала:

**Остаточный стохастический выбор с возвращением**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. Пусть | nt    , | i  |  | , где |
| 1,v |

  целая часть числа, а  - дробная часть числаnit,являются действительными числами, определенными на первом этапе оператора селекции.

**Целочисленная фаза**.

2. Детерминированно в популяцию Pt1 репродуцируется

nit  копий для каждого решения из репродукционного множества

Rt .

1. Определяется число недостающих копий v,необходимых для заполнения популяции Pt1полностьюкопиями:
   1.  v  v nit .
      1. 1
2. Еслиv0,то алгоритм завершает свою работу.

**Дробная фаза**.

5. Определяется N :  nit .

i1

6. Осуществляется отображение всех решений из Rt в смежные отрезки на оси действительных чисел 0,Nтак, чтобы каждый

сегмент i-го решения равнялся дробной частиnitего ожидаемого числа копий nit.

1. Случайным образом с равной вероятностью генерируется число 0, N .
2. В качестве копии выбирается то решение, в отрезок которого попало число .
3. Процедура стохастического выбора с возвращением повторяется с шага 7 до тех пор, пока не будет получено vкопий. (см. приложение)

**2)турнир**

Схема -турнирной селекции ( -tournament selection) организована несколько иначе, чем описанная выше схема. Случайным образом, с равной вероятностью из выбирается группа из особей. Среди выбранных в группу особей определяется «наилучшая» особь, копия которой репродуцируется в популяцию следующего поколения. Группа особей возвращается в . Процедура выбора копии «наилучших» особей повторяется требуемое число раз. Число решений называется размером турнира. (см. приложение)

Наиболее простым является бинарный турнир, проводимый между двумя решениями 2

**Так же я реализовала метод обработки ограничений «Декодер»:**

Он переводит любую кодировку в допустимое решение, то есть подходящее под ограничение «веса» особи.

**Эксперимент**

Что бы проверить реализованный эволюционный генетический алгоритм и сделать вывод, как работает программа, мною был написан алгоритм проведения «эксперимента», смысл которого заключается в следующем:

* 1. Проводится эксперимент с различными комбинациями операторов генетического алгоритма (всего 72 комбинации).
  2. Задается количество особей.

3)Задается количество итераций (поколений).

4) Так как перебираются все возможные комбинации операторов генетического алгоритма, учитывается задание значения (для – турнира, по умолчанию = 2).

5)Стоимость и вес прописываются в файле(C:/tmp/test\_.txt);

6)Ограничение на рюкзак вводится с клавиатуры.

7) Задается количество запусков ГА (количество проведений эксперимента).

**По каждому эксперименту:**

- Выводится максимальная функция приспособленности с первого по последнее поколение для каждой комбинации алгоритма.

- Выводится максимальное и минимальное значения среди всех значений функции приспособленности для конкретной комбинации.

- Выводится количество итераций до сходимости к лучшей особи.

**Общий результат:**

- Выводится максимальное и минимальное значения функций приспособленности для каждой комбинации среди всех экспериментов.

- Выводится количество итераций до сходимости к лучшей особи для каждой комбинации среди всех экспериментов.

- Выводится среднее значение количества итераций до сходимости особи для каждой комбинации среди всех экспериментов.

\*Все данные выводятся в таблицу Excel.

(алгоритм проведения эксперимента см. приложение)

Для всех задач(см. ниже) я взяла одинаковые параметры:

1)30 особей.

2)40 поколений.

3) = 20.

4)(Стоимость и вес считываются с файла, ограничение на вес задается в программе).

5) Эксперимент повторяется 10 раз.

**Задача №1**.

Цена: 21, 19, 27, 3, 24, 30, 6, 13, 2, 21, 26, 26, 24, 1,10

Вес: 2, 26, 23, 6, 19, 9, 8, 20,11, 1, 17, 21, 7,20,11

Максимальный вес: 80.

Решение: 175.

**Результаты эксперимента (приведена в приложении итоговая таблица по данному примеру).**

* 1. Лучшая особь (самое близкое к точному решение или точное решение) получается, если использовать *жадный алгоритм*(22 из 24 комбинаций дали точное решение). Хуже всего оказался *Алгоритм Данцига* для генерации начальной популяции(1 из 24 комбинаций). Предположительно из-за специфичного «случайного элемента», смысл которого в выбрасывании одного предмета( обнуляется вес), и предметы из точного решения - берутся с вероятностью 50%.

2) За минимальное среднее количество шагов итераций сходимость к лучшей особи даёт следующая комбинация:

*Жадный алгоритм. Одноточечный кроссовер. Инверсия. Бетта-Турнир.*  *Алгоритм сошелся за шаг.*

3)За большее среднее среди всех комбинаций количество итераций к лучшей особи дают сходимость следующие комбинации:

*Случайный алгоритм. Однородный кроссовер. Сальтация. Бетта-Турнир.*

*Алгоритм сошелся за 20 шагов.*

4) За самое минимальное число шагов сошлась к единому решению комбинация:

*Жадный алгоритм. Одноточечный кроссовер. Сальтация. Бетта-Турнир.Падение разнообразия за 1 шаг*

5)За самое большее число шагов сошлась к единому решению комбинация:

*Случайный алгоритм. Однородный кроссовер. Сальтация. Бетта-Турнир Падение разнообразия после 33 шагов. Алгоритм сошелся к лучшему решению.*

6) Самое неточное решение, худшую особь показала комбинация:

*Алгоритм Данцига. Однородный кроссовер. Транслакация. Бетта-Турнир.*

**Задача №2**.

Цена: 6, 3,13,30, 25, 4, 2, 30,22,23, 3, 3, 13, 8,14

Вес: 17,15,3,24, 23, 4,29,10,11,26, 28, 27,23,27,30

Максимальный вес: 80.

Решение: 124.

**Результаты эксперимента:**

1)Лучшая особь (самое близкое к точному решение или точное решение) получается, если использовать *случайный алгоритм*(24 из 24 комбинаций дали точное решение). Хуже всего оказался *Алгоритм Данцига* для генерации начальной популяции(3 из 24 комбинаций). Предположительно из-за специфичного «случайного элемента», смысл которого в выбрасывании одного предмета( обнуляется вес), и предметы из точного решения - берутся с вероятностью 50%.

2) За минимальное среднее количество шагов итераций сходимость к лучшей особи даёт следующая комбинация:

*Случайный алгоритм. Однородный кроссовер. Сальтация. Бетта-Турнир.*  *Алгоритм сошелся за шаг.*

3)За большее среднее среди всех комбинаций количество итераций к лучшей особи дают сходимость следующие комбинации:

*Алгоритм Данцига. Двуточечный кроссовер. Точечная мутация. Линейная-Ранговая.*

*Алгоритм сошелся за 13 шагов.*

4) За самое минимальное число шагов сошлась к единому решению комбинация: *Жадный алгоритм. Одноточечный кроссовер. Транслакация. Бетта-Турнир.Падение разнообразия за 1 шаг.*

5)За самое большее число шагов сошлась к единому решению комбинация:

*Алгоритм Данцига. Двуточечный кроссовер. Точечная мутация. Линейная- Ранговая. Падение разнообразия после 39 шагов. Алгоритм сошелся к лучшему решению.*

6) Самое неточное решение, худшую особь показала комбинация:

*Алгоритм Данцига. Одноточечный кроссовер. Сальтация. Бетта-Турнир.*

**Задача №3**.

Цена: 30,16, 3,11, 4, 6, 9,21,27,14,21,12,19,13,14

Вес: 19, 6, 9,25,10,18,23, 6,25,16, 28,13, 4,23,15

Максимальный вес: 96.

Решение: 141.

**Результаты эксперимента:**

* 1. Лучшая особь (самое близкое к точному решение или точное решение) получается, если использовать *жадный алгоритм*(22 из 24 комбинаций дали точное решение). Хуже всего оказался *Алгоритм Данцига* для генерации начальной популяции( 1 из 24 комбинаций). Предположительно из-за специфичного «случайного элемента», смысл которого в выбрасывании одного предмета( обнуляется вес), и предметы из точного решения - берутся с вероятностью 50%.

2) За минимальное среднее количество шагов итераций сходимость к лучшей особи даёт следующая комбинация:

*Жадный алгоритм. Одноточечный кроссовер. Точечная мутация. Бетта-Турнир.*  *Алгоритм сошелся за шаг.*

3)За большее среднее среди всех комбинаций количество итераций к лучшей особи дают сходимость следующие комбинации:

*Случайный алгоритм. Двуточечный кроссовер. Сальтация. Линейная-Ранговая.*

*Алгоритм сошелся за 9 шагов.*

4) За самое минимальное число шагов сошлась к единому решению комбинация: *Жадный алгоритм. Одноточечный кроссовер. Точечная мутация. Бетта-Турнир. Падение разнообразия за 1 шаг. Алгоритм сошелся к лучшему решению.*

5)За самое большее число шагов сошлась к единому решению комбинация:

*Алгоритм Данцига. Однородный кроссовер. Транслакация. Линейная-Ранговая. Падение разнообразия после 39 шагов.*

6) Самое неточное решение, худшую особь показала комбинация:

*Алгоритм Данцига. Одноточечный кроссовер. Инверсия. Бетта-Турнир.*

**Задача №4**.

Цена: 1,29,23,16, 2,17, 8, 1, 17, 21,28,16,18,27, 6

Вес: 5,14,14,22,30, 8,17, 28,19,12, 25,10,16,22,10

Максимальный вес: 100.

Решение: 152.

**Результаты эксперимента:**

* 1. Лучшая особь (самое близкое к точному решение или точное решение) получается, если использовать жадный алгоритм(16 из 24 комбинаций дали точное решение). Хуже всего оказался Алгоритм Данцига для генерации начальной популяции( 3 из 24 комбинаций). Предположительно из-за специфичного «случайного элемента», смысл которого в выбрасывании одного предмета( обнуляется вес), и предметы из точного решения - берутся с вероятностью 50%.

2) За минимальное среднее количество шагов итераций сходимость к лучшей особи даёт следующая комбинация:

*Жадный алгоритм. Двуточечный кроссовер. Транслакация. Бетта-Турнир.*  *Алгоритм сошелся за 2 шага.*

3)За большее среднее среди всех комбинаций количество итераций к лучшей особи дают сходимость следующие комбинации:

*Жадный алгоритм. Однородный кроссовер. Сальтация. Бетта-Турнир.*

*Алгоритм сошелся за 18 шагов.*

4) За самое минимальное число шагов сошлась к единому решению комбинация: *Жадный алгоритм. Одноточечный кроссовер. Транслакация. Бетта-Турнир.*

*Падение разнообразия за 1 шаг. Алгоритм сошелся к лучшему решению.*

5)За самое большее число шагов сошлась к единому решению комбинация:

*Алгоритм Данцига. Двуточечный кроссовер. Сальтация. Бетта-Турнир. Падение разнообразия после 39 шагов.*

6) Самое неточное решение, худшую особь показала комбинация:

*Алгоритм Данцига. Одноточечный кроссовер. Точечная мутация. Бетта-Турнир.*

**Задача №5**.

Цена: 18,24,14,22,13,18,16,30,25, 4,27,12,19,24,22

Вес: 1,11,16, 6,25,16,25, 9,14,13,13, 4,20, 5,9

Максимальный вес: 74.

Решение: 204

**Результаты эксперимента:**

* 1. Лучшая особь (самое близкое к точному решение или точное решение) получается, если использовать *жадный алгоритм*(18 из 24 комбинаций дали точное решение). Хуже всего оказался *Алгоритм Данцига* для генерации начальной популяции(0 из 24 комбинаций). Предположительно из-за специфичного «случайного элемента», смысл которого в выбрасывании одного предмета( обнуляется вес), и предметы из точного решения - берутся с вероятностью 50%.

2) За минимальное среднее количество шагов итераций сходимость к лучшей особи даёт следующая комбинация:

*Жадный алгоритм. Одноточечный кроссовер. Транслакация. Линейная-Ранговая.*  *Алгоритм сошелся за 4 шага.*

3)За большее среднее среди всех комбинаций количество итераций к лучшей особи дают сходимость следующие комбинации:

*Жадный алгоритм. Одноточечный кроссовер. Точечная мутация. Линейная-Ранговая.Алгоритм сошелся за 17 шагов.*

4) За самое минимальное число шагов сошлась к единому решению комбинация:

*Жадный алгоритм. Однородный кроссовер. Транслакация. Бетта-Турнир.*

*Падение разнообразия за 1 шаг*

5)За самое большее число шагов сошлась к единому решению комбинация:

*Алгоритм Данцига. Одноточечный кроссовер. Точечная мутация. Линейная-Ранговая. Падение разнообразия после 39 шагов.*

4) Самое неточное решение, худшую особь показала комбинация:

Алгоритм Данцига. Одноточечный кроссовер. Инверсия. Бетта-Турнир(50).

**Заключение**

В ходе анализа результатов экспериментов была выявлена лучшая комбинация параметров для каждой задачи, с которыми алгоритм выдает лучшее решение за короткое время работы по сравнению с остальными.

А также худшая, с которой алгоритму нужно больше времени для получения достаточно близкого к оптимальному решения.

Из всех экспериментов, нетрудно заметить, что алгоритм Данцига является нежелательным для формирования начальной популяции.

В противовес ему, жадный алгоритм является желательным для формирования начальной популяции. С этим алгоритмом в 80% из 100% ГА сходился к точному решению.

Нежелательным для популяции – является одноточечный кроссовер. Он учавствовал в большинстве комбинаций, которые показали самые низкие результаты.

Также выявлено понижение разнообразия в популяции в ходе работы.

Хотя алгоритм использовал линейную ранговую схему селекции.

Потвеждая терию, с преждевременной сходимостью в большем соотношении были комбинации, которые используют оператор селекции - турнир.

Для увеличения разнообразия стоит реализовать метод масштабирования функции приспособленности.

Исходя из результатов проведенных опытов с использованием различных параметров, можно сказать, что реализованный генетический алгоритм в большинстве случаев сходится к оптимальному решению для данной задачи, но может быть дополнен другими методами для повышения функциональности.

**Литература**

1)Д.И. Батищев, Е. А. Неймарк, Н. В. Старостин

«Применение генетических алгоритмов к решению задач дискретной оптимизации»,Н.Н,2007.

2)Х. Пападимитриу, К. Стайглиц “Комбинаторная оптимизация. Алгоритмы и сложность”. М.: Мир, 1985.

3)  Гери, М. Вычислительные машины и трудноҏешаемые задачи / М. Гери, Д. Джонсон. - М.: Мир, 1982.

4) <http://habrahabr.ru/post/128704/>

**Приложение**

* 1. ***Реализация генетического алгоритма:***

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace WindowsFormsApplication1

{

class Individ

{

private int[] individ = new int[15];

public int[] IND

{

get { return individ; }

set { individ = value; }//входящий параметр value

}

public override bool Equals(object obj)

{

int k=0;

int sum=0;

Individ individ1 = (Individ)obj; // хэмингово расстояние больше 1.

for (int i = 0; i < 15; i++)

{

k = this.IND[i] ^ individ1.IND[i];

sum += k;

}

if(sum==0)

return true;

else

return false;

}

}

class Population

{

int[] cost=new int[15];

int[] cost1=new int[15];

int[] weight=new int[15];

int[] weight1=new int[15];

string[] str;

string tmp;

System.IO.StreamReader file;

public int[] Getweight()

{

file = new System.IO.StreamReader(@"C://tmp//test\_.txt");

int[] weight = new int[15];

tmp = file.ReadLine();

str = tmp.Split(' ');

for ( int i=0; i<15; i++)

{

weight[i] = Convert.ToInt32(str[i]);

}

return (weight);

}

public int[] Getcost()

{

file = new System.IO.StreamReader(@"C://tmp//test\_.txt");

int[] cost = new int[15];

tmp = file.ReadLine();

str = tmp.Split(' ');

for (int i = 15; i < 30; i++)

{

cost[i-15] = Convert.ToInt32(str[i]);

}

return (cost);

}

public Population()

{

weight = Getweight();

weight1 = Getweight();

cost = Getcost();

cost1 = Getcost();

}

Random rand = new Random(System.DateTime.Now.Millisecond);

int s = 0;

public int summaryweight

{

get { return s; }

set { s = value; }//входящий параметр value

}

private int Limit = 0;

public int LIMIT

{

get { return Limit; }

set { Limit = value; }//входящий параметр value

}

//1)Алгоритм Данцига для особи из начальной популяции с элементом случайности выбрасывания одного предмета( обнуляется вес) и предметы из точного решения берутся с вероятностью 50%

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public Individ algorithm\_Danzig()

{

Individ ind = new Individ();

List<int> weight\_ = new List<int>();

List<double> specific\_cost = new List<double>();

double[] sp\_c = new double[15];

summaryweight = 0;

int j = rand.Next(15);

for (int i = 0; i < 15; i++)

{

if (i != j)

{

sp\_c[i] = (double)cost[i] / weight[i];

specific\_cost.Add(sp\_c[i]);

summaryweight += weight[i];

if (summaryweight <=LIMIT)

weight\_.Add(weight[i]);

}

else

{

sp\_c[i] = 0;

specific\_cost.Add(sp\_c[i]);

weight[i] = 0;

if (summaryweight <=LIMIT)

weight\_.Add(weight[i]);

}

}

specific\_cost.Sort();

specific\_cost.Reverse();

for (int k = 0; k < 15; k++)

{

for (int i = 0; i < 15; i++)

{

if (sp\_c[i] != -1)

{

if (sp\_c[i] == specific\_cost.ElementAt(k) && i < weight\_.Count && weight[i] != 0)

{

ind.IND[k] = rand.Next(2);

sp\_c[i] = -1;

break;

}

else

if (sp\_c[i] == specific\_cost.ElementAt(k) && i >= weight\_.Count || sp\_c[i] == specific\_cost.ElementAt(k) && weight[i] == 0)

{

ind.IND[k] = 0;

sp\_c[i] = -1;

break;

}

}

}

}

for (int i = 0; i < 15; i++)

{

if (i == j)

{

weight[i] = weight1[i];

}

}

return (ind);

}

//2)Жадный алгоритм для особи из начальной популяции, элемент случайности //\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public Individ algorithm\_greedy()

{

Individ ind = new Individ();

summaryweight = 0;

List<int> cost\_ = new List<int>();

for (int i = 0; i < 15; i++)

{

cost\_.Add(cost[i]);

}

cost\_.Sort();

cost\_.Reverse();

for (int i = 0; i < 15; i++)

{

for (int j = 0; j < 15; j++)

{

if (cost[j] == cost\_.ElementAt(i))

{

summaryweight += weight[j];

cost[j] = -1;

if (summaryweight <= LIMIT)

{

ind.IND[j] = rand.Next(2);

if (ind.IND[j] == 0)

summaryweight -= weight[j];

}

else

ind.IND[j] = 0;

break;

}

}

}

for (int i = 0; i < 15; i++)

cost[i] = cost1[i];

return (ind);

}

//3)Случайный алгоритм для особи из начальной популяции

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public Individ algoritm\_Rand()

{

Individ ind = new Individ();

summaryweight = 0;

for (int i = 0; i < 15; i++)

{

summaryweight += weight[i];

if (summaryweight <= LIMIT)

{

ind.IND[i] = rand.Next(2);

if (ind.IND[i] == 0)

summaryweight -= weight[i];

}

else

ind.IND[i] = 0;

}

return (ind);

}

//-------------------------------------------------------------------------------

public List<Individ> population\_alg(int n, int k) // для разных особей в популяции

{

List<Individ> populatiON = new List<Individ>();//для множества решений

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

{

Individ indd = new Individ();

switch (k)

{

case 1:

indd = algorithm\_Danzig();

break;

case 2:

indd = algorithm\_greedy();

break;

case 3:

indd = algoritm\_Rand();

break;

default: break;

}

if (!populatiON.Contains(indd))

populatiON.Add(indd);

else n++;

}

return (populatiON);

}

//4)Кроссовер (одноточечный и двуточечный).

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public List<Individ> Crossover\_Point(List<Individ> populatiON, int l)

{

List<Individ> population1 = new List<Individ>();

int k = 0;

int r = 0;

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

{

for (int j = 0; j < populatiON.Count; j++)

{

if (i != j)

{

k = 5; // rand.Next(13);

r = 10;// rand.Next(k + 1, 14);

Individ indiv = new Individ();

if (l == 1)

{

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

indiv.IND[s] = populatiON.ElementAt(j).IND[s];

for (int s = k; s < 15; s++)

indiv.IND[s] = populatiON.ElementAt(i).IND[s];

}

if (l == 2)

{

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

indiv.IND[s] = populatiON.ElementAt(j).IND[s];

for (int s = k; s < r; s++)

indiv.IND[s] = populatiON.ElementAt(i).IND[s];

for (int s = r; s < 15; s++)

indiv.IND[s] = populatiON.ElementAt(j).IND[s];

}

population1.Add(indiv);

}

}

}

population1.AddRange(populatiON);

return (population1);

}

// 4.1) Однородный кроссовер.

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public List<Individ> Crossover\_uniform(List<Individ> populatiON)

{

List<Individ> population1 = new List<Individ>();

int k = 0;

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

{

for (int j = 0; j < populatiON.Count; j++)

{

if (i != j)

{

Individ indiv = new Individ();

for (int s = 0; s < 15; s++)

{

k = rand.Next(10);

if (k > 5)

indiv.IND[s] = populatiON.ElementAt(j).IND[s];

else

indiv.IND[s] = populatiON.ElementAt(i).IND[s];

}

population1.Add(indiv);

}

}

}

population1.AddRange(populatiON);

return (population1);

}

// 5)Все типы мутации(точечная, инверсия, траслакация, сальтация)

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public List<Individ> mutation(List<Individ> populatiON, int l)

{

List<Individ> population1 = new List<Individ>();

int k = 0;

int r = 0;

int s = (int)(populatiON.Count - Math.Sqrt(populatiON.Count));

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

{

k = rand.Next(13);

r = rand.Next(k + 1, 14);

Individ indiv = new Individ();

indiv = populatiON.ElementAt(i);

if (rand.Next(100) == 1)

{

if (l == 0)

{

if (populatiON.ElementAt(i).IND[k] == 0)

indiv.IND[k] = 1;

else

indiv.IND[k] = 0;

}

if (l == 1)

{

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

{

if (populatiON.ElementAt(i).IND[j] == 0)

indiv.IND[j] = 1;

else

indiv.IND[j] = 0;

}

}

if (l == 2)

{

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

{

if (populatiON.ElementAt(i).IND[j] == 0)

indiv.IND[j] = 1;

else

indiv.IND[j] = 0;

}

for (int j = r; j < 15; j++)

{

if (populatiON.ElementAt(i).IND[j] == 0)

indiv.IND[j] = 1;

else

indiv.IND[j] = 0;

}

}

if (l == 3)

{

int a1, a2;

a1 = populatiON.ElementAt(i).IND[k];

a2 = populatiON.ElementAt(i).IND[r];

indiv.IND[r] = a1;

indiv.IND[k] = a2;

}

}

int c = 0;

for (int j = 0; j < 15; j++)

{

if (indiv.IND[j] == 0)

c++;

}

if (c != 15)

population1.Add(indiv);

}

for (int i = s; i < populatiON.Count; i++)

{

population1.Add(populatiON.ElementAt(i));

}

return (population1);

}

//6) Декодер.

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public List<Individ> evaluation(List<Individ> populatiON)// для отбора особей, подходящих под ограничение

{

List<Individ> population1 = new List<Individ>();

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

{

summaryweight = 0;

for (int g = 0; g < 15; g++)

{

if (populatiON.ElementAt(i).IND[g] == 1)

summaryweight += weight[g];

}

if (summaryweight <= LIMIT)

{

population1.Add(populatiON.ElementAt(i));

}

}

return (population1);

}

//7)Селекция. – турнир.

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public Individ selection(List<Individ> populatiON, int beta) //селекция бета-турнир

{

int c = 0;

int count = 0;

int summarycost;

List<int> cost\_ = new List<int>();

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

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

{

c = rand.Next(2);

summarycost = 0;

if (c == 1 && count < beta && !number.Contains(i))

{

for (int g = 0; g < 15; g++)

{

if (populatiON.ElementAt(i).IND[g] == 1)

{

summarycost += cost[g];

}

}

cost\_.Add(summarycost);

count++;

number.Add(i);

}

if (i == populatiON.Count - 1 && count < beta)

{

i = -1;

}

}

int max\_cost = 0;

Individ indiv = new Individ();

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

{

if (cost\_.ElementAt(i) > max\_cost)

{

max\_cost = cost\_.ElementAt(i);

indiv = populatiON.ElementAt(number.ElementAt(i));

}

else

cost\_[i] = max\_cost;

}

return (indiv);

}

// 8)Линейная – ранговая схема селекции.

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

public List<Individ> Selection\_Rang(List<Individ> populatiON, int c)

{

List<Individ> population1 = new List<Individ>();

List<Individ> population2 = new List<Individ>();

int[] sumcost = new int[populatiON.Count];

List<int> sumcost1 = new List<int>();

int[] rang = new int[populatiON.Count]; //для рангов

double[] n = new double[populatiON.Count]; // для всех ожид чисел

n[populatiON.Count - 1] = rand.NextDouble() + 1.1;

n[0] = 2 - n[populatiON.Count - 1];

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

{

for (int g = 0; g < 15; g++)

{

if (populatiON.ElementAt(i).IND[g] == 1)

sumcost[i]+=cost[g];

}

sumcost1.Add(sumcost[i]);

}

sumcost1.Sort();

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

{

for (int j = 0; j < populatiON.Count; j++)

{

if (sumcost[j] != -1)

{

if (sumcost[j] == sumcost1.ElementAt(i))

{

sumcost[j] = -1;

population1.Add(populatiON.ElementAt(j));

break;

}

}

}

}

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

{

rang[i] = i + 1;

n[i] = n[0] + (n[populatiON.Count - 1] - n[0]) \* (rang[i] - 1) / (populatiON.Count - 1);

if (n[i] >= 1)

{

population2.Add(population1[i]);

}

}

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

{

if (population2.Count < c)

{

for (int j = 0; j < populatiON.Count; j++)

{

if (n[j] > 1 && rand.Next((int)(n[j] - 1) \* 100) == 0 || n[j] < 1 && rand.Next((int)(n[j] \* 100)) == 0)

population2.Add(population1[j]);

}

i++;

}

}

if (population2.Count > c)

{

population1.Clear();

int k = 0;

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

{

k = rand.Next(population2.Count);

population1.Add(population2.ElementAt(k));

}

}

if (population2.Count > c)

return (population1);

else

return (population2);

}

//-----------------------------------------------------------

public int Return\_Max\_Cost(Individ indiv)

{ int summarycost = 0;

for (int g = 0; g < 15; g++)

if (indiv.IND[g] == 1)

summarycost += cost[g];

return (summarycost);

} // для вывода функции приспособленности

//----------------------------------------------------------

public int Return\_Max\_Cost\_At\_Generation(List<Individ> indiv)

{

int summarycost = 0;

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

{

if (Return\_Max\_Cost(indiv[i]) > summarycost)

summarycost = Return\_Max\_Cost(indiv[i]);

}

return summarycost;

}

}

}

***2.Реализация эксперимента***

excel.SheetsInNewWorkbook = startsCount + 1;

excel.Workbooks.Add(Type.Missing);

Excel.Workbook workbook = excel.Workbooks[1]; //получам ссылку на первую открытую книгу

Excel.Worksheet sheet = workbook.Worksheets.get\_Item(1);

string[] vsS ={ "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L",

"M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z",

"AA", "AB", "AC", "AD", "AE","AF", "AG", "AH", "AI", "AJ", "AK", "AL","AM","AN","AO","AP","AQ","AR","AS","AT","AU","AV",

"AW","AX","AY","AZ","BA","BB","BC","BD","BE","BF","BG","BH","BI","BJ","BK","BL","BM","BN","BO","BP","BQ","BR","BS",

"BT","BU"};

string[] initialPopulation = { "Алгоритм Данцига", "Жадный алгоритм", "Случайный алгоритм" };

string[] crossover = { "Одноточечный кроссовер", "Двуточечный кроссовер", "Однородный кроссовер" };

string[] mutation = { "Точечная мутация", "Инверсия", "Транслакация", "Сальтация" };

string[] selection = { "Бетта-Турнир", "Линейная-Ранговая" };

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

{

int count = 1;

sheet = workbook.Worksheets.get\_Item(s + 1);

sheet.Cells[iterationCount + 4, count] = "max";

sheet.Cells[iterationCount + 5, count] = "min";

sheet.Cells[iterationCount + 6, count] = "i";

for (int i = 0; i < 3; i++)

{

for (int j = 0; j < 3; j++)

{

for (int k = 0; k < 4; k++)

{

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

{

count++;

populatiON.Clear();

populatiON = popul.population\_alg(populationCount, i + 1);

for (int x = 0; x < iterationCount; x++)

{

switch (j)

{

case 0:

case 1:

populatiON = popul.Crossover\_Point(populatiON, j + 1);

break;

case 2:

populatiON = popul.Crossover\_uniform(populatiON);

break;

}

populatiON = popul.mutation(populatiON, k);

switch (g)

{

case 0:

List<Individ> population1 = new List<Individ>();

Individ indiv = new Individ();

populatiON = popul.evaluation(populatiON);

for (int h = 0; h < populationCount; h++)

{

indiv = popul.selection(populatiON, betta);

population1.Add(indiv);

}

populatiON.Clear();

populatiON.AddRange(population1);

break;

case 1:

populatiON = popul.evaluation(populatiON);

populatiON = popul.Selection\_Rang(populatiON, populationCount);

break;

}

sheet.Cells[x + 2, count] = popul.Return\_Max\_Cost\_At\_Generation(populatiON);

}

sheet.Cells[1, count] = initialPopulation[i] + Environment.NewLine + crossover[j] + Environment.NewLine + mutation[k] + Environment.NewLine + selection[g] + Environment.NewLine;

sheet.Cells[iterationCount + 4, count].Formula = "= MAX(" + vsS[count - 1] + 2 + ":" + vsS[count - 1] + (iterationCount + 1) + ")";

sheet.Cells[iterationCount + 5, count].Formula = "= MIN(" + vsS[count - 1] + 2 + ":" + vsS[count - 1] + (iterationCount + 1) + ")";

sheet.Cells[iterationCount + 6, count].Formula = "= MATCH(" + vsS[count - 1] + (iterationCount + 4) + "," + vsS[count - 1] + 2 + ":" + vsS[count - 1] + (iterationCount + 1) + "," + 0 + ")";

}

}

}

}

}

sheet = workbook.Worksheets.get\_Item(startsCount + 1);

int number = 1;

sheet.Cells[2, number] = "max";

sheet.Cells[3, number] = "min";

sheet.Cells[4, number] = "i";

sheet.Cells[5, number] = "i(среднее)";

for (int i = 0; i < 3; i++)

for (int j = 0; j < 3; j++)

for (int k = 0; k < 4; k++)

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

{

number++;

sheet.Cells[1, number] = initialPopulation[i] + Environment.NewLine + crossover[j] + Environment.NewLine + mutation[k] + Environment.NewLine + selection[g] + Environment.NewLine;

string max = getAFormula("= MAX(", 4, number, startsCount, vsS);

string min = getAFormula("= MIN(", 5, number, startsCount, vsS);

string average = getAFormula("= AVERAGE(", 6, number, startsCount, vsS);

string convergence = "=MIN(";

for (int v = 1; v < startsCount; v++)

{

convergence += "IF(" + vsS[number - 1] + 2 + "=Лист" + v + "!" + vsS[number - 1] + (iterationCount + 4) + "," + "Лист" + v + "!" + vsS[number - 1] + (iterationCount + 6) + "," + 100 + ")" + ",";

}

convergence += "IF(" + vsS[number - 1] + 2 + "=Лист" + startsCount + "!" + vsS[number - 1] + (iterationCount + 4) + "," + "Лист" + startsCount + "!" + vsS[number - 1] + (iterationCount + 6) + "," + 100 + "))";

sheet.Cells[2, number].Formula = max;

sheet.Cells[3, number].Formula = min;

sheet.Cells[4, number].Formula = convergence;

sheet.Cells[5, number].Formula = average;

}

MessageBox.Show(@"Отчет успешно создан!");

excel.Visible = true;

}

string getAFormula(string formula, int count, int number, int startsCount, string[] vsS)

{

string tmp = formula;

for (int d = 1; d < startsCount; d++)

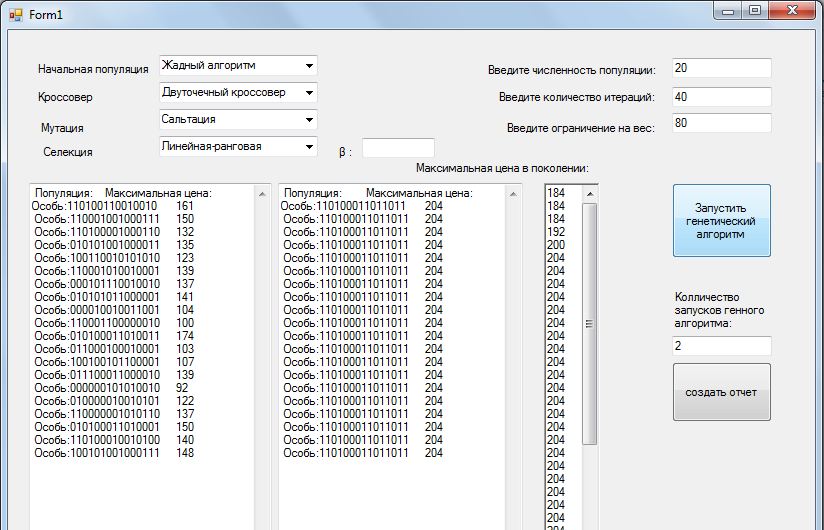
tmp += "Лист" + d + "!" + vsS[number - 1] + (iterationCount + count) + ",";

tmp += "Лист" + startsCount + "!" + vsS[number - 1] + (iterationCount + count) + ")";

return tmp;

}

***3.Пример запуска алгоритма 1 раз***

******

***5) Итоговая таблица для задачи №1.(Будет распечатана из Excel).***