

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по научно-исследовательской работе

**Тема: Реализация бенчмарка и сравнение эффективности эвристических
алгоритмов решения задачи покрытия множества**

Студент гр. 7381

Вологдин М.Д.

Преподаватель

Васькин П.И.

Санкт-Петербург

2022

ЗАДАНИЕ
НА НАУЧНО-ИССЛЕДОВАТЕЛЬСКУЮ РАБОТУ

Студент Вологдин М.Д.

Группа 7381

Тема работы: Реализация бенчмарка и сравнение эффективности эвристических алгоритмов решения задачи покрытия множества

Задание на работу:

Реализация алгоритма генерации таблиц покрытия на Python

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 11.10.2022

Дата сдачи реферата: 20.12.2022

Дата защиты реферата: 27.12.2022

Студент

Вологдин М.Д.

Преподаватель

Васькин П.И.

СОДЕРЖАНИЕ

Введение	5
1. Постановка задачи	6
2. Результаты работы в осеннем семестре.....	7
2.1. План	7
2.1.1. Детализация постановки задачи на осенний семестр	7
2.2. Оригинальный алгоритм генерации таблицы покрытия	7
2.3. Псевдокод алгоритма.....	10
2.4. Пример работы алгоритма.....	12
2.5. Описание предполагаемого метода решения	13
2.6. Ссылка на репозиторий с исходным кодом.....	14
3. План работы на Весенний семестр	15
3.1. Детализация постановки задачи на весенний семестр	15
3.2. Обоснование актуальности разработки	15
ЗАКЛЮЧЕНИЕ	16
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	17
ПРИЛОЖЕНИЕ А	19

АННОТАЦИЯ

В рамках данной работы был рассмотрен и реализован оригинальный алгоритм генерации таблиц покрытия. Данный алгоритм будет использоваться в дальнейших исследованиях как источник данных для экспериментов над алгоритмами решения задачи покрытия множества.

SUMMARY

Within the framework of this work, an original algorithm for generating coverage tables was considered and implemented. This algorithm will be used in further research as a data source for experiments on algorithms for solving the set covering problem.

ВВЕДЕНИЕ

Задача покрытия множества относится к классу NP-полных комбинаторных задач, точное решение которых состоит в полном переборе всех возможных вариантов.

Множество практических задач опирается на задачу покрытия множеств: построение расписаний, расположение пунктов обслуживания, построение электронных схем и т.п. На сегодняшний день нахождение оптимальных решений задачи не перестает быть актуальным, в связи с чем, существует большое количество эвристических алгоритмов её решения.

Целью данной работы является проведение сравнительного анализа эвристических алгоритмов решения задачи покрытия множества

Объектом исследования в данной работе выступают эвристические алгоритмы решения задачи покрытия множества

Предмет исследования – мощность приближенно-оптимального решения и время работы эвристических алгоритмов решения задачи покрытия множества

Практическая значимость работы и необходимость проведения исследования заключается в том, что для сравнения эвристик в задаче покрытия множества не существует иного способа, кроме как непосредственное сравнение одной эвристики с другой.

Для выполнения поставленной цели необходимо решить следующие задачи:

- Определение списка алгоритмов для исследования
- Реализация алгоритмов решения задачи покрытия множества
- Проведение экспериментов по сравнению алгоритмов на различных наборах данных
- Классификация алгоритмов и анализ полученных результатов

1. ПОСТАНОВКА ЗАДАЧИ

Пусть $A = (a_{ij})$ — произвольная матрица размера $m \times n$ с элементами $a_{ij} \in \{0,1\}$ без нулевых строк и столбцов. Будем говорить, что в A столбец i покрывается строкой j , если $a_{ij} = 1$. Подмножество строк называется покрытием, если в совокупности они покрывают все столбцы матрицы A . Требуется найти покрытие минимальной мощности (невзвешенная задача покрытия). Вводя переменные x_j , равные 1, если строка j входит в искомое покрытие, и равные 0 в противном случае, приходим к следующей формулировке задачи о покрытии: минимизировать сумму $\sum_{j=1}^m x_j$ при ограничениях

$$\sum_{j=1}^m a_{ij} x_j, i = 1, \dots, n, x_j \in \{0,1\}, j = 1, \dots, m$$

Введем обозначения:

$M = \{1, \dots, m\}, N = \{1, \dots, n\}$ — множества номеров строк и столбцов матрицы A ;

$N_i = \{j \in M \mid a_{ij} = 1\}$ — множество столбцов, покрываемых строкой $j \in M$;

$M_j = \{i \in N \mid a_{ij} = 1\}$ — множество строк, покрывающих столбец $i \in N$.

Подмножество строк $J \subseteq M$ является покрытием, если $\bigcup_{j \in J} N_j = N$.

Решение таблицы покрытия (ТП) состоит в нахождении минимума $|J|$ среди всех покрытий J . Покрытие J называется тупиковым, если при любом $j \in J$ множество $J \setminus \{j\}$ не является покрытием. Очевидно, что решение задачи следует искать среди тупиковых покрытий.

2. РЕЗУЛЬТАТЫ РАБОТЫ В ОСЕННЕМ СЕМЕСТРЕ

2.1. План

2.1.1. Детализация постановки задачи на осенний семестр

Реализовать все нереализованные эвристики, которые будут сравниваться и спланировать эксперимент по сравнению эвристик. Оригинальные эвристики должны быть реализованы либо на SQL в среде MS SQL Server 2019, либо на Python.

2.2. Оригинальный алгоритм генерации таблицы покрытия

При сравнении алгоритмов решения задачи покрытия множества немаловажным аспектом является процесс генерации таблиц покрытия, на которых будут проводиться эксперименты. Описанный алгоритм был предложен П.И. Васькиным специально для данного исследования.

Обозначим через $T = \begin{pmatrix} t_{11} & \dots & t_{1n} \\ \vdots & \ddots & \vdots \\ t_{m1} & \dots & t_{mn} \end{pmatrix}$ таблицу покрытия (ТП), в которой

$t_{ji} = 1$, если строка s_j покрывает столбец c_i , в противном случае $t_{ji} = 0$.

Следовательно, метаинформация псевдослучайного датчика таблиц покрытия должна включать в свой состав свой состав:

1. m – число строк ТП
2. n – число столбцов ТП

Обозначим через $S = \{s_1, \dots, s_j, \dots, s_m\}$ – множество строк формируемой таблицы покрытия, а через $C = \{c_1, \dots, c_i, \dots, c_n\}$ – множество ее столбцов. В качестве третьего параметра метаинформации возьмем $P = \{p_1, \dots, p_k, \dots, p_{k_{\max}}\}$ – распределение вероятностей, где p_k – вероятность того, что столбец имеет k ненулевых элементов.

Так как дополнительным требованием к генерируемым таблицам является требование цикличности (невозможность применения точных правил преобразования таблицы покрытия), то $p_1 = 0$. В противном случае будет получаться существенное количество столбцов, и строки, покрывающие эти столбцы, точно должны быть выбраны в решении ТП.

Так как вероятность применимости правила удаления столбцов с большим количеством отметок велика, то $k_{\max} < m$. Слишком большое значение k_{\max} будет приводить к невозможности получить частоты столбцов, соответствующие распределению вероятностей P .

Пусть $L = \{l_2, \dots, l_k, \dots, l_{k_{\max}}\}$ – множество, в котором каждый элемент l_k равен количеству столбцов ТП с k отметками. Тогда общее число отметок в генерируемой ТП должно быть $L_T = \sum_{k=2}^{k_{\max}} k \times l_k$.

Обозначим через \tilde{S}_i множество множеств строк, допустимых для выбора отметок столбца, получаемого на шаге i . В начале генерации ТП $\tilde{S}_1 = \{S\}$.

Рассмотрим первый шаг генерации ТП. В множестве \tilde{S}_1 только один член – множество всех строк ТП. При выборе отметок для первого столбца не существует никаких ограничений – любая строка равновероятна. Обозначим через $S(c_1) = \{s_1(c_1), \dots, s_\rho(c_1), \dots, s_{l_1}(c_1)\}$ множество строк с ненулевыми значениями в первом столбце ТП. При назначении строк для второго столбца нужно не допустить появления в нем вместе всех этих строк. Математически это можно реализовать исключением множества S , вместо которого нужно добавить в \tilde{S}_2 множества, полученные декомпозицией S по множеству $S(c_1)$: $D(S, S(c_1)) = \{\{S - s_\rho(c_1), \rho \in [1, l_1]\}\}$. Таким образом, вместо одного множества в \tilde{S}_2 включаются l_1 множеств с мощностью на единицу меньшей, чем мощность S .

На втором шаге множество допустимых строк выбираем из элементов множества \tilde{S}_2 . Для оптимизации алгоритма множество \tilde{S} должно быть упорядочено по убыванию мощности его элементов. На втором шаге мощность всех элементов множества \tilde{S} одинакова, поэтому можно выбрать для назначения строк с ненулевыми значениями во втором столбце любой из них. Указанный подход гарантирует, что столбцы 1 и 2 не будут соответствовать условию их исключения из ТП.

Рассмотрим выбор ненулевых значений столбца c_i . В качестве множества допустимых строк \hat{S}_i будем использовать любой элемент множества \tilde{S}_i наибольшей мощности. Если $|\hat{S}_i| < l_i$, то это означает, что требуемое распределение вероятностей P достичь невозможно. Если $|\hat{S}_i| < 2$, то к этому добавляется невозможность достижения требуемого числа столбцов n .

После определения псевдослучайным образом множества $S(c_i) \subseteq \hat{S}_i$ формируем множество \tilde{S}_{i+1} . Пусть $\tilde{S}_i = \{\tilde{s}_{i1}, \dots, \tilde{s}_{iv}, \dots, \tilde{s}_{iv_{\max}}\}$. Множество \tilde{S}_{i+1} получается в результате рассмотрения каждого элемента множества \hat{S}_i . Если $S(c_i) \subseteq \tilde{s}_{iv}$, то во множество \tilde{S}_{i+1} добавляются множества полученные декомпозицией $D(\tilde{s}_{iv}, S(c_i))$, иначе \tilde{s}_{iv} переносится во множество \tilde{S}_{i+1} без декомпозиции. Таким образом \tilde{S}_{i+1} увеличивается на $|\tilde{s}_{iv} \cap S(c_i)|$ членов, в том случае, если новые члены не являются нестрогими подмножествами других.

Можно заметить, что мощность множества \hat{S}_i растёт очень быстро, однако в предполагаемых сценариях использования (n близко к m) на последнем шаге будет оставаться много невостребованных элементов. Для оптимизации алгоритма будем использовать эвристическую функцию $h(\hat{S}_i)$, которая ограничит рост мощности \hat{S}_i .

Предложенная эвристическая функция состоит из 2х условий:

1. Все элементы из S содержатся в \hat{S}_i хотя бы в единичном экземпляре
2. Мощность множества $|\hat{S}_i| \geq \sqrt{n}$

Исходный код алгоритма на языке Python представлен в приложении А.

2.3. Псевдокод алгоритма

Псевдокод предложенного алгоритма выглядит следующим образом:

Вход: m – число строк ТП, n – число столбцов ТП,

$P = \{p_1, \dots, p_k, \dots, p_{k_{\max}}\}$ – распределение вероятностей

Выход: Готовая таблица покрытия

$\tilde{S}_0 = [1, \dots, m]$

$L = [i \cdot n \text{ for } i \text{ in } P] \# L = \{l_2, \dots, l_k, \dots, l_{k_{\max}}\}$

$T = \begin{pmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{pmatrix} \# c_i - \text{столбец } i \text{ матрицы}$

for i , $rows_count$ in L :

$\tilde{S}_i.sort()$ # сортировка элементов \tilde{S}_i по длине

$S(c_i) = choose_rows(rows_count)$ # выбор строк ТП, которые будут
заполнены в текущем столбце

$fill(T, S(c_1))$ # заполнение этих строк в результирующей ТП

$\tilde{S}_{i+1} = decomposition(\tilde{S}_i, S(c_1))$

return T

Важные вспомогательные функции выглядят следующим образом:

- Функция для выбора строк

choose_rows(rows_count):

s = choose_min_row(T) # выбор строки с наименьшим количеством отметок

$\tilde{s}_{iv} = any(s \subseteq \tilde{S}_i)$ # выбираем любой элемент из \tilde{S}_i

в котором есть строка s

return $\tilde{s}_{iv}[0:rows_count]$ # возвращаем rows_count элементов из $S(c_i)$

Примечание: возвращаемое множество должно содержать s

- Функция декомпозиции:

decomposition(\tilde{S}_i, S):

$\tilde{S}_{i+1} = []$

for s_{iv} in \tilde{S}_i :

if $S \subseteq \tilde{s}_{iv}$:

for s_p in S :

if $\left(|s_{iv} \setminus s_p| > 2 \text{ and } s_{iv} \setminus s_p \text{ не является подмножеством} \right)$
любого из элементов \tilde{S}_{i+1}

$\tilde{S}_{i+1}.append(s_{iv} \setminus s_p)$

else:

if $\left(|s_{iv}| > 2 \text{ and } s_{iv} \text{ не является подмножеством} \right)$
любого из элементов \tilde{S}_{i+1}

$\tilde{S}_{i+1}.append(s_{iv})$

if $h(\tilde{S}_i)$:

break

return \tilde{S}_{i+1}

2.4.Пример работы алгоритма

Запустим алгоритм при $n = 8, m = 8, P = \{0, 0.3, 0.4, 0.3\}$. Результат вывода программы представлен на рис. 1 и в табл. 1.

С помощью описанного ранее Муравьиного алгоритма было найдено решение этой таблицы. Это решение выделено цветом.

```

      cols  cols_count
0    {1, 5, 6}         3
1    {0, 3, 7}         3
2      {2, 3}          2
3    {2, 4, 6}         3
4    {0, 4, 6}         3
5  {2, 4, 5, 7}         4
6  {4, 5, 6, 7}         4
7    {1, 3, 7}         3
time 0.01695418357849121

Process finished with exit code 0

```

Рисунок 1 – Пример вывода программы

Таблица 1 – Пример сгенерированной таблицы

	1	2	3	4	5	6	7	8
1*		v				v	v	
2*	v			v				v
3			v	v				
4*			v		v		v	
5	v				v		v	
6			v		v	v		v
7					v	v	v	v
8		v		v				v

Несложно определить, что сгенерированная таблица является цикличной, т.е. никакое множество отметок в строке\столбце не является подмножеством множества отметок в любой другой строке\столбце, а также отсутствуют столбцы с единственной отметкой.

2.5. Описание предполагаемого метода решения

Последующие исследование предполагает использование генератора таблиц покрытия, реализованного в данной работе, для генерации таблиц, которые впоследствии будут служить источником данных для алгоритмов их решения.

Список исследуемых алгоритмов выглядит следующим образом:

- Жадный алгоритм
- Муравьиный алгоритм
- Генетический алгоритм (в разных реализациях)
- Алгоритм черной дыры
- Алгоритм светлячков

Поскольку многие из выбранных алгоритмы сильно зависят от параметров метаэвристики, которую они реализуют, то каждый алгоритм можно рассматривать как семейство схожих алгоритмов.

Предполагаемое решение предусматривает использование стороннего программного обеспечения (Далее CTS – Cover Table Solver) для упрощения проведения экспериментов. Также использование данного ПО поможет сосредоточиться именно на результатах, не отвлекаясь на собственную подобную разработку. Интерфейс CTS представлен на рис. 2.

CTS работает только с алгоритмами на SQL SERVER 2019, поэтому алгоритмы, разрабатываемые на Python, будут внедрены как внутренние SQL процедуры.

Приложение - Решение задачи по работе

Действия: Настройка, Отображение, Завершение работы, Справка

Настройка, Отображение, Завершение работы, Справка

Эксперименты

Код	Дата/время создания	Тип эксперимента	Описание эксперимента	Алгоритм решения	Параметры решения	Алгоритм решения	Параметры решения	Кол-во решений	Кол-во строк	Запускается	Среднее время решения (сек.)	Среднее время решения (мин.)	Содержимое файла, строки
1	29.09.2022 1...	Серия запусков	Серия из 20 запусков N14	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	4,7	0,3	Дл
2	30.09.2022 1...	Серия запусков	Пробный эксперимент N15	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	Без решения		10	10	20			Дл
3	30.09.2022 1...	Серия запусков	Серия из 10 запусков N16	Не указана		dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	5,5502	0,218526	Дл
4	30.09.2022 1...	Серия запусков	Пробный эксперимент N17	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	4,7	0,2	Дл
5	30.09.2022 1...	Серия запусков	Пробный эксперимент N18	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	5,3	0,4	Дл
6	30.09.2022 1...	Серия запусков	Пробный эксперимент N19	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	6,1	0,6	Дл
7	30.09.2022 1...	Серия запусков	Пробный эксперимент N20	Не указана		dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	5,5	0,35	Дл
8	30.09.2022 1...	Серия запусков	Пробный эксперимент N21	Не указана		dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	5,5	0,35	Дл
9	30.09.2022 1...	Серия запусков	Пробный эксперимент N22	Не указана		dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	5,3	0,4	Дл
10	30.09.2022 1...	Серия запусков	Пробный эксперимент N23	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	6,3	0,4	Дл
11	30.09.2022 1...	Серия запусков	Пробный эксперимент N24	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	5,5	0,35	Дл
12	30.09.2022 1...	Серия запусков	Пробный эксперимент N25	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	5,3	0,4	Дл
13	30.09.2022 1...	Одиночный запуск	Пробный эксперимент N26	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	1	0	Дл
14	30.09.2022 1...	Одиночный запуск	Пробный эксперимент N27	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	1	0	Дл
15	30.09.2022 1...	Одиночный запуск	Создание таблицы N28	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	Без решения		10	10	20			Дл
16	30.09.2022 1...	Серия запусков	Пробный эксперимент N29	Не указана		dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	6,03	0,40	Дл
17	30.09.2022 1...	Серия запусков	Пробный эксперимент N30	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	5,2667	0,266666	Дл
18	30.09.2022 1...	Одиночный запуск	Пробный эксперимент N31	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	Без решения		10	10	20			Дл
19	30.09.2022 1...	Серия запусков	Пробный эксперимент (уб) N34	dbo_Cut_Create	@m=10,@n=10,@l=10,@m=10,@n=10	dbo_Cut_Decision	@m=10,@n=10,@l=10,@m=10	10	10	20	5,74	0,44	Дл
20	30.09.2022 1...	Загрузка из файла	Загрузка таблицы из файла N35	Не указана		Без решения		10	10	20			Дл
21	30.09.2022 1...	Серия запусков	Пробный эксперимент N36	Не указана		Без решения		10	10	20			Дл
22	07.10.2022 1...	Одиночный запуск	Пробный эксперимент N37	Не указана		dbo_Cut_Decision_2	@m=100,@n=100,@l=100,@m=100	100	100	1	10	47	Дл
23	07.10.2022 2...	Одиночный запуск	Пробный эксперимент N38	Не указана		Без решения		100	100	1			Дл
24	07.10.2022 2...	Одиночный запуск	Пробный эксперимент N39	Не указана		dbo_Cut_Decision_2	@m=100,@n=100,@l=100,@m=100	100	100	1	10	10102	Дл

Рисунок 2 – Интерфейс используемого ПО

2.6. Ссылка на репозиторий с исходным кодом

<https://github.com/Makksx/VKR2023>



3. ПЛАН РАБОТЫ НА ВЕСЕННИЙ СЕМЕСТР

3.1. Детализация постановки задачи на весенний семестр

Планирование и проведение экспериментов по сравнению эвристик, которые реализованы либо на SQL в среде MS SQL Server 2019, либо на Python, а также исследование полученных результатов.

3.2. Обоснование актуальности разработки

Множество практических задач опирается на задачу покрытия множеств: построение расписаний, расположение пунктов обслуживания, построение электронных схем и т.д.

Создание работоспособной и эффективной эвристики больше искусство, чем наука. Когда эвристика придумана, возникает необходимость оценить ее качество. Другого способа, кроме непосредственного сравнения этой эвристики с другими, не существует. Из чего следует важность научного исследования, основанного на качественном сравнении эвристик.

ЗАКЛЮЧЕНИЕ

В результате работы в течение семестра был реализован и исследован эвристический алгоритм генерации таблиц покрытия. Алгоритм, адаптированный под применение в решаемой задаче, был разработан на языке Python, а также внедрён как внешняя функция в Microsoft SQL Server и будет использоваться в дальнейших исследованиях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Dorigo M., Stützle T. Ant colony optimization: overview and recent advances //Handbook of metaheuristics. – 2019. – С. 311-351.
2. Blum C. Ant colony optimization: Introduction and recent trends //Physics of Life reviews. – 2005. – Т. 2. – №. 4. – С. 353-373.
3. Bonabeau E. et al. Swarm intelligence: from natural to artificial systems. – Oxford university press, 1999. – №. 1.
4. Glover F. Future paths for integer programming and links to artificial intelligence //Computers & operations research. – 1986. – Т. 13. – №. 5. – С. 533-549.
5. Еремеев А. В., Заозерская Л. А., Колоколов А. А. Задача о покрытии множества: сложность, алгоритмы, экспериментальные исследования //Дискретный анализ и исследование операций. – 2000. – Т. 7. – №. 2. – С. 22-46.
6. Goldberg D. E. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley, 1989.
7. Alexandrov D. ; Kochetov Yu. Behavior of the ant colony algorithm for the set covering problem // Operations Research Proceedings 1999 (Magdeburg, 1999). Berlin: Springer, 2000. P. 255-260.
8. Коновалов И. С., Остапенко С. С., Кобак В. Г. Сравнение эффективности работы точных и приближенных алгоритмов для решения задачи о покрытии множества //Advanced Engineering Research. – 2017. – Т. 17. – №. 3 (90).
9. Grossman T., Wool A. Computational experience with approximation algorithms for the set covering problem //European journal of operational research. – 1997. – Т. 101. – №. 1. – С. 81-92.
10. Дроздов С. Н. Комбинаторные задачи и элементы теории вычислительной погрешности //Таганрог: Изд-во ТРТУ. – 2000. – Т. 61.

11. Ramalhinho-Lourenço H., Pinto J. L., Portugal R. Metaheuristics for the bus-driver scheduling problem. – 1998.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

```
import random
import time
from collections import Counter
from functools import reduce
from math import sqrt

import pandas as pd

class TPGeneration:
    def __init__(self, n_cols=20, n_rows=10, vr=None):
        if vr is None:
            vr = [0.0, 0.3, 0.4, 0.3]
        if sum(vr) > 1:
            raise Exception("sum(vr) must be <=1")
        vrn = [0] + [int(i * n_cols) for i in vr]
        vrn[-1] = n_cols - sum(vrn) + vrn[-1]
        self.vrn = vrn
        self.n_cols = n_cols
        self.n_rows = n_rows
        self.result = pd.DataFrame({"cols": [set() for _ in
range(n_rows)], "cols_count": [0] * n_rows})
        self.candidates = [set(range(n_rows))]
        self.col_choice = list(range(n_cols))
        random.shuffle(self.col_choice)

    def pretty_df(self):
        new_df = pd.DataFrame({"Row_": [], "Col_": []})
        for i in range(len(self.result.cols)):
            for j in self.result.loc[i, "cols"]:
                new_df.loc[len(new_df)] = [i, j]
```

```

        return new_df

    def _update_candidates(self, new_candidates, new_candidate):
        if len(new_candidate) >= 2 and not any(j >= new_candidate for
j in new_candidates):
            new_candidates.append(new_candidate)

    def _post_process(self):
        self.result['cols_count'] = self.result['cols'].apply(len)

    def _decomposition(self, rows_cur):
        new_candidates = []
        need_cols_in_candidates = set(
            i for i in range(len(self.result.cols_count)) if not
pd.isnull(self.result.cols_count[i]))
        while self.candidates:
            candidate = next((i for i in self.candidates if not
i.isdisjoint(need_cols_in_candidates)), None)
            if not candidate:
                if len(new_candidates) >= sqrt(self.n_cols):
                    break
                else:
                    candidate = max(self.candidates, key=len)
            self.candidates.remove(candidate)
            if set(rows_cur) <= candidate:
                for el in rows_cur:
                    new_candidate = candidate.copy()
                    new_candidate.discard(el)
                    self._update_candidates(new_candidates,
new_candidate)
            else:
                self._update_candidates(new_candidates, candidate)

```

```

        need_cols_in_candidates = need_cols_in_candidates -
(reduce(lambda x, y: x | y,

new_candidates) if new_candidates else set())
        if (not need_cols_in_candidates) and (len(new_candidates)
>= sqrt(self.n_cols)):
            break
        # print(len(new_candidates), len(self.col_choice))
        self.candidates = new_candidates

    def _fill_rows(self, rows_cur):
        i_cur = self.col_choice.pop(0)
        self.result.loc[rows_cur, 'cols_count'] += 1
        self.result.loc[rows_cur, 'cols'].apply(lambda x:
x.add(i_cur))

    def _choose_rows(self, marks_count):
        while True:
            str_need = self.result.cols_count.idxmin()
            candidates = [list(i) for i in self.candidates if str_need
in i]

            if candidates:
                break
            self.result.loc[str_need, 'cols_count'] = None
            rows = next((i for i in candidates if len(i) >= marks_count),
None)

            if rows is None:
                return max(candidates, key=len)
            random.shuffle(rows)
            rows.remove(str_need)
            rows = [str_need] + rows
            return rows[0:marks_count]

```

```

def generate(self):
    for marks_count in range(len(self.vrn)):
        for _ in range(self.vrn[marks_count]):
            if not self.candidates:
                break
            random.shuffle(self.candidates)
            rows_cur = self._choose_rows(marks_count)
            self._fill_rows(rows_cur)
            if self.col_choice:
                self._decomposition(rows_cur)
        self._post_process()
    return self.result

gen = TPGeneration(n_cols=100, n_rows=100, vr=[0.0, 0.1, 0.1, 0.1,
0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1])
time_start = time.time()
print(gen.generate())
print("time", time.time() - time_start)
gen.pretty_df().to_csv('result.csv', header=None, index=False)

```