Оглавление

Задача

Задача о назначениях

Имеется n работ и n исполнителей. Задана матрица стоимостей (размера $n \times n$) выполнения каждой работы тем или иным исполнителем. Необходимо распределить исполнителей по работам так, чтобы суммарная стоимость выполнения была минимальной.

Входные данные: матрица стоимостей.

Выходные данные: номера исполнителей для работ с номерами от 1 до n.

Данную задачу можно формализовать следующим образом. Пусть c_{ij} — стоимость назначения работника i на задачу j. Тогда задача сводиться к следующей форме:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij},$$

при условиях:

$$\begin{aligned} \sum_{i=1}^{n} x_{ij} &= 1, \quad \forall j = 1, 2, \ldots, n; \\ \sum_{j=1}^{n} x_{ij} &= 1, \quad \forall i = 1, 2, \ldots, n; \\ x_{ij} \in \{0, 1\}&, \quad \forall i, j = 1, 2, \ldots, n \end{aligned}$$

где $x_{ij} = 1$, если работник i назначен на задачу j, и 0 в противном случае.

Введение

В данной работе рассматриваются два подхода к решению задачи о назначениях: Венгерский алгоритм и симплекс-метод. Венгерский алгоритм, основанный на работе венгерских математиков, решает задачу о назначениях за полиномиальное время. Симплекс-метод, в свою очередь, предоставляет эффективное решение задач линейного программирования, основываясь на переходах между экстремальными точками множества решений. Целью данной работы является изучение алгоритмов и сравнение их реализации на языках программирования C++ и MATLAB, а также проведение эмпирической оценки их асимптотической сложности.

1. Венгерский алгоритм

Данный алгоритм, впервые описанный Г. Куном в 1955 году в [1], назван в честь двух венгерских математиков Д. Кенига и Й. Эгервари, на чьей работе он основан.

В 1957 году Д. Манкрес доказал в [2], что алгоритм имеет строго полиномиальную сложность. Позднее он стал известен как алгоритм Куна — Манкреса. Исходно временная сложность была оценена как $O(n^4)$, но работа Д. Эдмондса и Р. Карпа продемонстрировала, что его можно модифицировать, достигнув времени выполнения $O(n^3)$. Эта модификация получила название алгоритм Хопкрофта — Карпа. В 2006 году в [3] было отмечено, что решение задачи о назначениях было предложено Якоби еще в XIX веке и опубликовано на латыни в посмертном сборнике его трудов в 1890 году.

1.1. Реализация за $O(n^4)$

Дальнейшие рассуждения во многом основаны на статье [4]. Пусть $A[1 \dots n][1 \dots n]$ — матрица стоимостей (для универсальности индексацию будем начинать с единицы). Два произвольных масива чисел $u[1 \dots n]$ и $v[1 \dots n]$ будем называть **потенциалом**, если выполняется следуюущее неравенство:

$$u[i] + v[j] \leq A[i][j]. \qquad \forall i, j \in [1 \dots n] \tag{1}$$

Число $f = \sum\limits_{i=1}^{n} u[i] + \sum\limits_{i=1}^{n} v[i]$ назовем **значением потенциала**. Очевидно, что в силу выражения (1) для искомого решения sol и любого значения потенциала f верно неравенство: $f \leq sol$. Венгерский алгоритм предполагает, что всегда существуют такое решение и потенциал, что $f = sol$. Позднее приведем доказательство этого факта.

Зафиксируем некоторый потенциал. Назовем ребро (i, j) **жестким**, если верно:

$$u[i] + v[j] = A[i][j].$$

Сформулируем задачу о назначениях с помощью двудольного графа. Пусть H — двудольный граф, составленный только из жёстких рёбер, а M — максимальное по количеству рёбер паросочетание графа H. Напомним, что **мощностью** паросочетания называют количество рёбер в нём, а **максимальным** паросочетанием паросочетание, мощность которого максимальна среди всех возможных паросочетаний в данном графе [5].

Ниже изложен венгерский алгоритм. Положим потенциал равным нулю, т.е. $u[i] = v[i] = 0$, а паросочетание M пустым. Далее, на каждом шаге алгоритма пытаемся, не меняя потенциала, увеличить мощность текущего паросочетания M на единицу. Для этого фактически используется обычный алгоритм Куна поиска максимального паросочетания в двудольных графах [6], основанным на теореме К. Бержа впервые описанной им в 1957 году в [7]. Кратко опишем данный алгоритм.

Все рёбра паросочетания M ориентируются по направлению от правой доли к левой, все остальные рёбра графа H ориентируются в противоположную сторону. В терминологии поиска паросочетаний вершину принято называть **насыщенной**, если ей смежно ребро из текущего паросочетания. Вершина, которой не смежно ни одно ребро из текущего паросочетания, называется **ненасыщенной**. Путь нечетной длины в котором рёбра поочередно принадлежат/не принадлежат паросочетанию и оба конца не принадлежат паросочетанию — называется **дополняющим путём**.

Запустим обход в глубину (DFS) из всех ненасыщенных вершин левой доли. Если в результате обхода удалось достигнуть ненасыщенной вершины правой доли, то это означает, что мы нашли дополняющий путь из левой доли в правую. Если "инвертировать"найденный путь, то мощность паросочетания увеличиться на единицу. В случае, если дополняющий путь не был найден, то текущее паросочетание M — максимально в графе H при данном потенциале.

Произведем пересчет потенциала, чтобы в дальнейшем увеличить паросочетание. Пусть L и R — посещенные алгоритмом Куна вершины левой и правой доли соответсвенно. Рассчитаем величину Δ следующим образом:

$$\Delta = \min_{i \in L, j \notin R} \{A[i][j] - u[i] - v[j]\}. \tag{2}$$

Покажем, что $\Delta > 0$. Предположим, что $\Delta = 0$. Тогда существует жёсткое ребро (i, j), причём $i \in L$ и $j \notin R$. Следовательно, ребро (i, j) должно было быть ориентированным от второй доли к первой, т.е. это жёсткое ребро (i, j) должно входить в паросочетание M. Однако это невозможно, т.к. мы не могли попасть в насыщенную вершину i, кроме как пройдя по ребру из j в i. Пришли к противоречию, значит, $\Delta > 0$.

Произведем пересчет потенциала следующим образом: для всех вершин $i \in L$ сделаем $u[i] = u[i] + \Delta$, а для всех вершин $j \in R$ — сделаем $v[j] = v[j] - \Delta$. Докажем, что полученный потенциал останется корректным, т.е. что по-прежнему для всех i и j выполняется: $u[i] + v[j] \leqslant A[i][j]$. Для случаев, когда $i \in L, j \in R$ или $i \notin L, j \notin R$ — это так, поскольку для них сумма $u[i]$ и $v[j]$ не изменилась. Когда $i \notin L, j \in R$ — неравенство только усилилось. Наконец, для случая $i \in L, j \notin R$ — левая часть неравенства увеличивается, однако неравенство всё равно сохраняется, поскольку величина Δ, как видно по её определению является максимальным увеличением, не приводящим к нарушению неравенства.

Отметим, что все ребра паросочетания M останутся жесткими, так как для этого необходимо, чтобы равенство $u[i] + v[j] = A[i][j]$ превратилось в неравенство $u[i] + v[j] < A[i][j]$. Однако левая часть могла уменьшиться только в одном случае: когда

$i \notin L, j \in R$, но при $i \notin L$ ребро (i, j) не может быть ребром паросочетания.

Докажем, что при каждом описанном изменении потенциала количество вершин, достижимых при обходе строго увеличивается и не более чем за n пересчётов будет найден дополняющий путь и мощность паросочетания будет увеличена. Во-первых, любая вершина, которая была достижимой до пересчета, достижимой и останется. В самом деле, если некоторая вершина достижима, то до неё есть некоторый путь из достижимых вершин, начинающийся в ненасыщенной вершине левой доли, поскольку для рёбер вида (i, j), $i \in L, j \in R$ сумма $u[i] + v[j]$ не меняется, то данный путь полностью сохранится после пересчета потенциала. Во-вторых, покажем, что в результате пересчёта потенциала появилась хотя бы одна новая достижимая вершина. По определению Δ: ребро (i, j), на котором был достигнут минимум, теперь станет жёстким, а, значит, вершина j станет достижимой благодаря этому ребру и вершине i.

В худшем случае должно произойти n увеличиний паросочетаний, перед каждым из которых происходит не более n пересчетов потенциалов, каждый из которых осуществляется за $O(n^2)$.

1.2. Реализация за $O(n^3)$

Однако, как было отмечено ранее Д. Эдмондсом и Р. Карпом была разработана реализация данного алгоритма за $O(n^3)$. Это может быть достигнуто за счет рассмотрения не всей матрицы сразу, а ее строк одну за другой. Модифицированный алгоритм имеет вид

1) добавляем в рассмотрение очередную строку матрицы A;
2) пересчитываем потенциал, пока нет дополняющего пути, начинающегося в этой строке;
3) при появлении дополняющего пути, чередуем паросочетание (включая тем самым последнюю строку в паросочетание), и переходим к рассмотрению следующей строки.

Опишем шаги 2 и 3. Как было показано ранее при изменении потенциала вершины, которые были достижимы обходом Куна, достижимыми и останутся, а также дополняющий путь для увеличения мощности паросочетания может быть найден не более чем за n пересчетов. Следовательно, для поиска дополняюшего пути нет необходимости запускать обход Куна с начала после каждого пересчёта потенциала. Каждый раз, когда потенциал пересчитывается, добавленные жесткие ребра просматриваются, и если их левые ребра достижимы, то правые ребра также помечаются как достижимые, и обход продолжается уже из них.

Развивая эту идею дальше, можно прийти к такому представлению алгоритма: это цикл, на каждом шаге которого сначала пересчитывается потенциал, затем находится вершина, ставшая достижимой (а таковая всегда найдётся, поскольку после пересчёта потенциала всегда появляются новые достижимые вершины), и если эта вершина была ненасыщена, то найден дополняющий путь, а если насыщена — то соответствующая ей в паросочетании вершина также становится достижимой.

Введем вспомогательные минимумы по каждому из столбцов j, чтобы производить пересчет потенциалов за $O(n)$:

$$m[j] = \min_{i \in L}\{A[i][j] - u[i] - v[j]\}.$$

Таким образом выражение (2) для поиска Δ выражается следущим образом:

$$\Delta = \min_{j \notin R} m[j].$$

Необходимо обновлять массив $m[1 \dots n]$ при появлении новых посещенных строк и при пересчете потенциала.

1.3. Псевдокод реализации за $O(n^3)$

Ниже приведен псевдокод описанного выше алгоритма. При его написании использовались рассуждения приведенные в [8]. В данном коде

- L^+ и R^+ — посещенные посещенный вершины левой и правой доли соответственно;
- L^- и R^- — непосещенные вершины левой и правой доли;
- $p[0 \dots n]$ — массив паросочетания. Для каждого стобца $i = 0, \dots n$ храниться номер соответствующей выбранной строки $p[i]$ (или 0, если строка не назначена). Полагаем, что $p[0]$ равно номеру рассматриваемой строки;
- $m[1 \dots n]$ — массив, хранящий для каждого столбца j вспомогательные минимумы, необходимые для быстрого пересчета потенциала.

function HungarianAlgorithm(A[1 ... n][1 ... n])
    for $v \in L$ **do**
        $v \in L^+$, $L \setminus \{v\} \in L^-$
        $R^- \leftarrow R$
        $m[j] \leftarrow C[v][j]$
        while True **do**
            $\Delta, (\tilde{i}, \tilde{j}) \leftarrow \min_{j \in R^+} m[j]$ ▷ Минимум достигается на $(\tilde{i}, \tilde{j})$
            $u[i] \leftarrow u[i] + \Delta, i \in L^+$

$v[j] \leftarrow v[j] - \Delta, j \in R^+$
$m[j] \leftarrow m[j] - \Delta$
$R^+ \leftarrow R^+ \cup \{\tilde{j}\}$
if $p[\tilde{j}] = \varnothing$ **then** ▷ Проверка является ли вершина насыщеннной
    Инвертируем дополняющий путь
    break
else
    $L^+ \leftarrow L^+ \cup \{p[\tilde{j}]\}$
    $m[j] \leftarrow \min\{m[j], A[p[\tilde{j}]][j] + u[p[\tilde{j}]] + v[j]\}$
end if
end while
end for
end function

2. Симплекс-метод

В данном разделе будет кратко описана идея симплекс-метода и подробно разобраны применяемые в его работе вычислительные процендуры. Используемая терминология заимствованна из [9]. Также использованы идеи описанные в [10].

В ходе решения задач линейного програмированния можно использовать графический метод. Он предполагает геометрическое представление допустимых решений с учетом всех ограничений модели. Полученное таким образом пространство решений — n-мерный тетраэдр (или симплекс). В каждой точке, принадлежащей внутреннней области или границам тетраэдра, все ограничения выполняются, поэтому решения, соответствующие этим точкам, являются **допустимыми**. Как известно [9], оптимальному решению всегда может быть поставлена **угловая** точка пространства. Следовательно, можно найти оптимальное решение, рассматривая лишь конечное число угловых точек. Данная идея положена в основу симплекс-метода, начиная с некоторой исходной допустимой угловой точки, осуществляются последовательные переходы от одной допустимой экстремальной точки к другой до тех пор, пока не будет найдена точка, соответствующая оптимальному решению.

Будем считать, что линейная модель стандартной формы содержит m уравнений и n неизвестных, тогда, как показано в [9], все допустимые эстремальные точки определяются, как все однозначные неотрицательные решения системы m уравнений, в которой $m-n$ переменных равны нулю. Такие решения будем называть **базисными**. Кроме того, как известно из [9], все смежные экстремальные точки отличаются лишь одной переменной, таким образом каждую последуюущую точку можно определить путем замены одной из базисных перменных на текущую небазисную переменную. Неизвестную, которая будет включена в множество базисных на следующей итерации, называют **включаемой**, а ту которая подлежит исключение — **исключаемой.**

2.1. Вычислительные процедуры симплекс-метода

В начале, исходная задача должна быть приведена к **стандартной форме**, а именно

1) все ограничения должны быть записаны в виде равенств с неотрицательной правой частью;
2) значения всех переменных модели неотрицательны;
3) целевая функция подлежит максимизации или минимизации.

В случае, если ограничения заданы неравенствами, они преобразуются в равенства путем введения дополнительньнных переменных, однако в рассматриваемой задаче это не требуется. Процесс приведения задачи к стандартной форме подробно описан в [9].

Нахождение исходного базиса

Рассмотрим вопрос о выборе исходного базиса. Представим ограничения в задаче в матричном виде, далее в ней необходимо найти столбец, который соответствует единичному вектору. Если такой столбец найден, соответствующая переменная становится базисной. В случае, когда в матрице нет подходящего столбца прибегают к преобразованию какого-либо столбца в единичный, поделив всю строку на ненулевой элемент этого столбца или используют метод исключения Гаусса [11]. В случае, если после преобразований, некоторые из правых частей b_i стали отрицательными, необходимо умножить соответствующие строки на -1.

Построение симплекс-таблицы

Пример симплекс-таблицы можно увидеть в Табл. 1. Опишем процесс ее построения. Запишем в первой строке неизвестные переменные $x_1, \ldots, x_n$. Вторую строку заполним коэффициентами $c_1, \ldots, c_n$ целевой функции. Затем построчно перечисляются базисные переменные и коэффициенты ограничений $a_{11}, \ldots, a_{1n}, \ldots, a_{mn}$, а также столбец правых частей $b_1, \ldots, b_m$.

Столбец симплекс-таблицы ассоциированный с вводимой переменнной будем называть **ведущим**. Строку, соответсвующую исключаемой переменной, назовем **ведущей**.

Таблица 1. Симплекс-таблица

Базисные переменные	x_1	x_2	$\cdots$	x_n	b
z	c_1	c_2	$\cdots$	c_n	0
x_{e_1}	a_{11}	a_{12}	$\cdots$	a_{1n}	b_1
x_{e_2}	a_{21}	a_{22}	$\cdots$	a_{2n}	b_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_{e_m}	a_{m1}	a_{m2}	$\cdots$	a_{mn}	b_m

Проверка на оптимальность

Сформулируем условие оптимальности решения [12]. Введем оценку

$$\Delta_i = \sum_{j=1}^{m} c_j a_{ij} - c_i.$$

В случае минимазации (максимизации) функции, текущее решение считается оптимальным, если все Δ_i неположительны (неотрицательны).

Переход к более оптимальному решению

В случае, если текущее решение оказывается оптимальным, то алгоритм завершает работу. В противном случае, выбирается новый ведущий столбец и ведущая строка. Ведущим выбирается столбец имеющий наибольшую оценку Δ_i. Переменная соответвутющая данному столбцу становиться включаемой. Далее, для каждой строки вычисляется отношение ее правой части, к соответвующему значению ведущего столбца. Строка имеющая минимальное соотношение становиться ведущей. Учитываются лишь положительные соотношения, если таких отношений нет, то алгоритм останавливает свою работу, так как целевая функция неограничена и решения не существует. После чего решение проверяется на оптимальность, и алгоритм продолжается до тех пор пока не будет найдено оптимальное решение.

3. Особенности реализации на языке C++

3.1. Венгерский алгоритм

Опишем решения, использованные при реализации Венгерского алгоритма на языке программирования C++. Основными структурами данных в реализации являются следующие.

Матрица затрат `a` — двумерный вектор `std::vector<std::vector<double>>`, в котором хранятся значения стоимости назначения каждого работника на каждую задачу. Размер этой матрицы равен $n \times n$, где n — количество работников и задач.

Вектор потенциалов строк `u` и вектор потенциалов столбцов `v` — представлены в виде одномерных векторов `std::vector<double>` размерности $n + 1$. Изначально все элементы этих векторов инициализируются нулями, что соответствует начальной пустой конфигурации.

Массив паросочетаний `p` — вектор `std::vector<int>` размера $n + 1$, в котором для каждого столбца j (где $1 \le j \le n$) хранится номер строки i, такой что строка i назначена на столбец j. Если для столбца j нет назначенной строки, то `p[j] = 0`. Элемент `p[0]` используется для хранения текущей строки, рассматриваемой в процессе алгоритма.

Массив минимальных затрат `minv` — вектор `std::vector<double>` размера $n+1$, который содержит минимальные значения для каждого столбца, необходимые для быстрого пересчета потенциалов.

Массив предшествующих столбцов `way` — вектор `std::vector<int>` размера $n + 1$, который содержит информацию о том, где эти минимумы достигаются, чтобы впоследствии восстановить увеличивающую цепочку. Можно предположить, что для каждого столбца в массиве `way[]` необходимо хранить номер строки, а также создать дополнительный массив, где для каждой строки будет записан номер столбца, из которого был выполнен переход. Однако можно заметить, что алгоритм Куна всегда попадает в строки, проходя по ребру паросочетания из столбцов, поэтому номера строк для восстановления цепочки всегда можно взять из паросочетания. Таким образом, `way[j]` для каждого столбца j содержит номер предшествующего столбца (или 0, если такого нет).

Переменная `INF` — константа, представляющая бесконечность, определенная как `std::numeric_limits<double>::max()`, используемая для инициализации значений массива минимальных затрат `minv`.

В коде используются несколько ключевых функций для реализации алгоритма.

Функция `hungarianAlgorithm` — основная функция, реализующая алгоритм. Она принимает на вход матрицу затрат, инициализирует вспомогательные массивы, затем выполняет итерации алгоритма, добавляя новые строки в паросочетание и обновляя потенциалы. Возвращает минимальную стоимость оптимального назначения.

Функция `main` — инициализирует входные данные, такие как размер матрицы и значения затрат, и вызывает функцию `hungarianAlgorithm`. После выполнения алгоритма выводит на экран результат — минимальную стоимость назначения всех работников на задачи.

Алгоритм строится вокруг внешнего цикла по строкам матрицы. Внутренний цикл `do-while(p[j0] != 0)` выполняется до тех пор, пока не будет найден свободный столбец j_0. В ходе каждой итерации внутреннего цикла

1) новый столбец j_0 помечается как посещённый новый столбец j_0;
2) определяется строка i_0, смежная этому столбцу в текущем паросочетании ($i_0 = p[j_0]$). Если $j_0 = 0$, то берётся текущая рассматриваемая строка i;
3) обновляется массив `minv` для пересчёта минимальных значений. В этот момент также находится минимальное значение `delta` и столбец j_1, где это значение достигается;
4) если `delta` оказывается равным нулю, это означает, что потенциалы можно оставить неизменными, так как уже есть доступный столбец;
5) производится пересчёт потенциалов `u` и `v`, а также соответствующее обновление массива `minv`.

По завершении внутреннего цикла `do-while` находится увеличивающая цепочка, которая оканчивается в столбце j_0. Эту цепочку можно восстановить, используя массив `way`.

3.2. Симплекс-метод

Кратко опишем решения использованные при реализации алгоритма. Далее перечислим основные структуры данных.

Матрица стоимости `cost` — вектор `std::vector<std::vector<double>>`, в котором хранятся значения стоимости назначения каждого работника на каждую задачу.

Симплекс-таблица `table` — представлена как вещественный двумерный вектор `std::vector<std::vector<double>>`. В таблице хранятся коэффициенты целевой функции и ограничения, необходимые для перехода к оптимальному решению. Количество строк симплекс-таблицы равно $2n + 1$, где n — это количество работников и задач. Строки с 1 по n представляют ограничения для каждой задачи, т.е. каждый работник может быть назначен только на одну задачу. Строки с $n + 1$ по $2n$ представляют ограничения для каждого работника, т.е. каждая каждый работник может сделать только одну задачу.

Базисные переменные `basis` — вектор `std::vector<int>` размера $2n$, который содержит индексы текущих базисных переменных симплекс-метода. Изначально все элементы массива равны -1, что обозначает, что базисные переменные ещё не выбраны. Связь индексов массива *basis* с переменными можно выразить так: элемент `basis`$[\mathtt{k}] = p$ означает, что переменная x_{ij}, которая соответствует назначению работника i, на задачу j, является базисной на текущем этапе симплекс-метода. Значение p

в массиве соответствует переменной x_{ij}, где $i = \frac{p}{n}$ и $j = p \mod n$. Индекс k означает, что в строке симплекс-таблицы, соответствующей индексу $k+1$, находится уравнение, в котором переменная с индексом p является базисной.

В коде используются несколько ключевых функций для выполнения симплекс-метода.

Функция `printTable` — предназначена для вывода текущего состояния симплекс-таблицы и базисных переменных на каждом шаге алгоритма. Это необходимо для наглядного представления процесса итераций симплекс-метода и контроля за изменением таблицы.

Функция `findPivotColumn` — осуществляет выбор ведущего столбца, который определяет переменную, вводимую в базис на данной итерации. Столбец с наибольшей дельтой, приводящий к улучшению значения целевой функции, становится ведущим.

Функция `findPivotRow` — определяет ведущую строку на основе минимального положительного отношения свободного члена к значению в разрешающем столбце. Это позволяет выбрать переменную, которая будет выводиться из базиса.

Функция `pivot` — выполняет симплекс-поворот. Она обновляет симплекс-таблицу, деля строку ведущего элемента на его значение, а затем корректирует остальные строки. Это позволяет обновить систему уравнений, что необходимо для перехода к следующей итерации.

Функция `solveAssignmentProblem` — это основная функция алгоритма. Она инициализирует симплекс-таблицу и базисные переменные, заполняя начальные значения на основе входных данных. Далее с помощью итераций симплекс-метода происходит нахождение оптимального решения или выявляется, что решение не существует.

4. Особенности реализации на языке MATLAB

4.1. Венгерский алгоритм

Реализация алгоритма аналогична версии, описанной в соответствующем разделе для языка C++, но адаптирована для использования в среде MATLAB. Кратко опишем её основные компоненты.

Основной структурой данных является матрица стоимости `a`, представляющая собой двумерный массив размером $n \times n$, где n — это количество объектов и назначений.

Для решения задачи о назначениях используется несколько вспомогательных массивов

- векторы потенциалов строк и столбцов, `u` и `v` соответственно;

- вектор текущих назначений `p`, который хранит текущее паросочетание;
- вектор `way`, используемый для хранения обратных путей, позволяющих восстановить оптимальное назначение.

Алгоритм состоит из нескольких ключевых этапов. На первом этапе происходит инициализация вспомогательных массивов, включая задание начальных значений для потенциалов и векторов назначений. Затем для каждой строки матрицы стоимостей строится минимальный путь, минимизирующий общую стоимость назначения с учетом текущих потенциалов. В процессе построения пути обновляются минимальные возможные улучшения для каждого столбца, которые хранятся в массиве `minv`, а также осуществляется проверка на использование столбцов с помощью массива `used`.

После нахождения оптимального пути потенциалы строк и столбцов корректируются, чтобы учесть найденное минимальное улучшение. На заключительном этапе производится восстановление оптимального пути назначения путем обратного прохода по массиву `way`, что позволяет определить окончательное соответствие между строками и столбцами.

Результатом выполнения алгоритма является минимальная стоимость назначения, которая вычисляется как `-v[0]`.

4.2. Симплекс-метод

Реализация алгоритма схожа с описанной в соответствующем разделе для C++. Однако, кратко опишем ее.

Основной структурой данных является матрица стоимости `cost`, представляющая собой двумерный массив размером $n \times n$, где n — это количество работников и задач. Элементы этой матрицы задаются пользователем и отражают стоимость назначения конкретного работника на задачу. Заполнение массива происходит через ввод строк с элементами стоимости, которые затем преобразуются в числовой формат с использованием функции `str2num`.

Симплекс-таблица `table` представляет собой расширенную матрицу, содержащую данные для симплексных вычислений. Размер таблицы — $(2n+1) \times (n^2+1)$. Первая строка таблицы хранит коэффициенты целевой функции, которые рассчитываются на основе стоимости назначения. Остальные строки представляют ограничения задачи, разделённые на два набора: первый отвечает за ограничения для работников, а второй — для задач.

Для хранения информации о базисных переменных используется вектор `basis` длиной $2n$. В нём хранятся индексы переменных, которые на каждой итерации симплекс-метода входят в базис.

Алгоритм состоит из нескольких функций, каждая из которых выполняет конкретную задачу в рамках симплекс-метода.

Функция `Simplex()` — основной интерфейс взаимодействия с пользователем. Она отвечает за ввод данных о размере задачи и стоимости назначений. После получения данных вызывается функция `solveAssignmentProblem()` для решения задачи назначения.

Функция `solveAssignmentProblem()` инициализирует симплекс-таблицу и базисные переменные, а также выполняет итерации симплекс-метода. Она строит начальный базис, заполняет ограничения в симплекс-таблице и запускает цикл итераций до нахождения оптимального решения или вывода о неограниченности задачи.

Функция `findPivotColumn()` определяет ведущий столбец.

Функция `findPivotRow()` находит ведущую строку.

Функция `pivot()` выполняет симплексное вращение, обновляя симплекс-таблицу и базисные переменные. В результате происходит переход к новому базису, что улучшает решение на каждом шаге.

Функция `printTable()` выводит текущее состояние симплекс-таблицы и базисных переменных на каждой итерации, что позволяет отслеживать процесс поиска оптимального решения.

5. Ассимптотическая оценка

В разделе 1 был подробно описан Венгерский алгоритм с обоснованием его ассимптотической сложности, поэтому в данном разделе для него будут приведены лишь экспериментальное время выполнения.

5.1. Симплекс-метод

Рассмотрим сложность симплекс-метода для задачи о назначениях. Оценим основные этапы метода.

Инициализация симплекс-таблицы

Создание симплекс-таблицы размера $(2N+1) \times (N^2+1)$ требует $O(N^3)$ времени и памяти.

Функция `findPivotColumn`

Определение разрешающего столбца требует прохождения по всем элементам таблицы. Это занимает $O(N^3)$ времени для каждого вызова, так как необходимо вычислить значение Δ для каждого столбца.

Функция `findPivotRow`

Определение разрешающей строки требует перебора всех строк симплекс-таблицы. Это занимает $O(N)$ времени для каждого вызова.

Функция `pivot`

Выполнение симплекс-поворота требует изменения всех строк симплекс-таблицы. В худшем случае это требует $O(N^2)$ операций, так как нужно обновить все элементы каждой строки таблицы.

Основной цикл симплекс-метода

Данный цикл имеет экспоненциальную сложность в худшем случае, так как число рассматриваемых базисов может достигать $C_{N^2}^{2N}$.

Однако, как описано в [13] возможна модификация симплекс-метод, при которой метод имеет полиномиальную сложность для задачи о назначениях с целочисленными коэффициентами. Пусть $\tilde{C} = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$, а x^0 и x^* — начальное и оптимальное решение соответсвенно. Тогда при данной реализации будет рассмотрено не более чем $n^3 \ln(\tilde{C}(x^0) - \tilde{C}(x^*))$ базисов прежде, чем будет найдено оптимальное решение.

Ассимптотическая сложность симлпекс-метода для различных задач и входных данных подробно описана, например, в [14].

5.2. Эмпирический анализ времени выполнения

В Табл. 2 представлено сравнение времени работы венгерского алгоритма и симплекс-метода в зависимости от размеров матрицы стоимостей. В Табл. 3,4 более наглядно представлена зависимость времени работы от размера входных данных. На их основа можно сделать однозначный вывод: Венгерский алгоритм превосходит симплекс-метод по времени выполнения при решении задачи, а также более предсказуем.

Таблица 2. Сравнение времени работы алгоритмов для разных размеров матриц

Размер матрицы	**Венгерский алгоритм (с)**	**Симплекс-метода (с)**
32×32	<0.01	9.24
64×64	<0.01	15.68
128×128	<0.01	33.51
256×256	0.02	1020.01

Таблица 3. Венгерский алгоритм

Размер матрицы	**Венгерский алгоритм (с)**
400×400	0.09
800×800	1.71
1600×1600	9.41
3200×3200	58.95

Таблица 4. Симплекс-метод

Размер матрицы	**Симплекс метод (с)**
50×50	0.33
75×75	2.13
112×112	10.98
168×168	89.11

Заключение

В ходе проведенной работы были изучены и реализованы: Венгерский алгоритм и симплекс-метод. Оба метода доказали свою эффективность, однако имеют существенные различия. Венгерский алгоритм показал высокую скорость решения задачи о назначениях, что делает его предпочтительным для нее. Его асимптотическая сложность и практическая производительность позволяют решать задачи большой размерности.

Симплекс-метод, несмотря на гораздо более высокую вычислительную сложность, обладает значительным преимуществом в универсальности. Он применим к широкому классу задач линейного программирования и может быть использован для решения различных оптимизационных проблем с множеством ограничений.

Список использованных источников

1. Harold W.K. The Hungarian Method for the assignment problem // Naval Research Logistics Quarterly. — 1955. — №2. — P. 83-97.
2. Munkres J. Algorithms for the Assignment and Transportation Problems // Journal of the Society for Industrial and Applied Mathematics. — 1957. — №5. — P. 32-38.
3. Borchardt C.W., Jocobi C.G.J. About the research of the order of a system of arbitrary ordinary differential equations // Journal für die reine und angewandte Mathematik. — 2006. — №64. — P. 297-320.
4. Венгерский алгоритм решения задачи о назначениях // MAXimal URL: http://e-maxx.ru/algo/assignment_hungary (дата обращения: 28.08.2024).
5. Паросочетания // Алгоритмика URL: https://algorithmica.org/ru/matching (дата обращения: 28.08.2024).
6. Алгоритм Куна для поиска максимального паросочетания // ИТМО URL: https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Куна_для_поиска_максимального_паросочетания (дата обращения: 28.08.2024).
7. Berge C. Two theorems in graph theory // Proceedings of the National Academy of Sciences of the United States of America. — 1957. — V. 43, №9. — P. 842-844.
8. Венгерский алгоритм решения задачи о назначениях // ИТМО URL: https://neerc.ifmo.ru/wiki/index.php?title=Венгерский_алгоритм_решения_задачи_о_назначениях (дата обращения: 8.09.2024).
9. Таха Х. Введение в исследование операций: В 2 т. — М.: Мир, 1985. — 479 с. — Т. 1
10. Вагнер Г. Основы исследования операций: В 3 т. — М.: Мир, 1972. — 335 с.
11. Канатников А.Н., Крищенко А.П. Аналитическая геометрия. — 10-е — М.: Изд-во МГТУ им. Н. Э. Баумана, 2024. — 374 с.
12. Губарь Ю.В. Введение в математическое программирование. — 3-е — М.: ИНТУИТ, 2021. — 225 с.
13. Hung S. A Polynomial Simplex Method for the Assignment Problem // Operations Research. — 1983. — V. 31, №3. — P. 595-600.
14. Borgwardt K.H. The Simplex Method: A probabilistic analysis. — Berlin: Springer-Verlag, 1987. — 270 p.