Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Робототехника и комплексная автоматизация*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ

«Задача линейного упорядочивания: области применения и способы решения»

Студент РК6-41М — А. С. Антонов
(Группа) (подпись, дата) (инициалы и фамилия)

Руководитель — А. Н. Божко
(подпись, дата) (инициалы и фамилия)

Москва, 2025 г

Оглавление

1. ВВЕДЕНИЕ

Задача линейного упорядочения (Linear Ordering Problem, LOP) занимает центральное место в комбинаторной оптимизации благодаря своей универсальности и широкому спектру приложений. В условиях роста объема данных [1] и потребности в эффективном управлении сложными системами LOP становится инструментом для решения задач ранжирования, планирования и анализа зависимостей.

Гэри и Джонсон (1979) продемонстрировали, что LOP является NP-сложной задачей. Однако, благодаря её многочисленным применениям в различных областях, таких как археология (Гловер и др., 1972), экономика (Леонтьев, 2008), теория графов (Харон и Худри, 2007), машинный перевод (Тромб и Эйснер, 2009) мы можем найти большое количество работ, в которых бы LOP решался с помощью точных, эвристических и метаэвристических методов.

В экономике LOP используется для ранжирования инвестиционных проектов на основе их потенциальной доходности и рисков. В машинном обучении задача помогает строить консенсусные рейтинги в рекомендательных системах [2]. В спортивной аналитике с её помощью определяют рейтинги команд, учитывая исторические результаты матчей [3]. В биоинформатике LOP применяется для упорядочения геномных последовательностей, что критично для понимания эволюционных процессов [4]. В археологии LOP нашёл применение в задаче стратификации для определения наиболее вероятного хронологического порядка образцов, найденных в разных местах. Матрица, описывающая эту проблему, известна как матрица Харриса.

Рост интереса к Big Data и искусственному интеллекту усиливает потребность в алгоритмах, способных работать с высокоразмерными данными, что делает исследование LOP особенно актуальным.

Цель работы состоит в изучении теоретических основ задачи линейного упорядочения, исследовании областей применения и проведении сравнительного анализа методов её решения.

Работа включает в себя пять разделов. В теоретической части раскрывается постановка задачи и её связь с другими проблемами оптимизации. Далее анализируются

методы решения, а в практическом разделе рассматриваются области применения данной задачи.

2. ПОСТАНОВКА ЗАДАЧИ

Для заданной матрицы B размера $n \times n$ задача линейного упорядочения (LOP), формулируется, как задача нахождения перестановки строк и столбцов π, которая бы максимизировала функцию $f(\pi)$ в формуле 1.

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} B_{\sigma(i)\sigma(j)}, \quad (1)$$

где $\sigma(i)$ обозначает индекс строки (и столбца), занимающей позицию i в решении σ. Иными словами, цель заключается в нахождении такой перестановки строк и столбцов матрицы C, которая бы максимизировала сумму элементов, находящихся выше главной диагонали. В данной постановке задача линейного упорядочивания известна, как triangulation problem of input-output matrices. Тем не менее существуют и альтернативные варианты представления LOP. Так Марти и Рейнельт в своей книге 2011-го года «The linear ordering problem: exact and heuristic methods in combinatorial optimization» дают интерпретацию в терминах теории графов с поиском ациклического турнира (полный ориентированный ациклический подграф), максимизирующего сумму весов дуг полного ориентированного графа.

Сами значения $B_{i,j}$ отображают силу предпочтения i-го элемента j-ому.

Рассмотрим пример для $n = 5$, который будет использоваться в дальнейшем. На рисунке 1 представлены три различные решения: e, σ и σ^*. Исходная матрица представлена перестановкой $e = (1,2,3,4,5)$ (рис. 1a) и значением её фитнес функции $f(e)$, равным 138. Решение $\sigma = (2,3,1,4,5)$ (рис. 1b) иллюстрирует другой вариант решения, имеющий более оптимальное значение фитнеса $f(\pi) = 158$. Наилучшее возможное решение представлено перестановкой $\sigma^* = (5,3,4,2,1)$ (рис. 1c). с фитнесом $f(\sigma^*) = 247$.

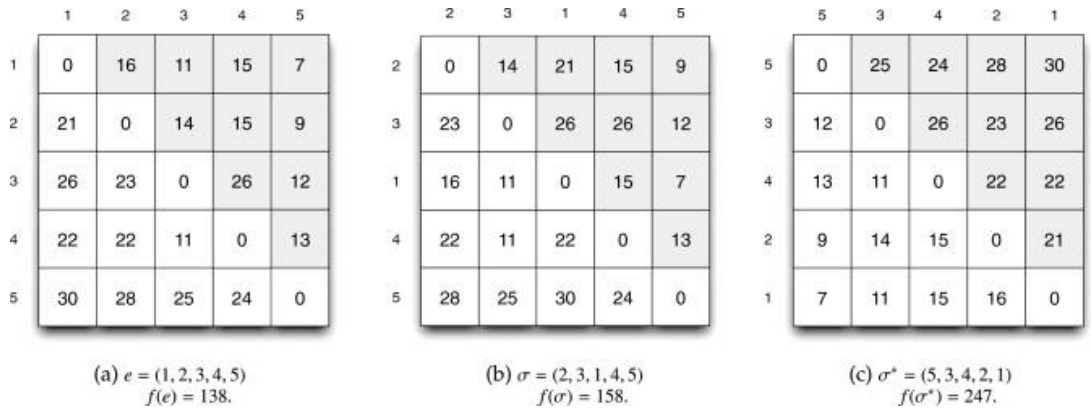


Рис. 1 – Три различных решения для матрицы 5×5: (a) исходная матрица; (b) неоптимальное решение; (c) оптимальное решение.

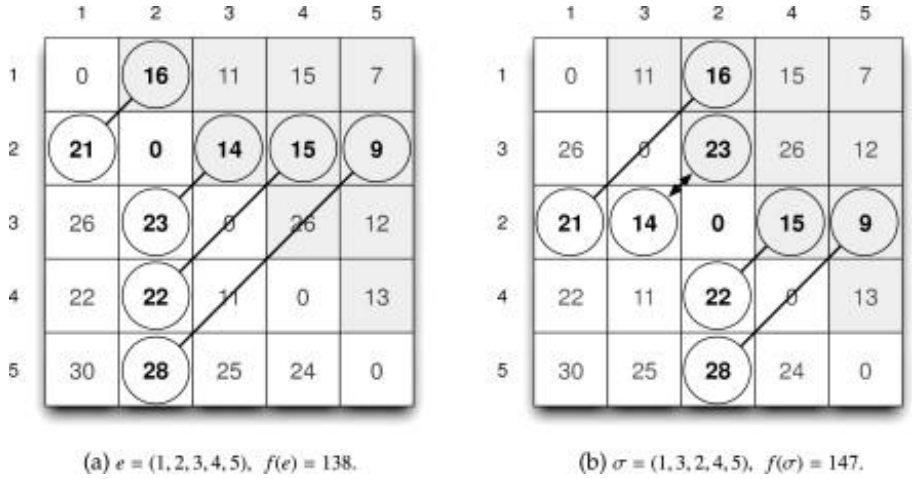
3. АНАЛИЗ ПРОБЛЕМЫ

В ходе анализа проблемы было выявлено, что для любой перестановки индексов σ в матрице B размером $n \times n$ справедливо следующее:

- С каждым индексом $\sigma_i = k, i = 1, \dots, n$ связано $2(n-1)$ записей из матрицы B: $(n-1)$ из строки k и $(n-1)$ из столбца k.
- Набор связанных записей каждого индекса $\sigma_i = k, i = 1, \dots, n$ может быть сгруппирован в пары. Например, каждой записи из строки k, $B_{k\sigma_j}$, соответствует точка из столбца k, $B_{\sigma_j k}$, симметрично расположенная относительно главной диагонали.
- Все пары записей, связанные с индексом σ_i, остаются связанными с ним даже после перестановок.
- Каждая запись $B_{\sigma_i\sigma_j}$ связана с двумя индексами, σ_i и σ_j.
- Для каждой пары $\left\{B_{\sigma_i\sigma_j}, B_{\sigma_j\sigma_i}\right\}$ одна из записей всегда расположена выше главной диагонали, а другая ниже.

Утверждения проиллюстрированы примером на рисунке 2. В этом примере представлены два различных решения: $e = (1,2,3,4,5)$ на рис. 2a и $\sigma = (1,3,2,4,5)$ на рис. 2b. На обоих рисунках записи, связанные с индексом 2, выделены жирным шрифтом. Мы видим, что, несмотря на разное упорядочивание, в обоих решениях набор записей, связанных с индексом 2, одинаков, то есть (21, 14, 15, 9, 16, 23, 22, 28).

Несмотря на то, что позиция индекса 2 в e и σ различна ($e_2 = 2$ и $\sigma_3 = 2$), попарное отношение связанных с ним записей остается неизменным (см. обведенные индексы на рис. 2).



(a) $e = (1, 2, 3, 4, 5)$, $f(e) = 138$. (b) $\sigma = (1, 3, 2, 4, 5)$, $f(\sigma) = 147$.

Рис. 2 – Два различных решения для экземпляра $n = 5$. Обведенные записи, соединенные ребрами, обозначают пары записей, связанные с индексом 2: (a) исходное состояние; (b) после перестановки.

Проверяя расположение связанных записей индекса 2 в σ, мы замечаем, что пара (14, 23) поменяла свои позиции в σ, 14 теперь ниже главной диагонали, а 23 выше неё. Введём новое понятие: вклад индекса в фитнес-функцию.

Когда индекс $k = 1, \dots, n$ занимает позицию i в σ, то есть $\sigma_i = k$, вклад индекса k в функцию определяется суммой записей столбца k в строках $\sigma_1, \dots, \sigma_{i-1}$ и суммой записей строки k в столбцах $\sigma_{i+1}, \dots, \sigma_n$. Иными словами, предыдущие $i - 1$ индексы $\sigma_1, \dots, \sigma_{i-1}$ и последующие $n - i$ индексы $\sigma_{i+1}, \dots, \sigma_n$ определяют вклад индекса k в фитнес-функцию. Формальное описание представлено формулой 2.

$$c(\sigma, i) = \sum_{j=1}^{i-1} B_{\sigma_j \sigma_i} + \sum_{j=i+1}^{n} B_{\sigma_i \sigma_j} \qquad (2)$$

Вернемся к примеру на рисунке 2, благодаря обмену позициями в пару (14, 23) вклад индекса 2 изменился с 54 (16 + 14 + 15 + 9) в e (рис. 2a) до 63 (16 + 23 + 15 + 9) в σ (рис. 2b). В случае индекса 3 его вклад также увеличился, поскольку пара (14, 23) связана с обоими индексами, 2 и 3. И наоборот, в случае индексов 1, 4 и 5 их вклад не меняется от e к σ.

Если мы внимательно посмотрим на уравнение (2), то поймем, что вклад индекса $\sigma_i = k$ на самом деле не определяется конкретным упорядочиванием индексов в предыдущей и последующей позициях i, но их группировкой в этих двух наборах позиций. Как показано в примере 2, вклад индексов 1, 4 и 5 не меняется от e к σ, поскольку группировка остальных индексов в предыдущем и последующем наборах позиций, связанных с индексами 1, 4 и 5, была одинаковой.

Следовательно, при заданном решении σ вклад индекса $\sigma_i, i \in 1, \dots, n$, в фитнес-функцию $f(\sigma, i)$ не зависит от порядка предыдущих индексов $\sigma_1, \dots, \sigma_{i-1}$ и от порядка последующих индексов $\sigma_{i+1}, \dots, \sigma_n$.

Пример из рисунка 3 иллюстрирует, как вклад индекса 3 не зависит от упорядочивания предыдущего и последующего наборов индексов. На рис. 3а вклад индекса 3 равен 63, как результат суммы (11 + 14 + 26 + 12). Если мы проверим вклад индекса 3 в σ (см. рис. 3б), то увидим, что он также равен 63, несмотря на то что индексы {1, 2} и {4, 5} поменялись местами.

	1	2	3	4	5
1	0	16	**11**	15	7
2	21	0	**14**	15	9
3	**26**	**23**	**0**	**26**	**12**
4	22	22	**11**	0	13
5	30	28	**25**	24	0

(a) $e = (1, 2, 3, 4, 5)$, $c(e, 3) = 63$.

	2	1	3	5	4
2	0	21	**14**	9	15
1	16	0	**11**	7	15
3	**23**	**26**	**0**	**12**	**26**
5	28	30	**25**	0	24
4	22	22	**11**	13	0

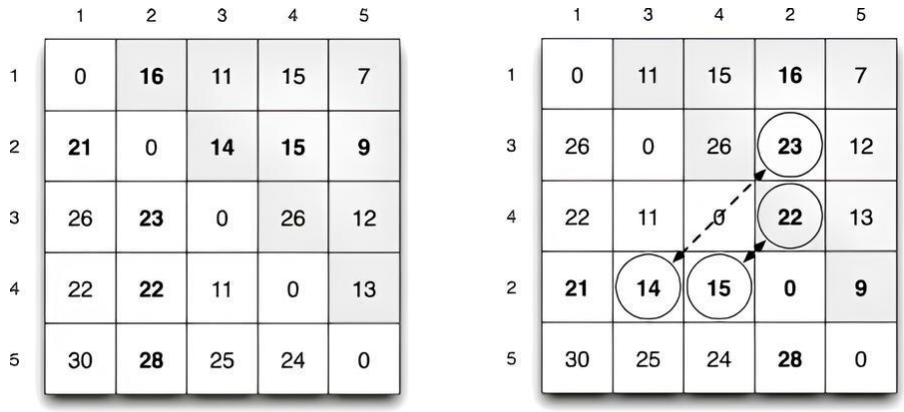
(b) $\sigma = (2, 1, 3, 5, 4)$, $c(\sigma, 3) = 63$.

Рис. 3 – Эффект от замены индексов в позициях 1,2 и 4,5 для индекса 3: (a) исходное состояние; (b) после перестановки.

Как было сказано выше, вклад индекса k не зависит от порядка следования индексов в предыдущих и последующих множествах. Но что произойдет, если индекс $\sigma_j = l$ будет перемещен из предыдущего набора индексов σ_i в последующий набор индексов? В отличие от предыдущего случая, вклад индекса σ_i будет изменён. В этот момент стоит вспомнить, что каждая пара записей $\left\{B_{\sigma_i\sigma_j}, B_{\sigma_j\sigma_i}\right\}$ в матрице

связана с двумя индексами, σ_i и σ_j, а значит, любой обмен местоположением σ_i по определению влияет на вклад в функцию приспособленности σ_i и σ_j. Фактически, перемещение σ_i в позицию j влияет на вклад всех индексов, расположенных между позициями i и j. Приведенный ниже пример (рисунок 4) иллюстрирует изменения в фитнес-функции, вызванные перемещением индекса.

На рисунке 4a и b показана матрица C в соответствии с решениями $e = (1,2,3,4,5)$и $\sigma = (1,3,4,2,5)$. В этом примере мы анализируем последствия перемещения индекса $e_2 = 2$ в позицию 4. В результате этой модификации индексы 3 и 4 сдвигаются на одну позицию влево, что изменяет их вклад в фитнес-функцию. В частности, мы видим, что пары {14, 23} и {15, 22}, связанные с индексами 2-3 и 2-4, поменялись местами. Поэтому вклад индекса 3 меняется с 63 (11 + 14 + 26 + 12) на 72 (11 + 26 + 23 + 12). Аналогично, вклад индекса 4 меняется с 69 (15 + 15 + 26 + 13) до 76 (15 + 26 + 22 + 13). Что касается индекса 2, то его вклад также меняется с 54 (16 + 14 + 15 + 9) до 70 (16 + 23 + 22 + 9). Обратите внимание, что вариация фитнес-вклада индекса 2 равна сумме вариаций индексов 3 и 4.



(a) $e = (1, \mathbf{2}, 3, 4, 5)$
$c(e, 2) = 54$, $c(e, 3) = 63$, $c(e, 4) = 69$.

(b) $\sigma = (1, 3, 4, \mathbf{2}, 5)$
$c(\sigma, 2) = 70$, $c(\sigma, 3) = 72$, $c(\sigma, 4) = 76$.

Рис. 4 – Иллюстрация влияния перемещения индекса 2 (из позиции 2 в позицию 4) на вклад индексов 2, 3 и 4 в фитнес-функцию. Числа, выделенные жирным шрифтом, обозначают записи, связанные с индексом 2. Обведенные кружком пары записей выделяют замененные записи: (a) исходное состояние; (b) после перестановки.

4. ОКРЕСТНОСТИ ВСТАВОК И ЛОКАЛЬНАЯ ОПТИМАЛЬНОСТЬ

Среди широкого разнообразия алгоритмов для решения задачи LOP широко распространены алгоритмы, в основе которых частично или полностью лежат процедуры локального поиска. Хорошо известно, что алгоритмы локального поиска начинают со стартового или базового решения и итеративно пытаются заменить текущее решение лучший, используя для этого систему окрестности. Среди различных систем окрестностей, предложенных для задачи LOP лучший результат показала система окрестностей вставки (Insert Neighborhood System, INS). По этой причине INS будет рассмотрен детально.

Далее даются базовые определения в контексте INS и локальной оптимальности.

Окрестность вставок

Два решения σ и σ' являются соседями в INS (N_I), если σ' получена перестановкой двух элементов i и j сдвигом остальных элементов, что представлено формулой 3.

$$\sigma' \in N_I(\sigma) \Leftrightarrow \exists i, j \in \{1, \dots, n\}, i \neq j, \text{такие, что:}$$

$$\sigma' = \begin{cases} (\sigma'_z = \sigma_z, z < i \wedge z > j) \\ (\sigma'_z = \sigma_{z+1}, i \leq z < j) \wedge & \text{когда } i < j \\ \sigma'_j = \sigma_i \\ \text{или} \\ (\sigma'_z = \sigma_z, z < i \wedge z > j) \ \wedge \\ (\sigma'_z = \sigma_{z-1}, i < z \leq j) \wedge & \text{когда } i > j \\ \sigma'_i = \sigma_j \end{cases} \quad (3)$$

При выполнении операции вставки (перемещении индекса i в позицию j) некоторые элементы в верхнем треугольнике меняются местами с соответствующими элементами в нижнем треугольнике. Рассмотрим два различных сценария операции вставки и выделим конкретные элементы, которые обмениваются:

Случай $i > j$:

- Элементы $\{b_{\sigma_j\sigma_i}, b_{\sigma_{j+1}\sigma_i}, \dots, b_{\sigma_{i-1}\sigma_i}\}$ из верхнего треугольника (ячейки со светлой штриховкой) перемещаются в позиции $\{(j+1, j), (j+2, j), \dots, (i, j)\}$ в нижнем треугольнике

- Элементы $\{b_{\sigma_j\sigma_i}, b_{\sigma_i\sigma_{j+1}}, \dots, b_{\sigma_i\sigma_{i-1}}\}$ из нижнего треугольника (ячейки с темной штриховкой) перемещаются в позиции $\{(j, j+1), (j, j+2), \dots, (j, i)\}$ в верхнем треугольнике

Целевое значение соседней перестановки $\sigma', f(\sigma')$, вычисляется как сумма целевого значения $f(\sigma)$ и разницы обмениваемых элементов:

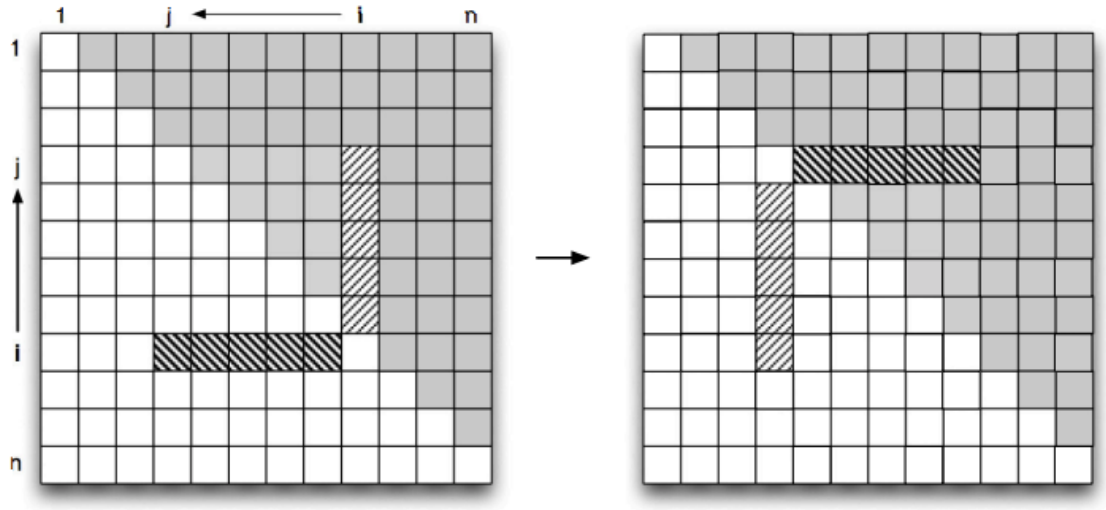
$$f(\sigma') = f(\sigma) + \sum_{z=j}^{i-1} \left(b_{\sigma_z\sigma_i} - b_{\sigma_i\sigma_z}\right) \quad (4)$$

Случай $i < j$:

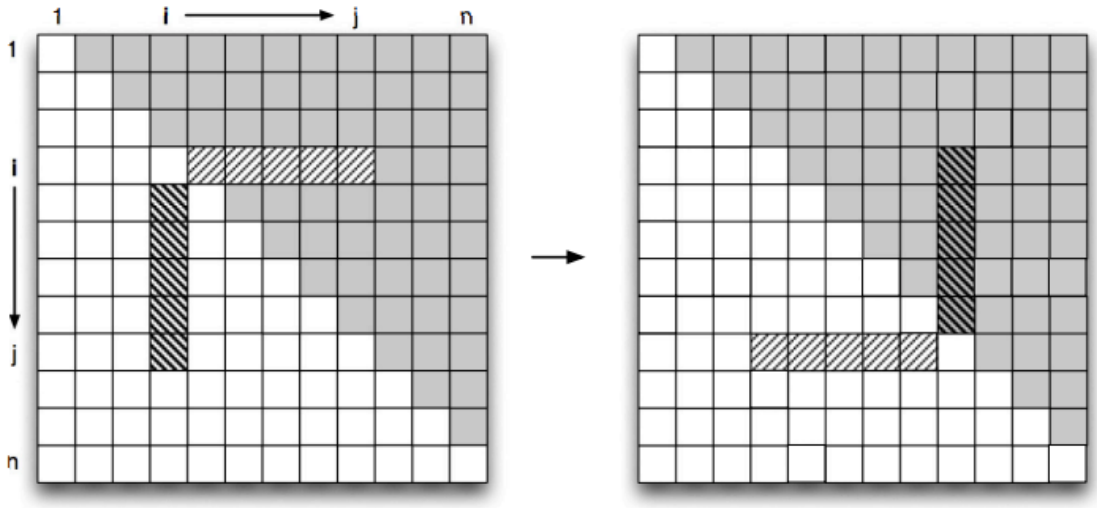
- Элементы $\{b_{\sigma_i\sigma_{i+1}}, b_{\sigma_i\sigma_{i+2}}, \dots, b_{\sigma_i\sigma_j}\}$ из верхнего треугольника (ячейки со светлой штриховкой) перемещаются в позиции $\{(j, i), (j, i+1), \dots, (j, j-1)\}$ в нижнем треугольнике
- Элементы $\{b_{\sigma_{i+1}\sigma_i}, b_{\sigma_{i+2}\sigma_i}, \dots, b_{\sigma_j\sigma_i}\}$ из нижнего треугольника (ячейки с темной штриховкой) перемещаются в позиции $\{(i, j), (i+1, j), \dots, (j-1, j)\}$ в верхнем треугольнике

Аналогично предыдущему случаю, целевое значение вычисляется как:

$$f(\sigma') = f(\sigma) + \sum_{z=i+1}^{j} \left(b_{\sigma_z\sigma_i} - b_{\sigma_i\sigma_z}\right) \quad (5)$$



(a)



(b)

Рис. 5 – Обмен элементами при перемещении индекса. При перемещении индекса i в позицию j происходит обмен элементов матрицы. Рассматриваются два различных сценария: $i > j$ и $i < j$: (a) $i > j$. До и после операции вставки; (b) $i < j$. До и после операции вставки.

Как показано на рисунке 5, пары элементов в ячейках с различной штриховкой (темной и светлой) являются единственными элементами, которые обмениваются в результате операции вставки. Остальные элементы остаются на той же стороне главной диагонали, где и находились до операции. Каждый индекс i имеет связанные пары элементов $\{b_{i1}, b_{1i}\}, \dots, \{b_{in}, b_{ni}\}$. Вводится вектор разностей $(b_{i1} - b_{1i}, \dots, b_{in} - b_{ni})$, где каждое значение вектора описывает изменение целевой функции при обмене соответствующей пары элементов вследствие перемещения индекса i. Далее мы покажем, как этот вектор разностей для каждого индекса играет ключевую роль в определении того, приводит ли позиция индекса к локально оптимальным решениям.

Локальная оптимальность

Решение σ^* является локальным оптимумом для окрестности вставок, если все соседние решения σ имеют меньшее значение целевой функции:

$$\forall \sigma \in N_I(\sigma^*), f(\sigma^*) \geq f(\sigma) \quad (6)$$

Таким образом, в системе окрестности вставок решение считается локально оптимальным тогда и только тогда, когда среди всех возможных операций вставки не существует перемещения, которое улучшает текущее решение.

Согласно формуле 6, для данного решения σ и соседнего решения σ', полученного перемещением индекса k в новую позицию, значение $f(\sigma')$может быть вычислено путём:

- Перерасчёта вклада индекса k в новой позиции
- Добавления полученной вариации к $f(\sigma)$

Следовательно, решение является локальным оптимумом в окрестности вставок, если для любого индекса $i = 1, \dots, n$ не существует операции вставки, которая увеличивает вклад i в целевую функцию.

Для локально оптимального решения σ^* в окрестности вставок выполняется:

- Для каждого индекса $\sigma_i^*, i = 1, \dots, n$, все частичные суммы разностей между соответствующими элементами, расположенными перед i, неотрицательны:

$$\sum_{j=z}^{i-1} \left(b_{\sigma_j^*, \sigma_i^*} - b_{\sigma_i^*, \sigma_j^*} \right) \geq 0, z = i-1, \dots, 1 \tag{7}$$

- Все частичные суммы разностей для элементов, расположенных после i, неположительны:

$$\sum_{j=i+1}^{z} \left(b_{\sigma_j^*, \sigma_i^*} - b_{\sigma_i^*, \sigma_j^*} \right) \leq 0, z = i+1, \dots, n \tag{8}$$

5. ОГРАНИЧЕННАЯ ОКРЕСТНОСТЬ ВСТАВОК

В предыдущем разделе мы описали свойства (формулы 7 и 8), которым должны удовлетворять индексы решения σ, чтобы σ являлось локально оптимальным решением.

Следующий логичный шаг заключается в идентификации конкретных позиций, где индексы порождают локально оптимальные решения. Однако детальный анализ показывает, что такой подход является NP-трудной задачей, поскольку для каждого индекса в каждой позиции, требуется проверить $(n-1)!$ перестановок.

Тем не менее, существуют позиции, в которых индексы не порождают локальный оптимум независимо от порядка остальных индексов. В отличие от предыдущего случая, обнаружение таких позиций является тривиальной задачей.

На основе вектора разностей, ассоциированного с индексами, в данном разделе мы анализируем основания для исключения позиций, в которых индексы не могут порождать локально оптимальные решения. В результате анализа предлагается ограниченная окрестность вставок, исключающая операции перемещения индексов в позиции, не генерирующие локальные оптимумы. Для иллюстрации процесса идентификации таких позиций рассмотрим тривиальные граничные случаи, когда индекс k находится на первой или последней позиции:

- Для исключения первой позиции ($\sigma_1 = k$) необходимо показать, что ни одна перестановка вектора разностей не удовлетворяет уравнению (8). Достаточным условием является невыполнение (8) при $z = n$, так как сумма:

$$\sum_{j=1}^{n} \left(b_{\sigma_j k} - b_{k \sigma_j} \right) \tag{9}$$

  не зависит от порядка индексов $\{\sigma_2, \dots, \sigma_n\}$.
- Аналогично, для исключения последней позиции ($\sigma_n = k$) достаточно проверить невыполнение уравнения (7) при $z = 1$.

Для произвольной позиции i ($\sigma_i = k$) необходимо проверить отсутствие разбиения индексов на группы $\{1, \dots, i-1\}$ и $\{i+1, \dots, n\}$, при котором одновременно выполняются условия (7) и (8).

В связи с этим предлагается простой алгоритм, который начинается с сортировки вектора разностей, ассоциированного с индексом k, в порядке убывания. Поскольку наша цель - исключить неподходящие позиции, второй шаг заключается в проверке, удовлетворяет ли наиболее благоприятное разбиение вектора разностей уравнениям (7) и (8) при $\sigma_i = k$.

Процедура распределения значений включает:

- Размещение наибольшего значения вектора на позиции $i - 1$

- Второго по величине значения - на позиции $i-2$
- И так далее для остальных значений

Для наименьших значений используется обратный порядок:

- Наименьшее значение размещается на позиции $i+1$
- Следующее по величине - на позиции $i+2$

Обозначим через σ' решение, полученное при таком упорядочивании вектора разностей. Тогда индекс k не порождает локально оптимальное решение в позиции i, если выполняется условие:

$$\sum_{z=1}^{i-1}\left(b_{\sigma'_z k}-b_{k\sigma'_z}\right)<0 \text{ или } \sum_{z=i+1}^{n}\left(b_{\sigma'_z k}-b_{k\sigma'_z}\right)>0 \qquad (10)$$

Продолжая данный подход для всех индексов в решении σ и всех возможных позиций, мы вычисляем бинарную матрицу ограничений R, где нулевые элементы соответствуют позициям, в которых индексы не порождают локально оптимальные решения.

Хотя на первый взгляд может показаться, что использование ограниченной окрестности не обязательно даст преимущество перед классическим жадным поиском (поскольку "плохие" решения иногда ведут к более перспективным областям пространства поиска), в случае задачи LOP обнаружена важную закономерность: операция вставки, выбираемая жадным алгоритмом (дающая максимальное улучшение), никогда не относится к запрещенным операциям согласно матрице R.

6. МЕТОДЫ РЕШЕНИЯ

LOP – это NP-полная задача, то есть мы не можем ожидать алгоритма, который решит ее за полиномиальное время. Однако LOP имеет множеств практических приложений [2, 3, 4], и поэтому алгоритмы для его эффективного решения востребованы. Было предложено несколько точных и эвристических алгоритмов. Точные алгоритмы включают в себя метод границ и ветвей(B&B), использующий LP-релаксацию для нижней границы, предложенный Каасом (A branch and bound algorithm for the acyclic subgraph problem), алгоритм ветвей и отсечений(B&C), предложенный

Грёцшелем, Юнгером и Рейнелтом (A cutting plane algorithm for the linear ordering problem) и комбинированный алгоритм Митчелла и Борчерса (Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm). Современные точные алгоритмы могут решать достаточно большие задачи из определенных классов задач с числом столбцов и строк до нескольких сотен, в то время как на экземплярах из других классов гораздо меньшего размера они терпят неудачу. Независимо от типа решаемых задача, время вычислений точных алгоритмов сильно увеличивается с ростом размера матрицы.

LOP также решается с помощью ряда эвристических алгоритмов. К ним относятся жадный алгоритм Беккера, алгоритмы локального поиска, а также множество метаэвристических подходов, включая поиск с запретами (tabu search), scatter search и алгоритмы итеративного локального поиска (ILS). В частности, подходы на основе ILS в настоящее время считаются наиболее успешными метаэвристиками.

Существующие алгоритмы обычно тестировались на множестве классов реальных и случайно сгенерированных примеров. Однако до сих пор мало известно о том, как производительность современных алгоритмов зависит от специфических характеристик различных доступных классов задач LOP, а также о том, как различия между примерами влияют на особенности их пространства поиска. Первые шаги в решении этих открытых вопросов были сделаны в работах Скьявинотто и Шутцле от 2003-го и 2004-го года.

6.1. Метод границ и ветвей с LP-релаксацией

В данном методе задача формулируется в терминах целочисленного линейного программирования. Конкретная формулировка представлена формулой 3.

$$\max \sum_{i \neq j} W_{ij} \cdot x_{ij}, \tag{3}$$

где $x_{ij} = 1$, если элемент i предшествует j, и 0 иначе. При этом накладывается ограничение на сумму парных элементов: $x_{ij} + x_{ji} = 1$ для всех $(i \neq j)$, и на транзитивность: $x_{ij} + x_{jk} - x_{ik} \leq 1$для всех троек$(i, j, k)$.

LP-релаксация переводит целочисленные переменные ($x_{ij} \in \{0,1\}$) в непрерывные ($0 \leq x_{ij} \leq 1$). Решение релаксированной задачи даёт верхнюю границу целевой функции. Если решение релаксации целочисленное и транзитивное, то оно оптимальное.

В иных случаях выбирается переменная (x_{ij}), значение которой в LP-решении ближе всего к 0.5 (наиболее неопределённая). Далее создаются две подзадачи или узла:

- Ветвь 1: ($x_{ij} = 1$);
- Ветвь 2: ($x_{ij} = 0$).

Подзадачи формируют список L. Для каждого узла находится решение и происходит проверка на соответствие ограничениям. Ветви нарушающие ограничения транзитивности отсекаются. Для корректных узлов вычисляется верхняя граница, если она меньше текущего лучшего целочисленного решения, то её тоже необходимо отсечь.

Алгоритм:

1) Инициализация: корневой узел с LP-релаксацией.
2) Пока есть неисследованные узлы:
    a) Выбрать узел с наивысшей верхней границей.
    b) Решить LP-релаксацию.
        i) Если решение целочисленное и транзитивное → обновить лучшее решение.
        ii) Иначе:
            (1) Выбрать переменную для ветвления.
            (2) Создать подзадачи, добавив новые ограничения.
            (3) Добавить подзадачи в очередь узлов.

В ходе работы алгоритма могут быть исследованы все узлы. Поэтому он обладает экспоненциальной сложностью.

6.2. Метод ветвей и отсечений

Метод ветвей и отсечений является улучшением метода ветвей и границ. B&C обнаруживает циклы ($A \rightarrow B \rightarrow C \rightarrow A$) и добавляет их в набор ограничений ($x_{AB} + x_{BC} + x_{CA} \leq 2$), что сужает пространство решений. Данная модификация особенно актуальна для задач с большим числом элементов.

Алгоритм:

1) Инициализация: корневой узел с LP-релаксацией.
2) Пока есть неисследованные узлы:
    a) Выбрать узел с наивысшей верхней границей.
    b) Решить LP-релаксацию.
        i) Если решение целочисленное и транзитивное → обновить лучшее решение.
        ii) Иначе:
            (1) Сгенерировать отсечения для устранения недопустимых решений.
            (2) Если отсечения улучшили верхнюю границу → пересчитать LP.
            (3) Если решение всё ещё нецелочисленное → выполнить ветвление.
            (4) Добавить подзадачи в очередь узлов.

Данный метод сложнее оригинального в реализации, но в то же время и быстрее. Поэтому для задач с большим числом записей следует использовать модификацию. Более подробное сравнение представлено в таблице 1.

6.3. Комбинированный алгоритм Митчелла и Борчерса

Этот метод объединяет метод внутренней точки (Interior Point Method, IPM) и симплекс-метод в рамках алгоритма отсечений (Cutting Plane Algorithm) для решения задач линейного упорядочения (LOP). Его ключевая идея — использовать сильные стороны обоих методов:

- IPM — быстро находит приближённые решения на начальных этапах;
- Симплекс-метод — эффективно уточняет решение и генерирует отсечения.

В данном алгоритме существует три фазы:

1) Фаза внутренней точки: нахождение решения, приближённого к оптимальному. Решается LP-релаксация LOP с помощью IPM за счёт высокой эффективности для задач большой размерности;
2) Фаза генерации отсечений: устранение нарушений транзитивности, цикличности и нецелочисленных решений.
3) Фаза симплекс-метода: уточнение решения после добавления отсечений. Высокая эффективность для локального поиска точных решений в суженном пространстве.

Фазы 2 и 3 повторяются, пока не будет найдено решение.

Преимущества подхода:

1) Скорость и точность:
   a) IPM быстро приближается к оптимуму.
   b) Симплекс-метод точно обрабатывает добавленные отсечения.
2) Сокращение итераций:
   a) Динамические отсечения сужают пространство решений, уменьшая число ветвей.
3) Устойчивость к размерности:
   a) IPM хорошо работает для задач с тысячами переменных.

Комбинированный алгоритм Митчелла и Борчерса для решения LOP объединяет скорость метода внутренних точек (IPM) и точность симплекс-метода, дополненных динамическими отсечениями. IPM быстро приближается к оптимуму, решая LP-релаксацию, после чего симплекс-метод уточняет решение, добавляя отсечения для устранения циклов и нарушений транзитивности. Это позволяет эффективно решать крупномасштабные задачи линейного упорядочения, сокращая число итераций и вычислительные ресурсы за счёт гибридного подхода, который балансирует между скоростью IPM и точностью симплекса.

Сравнение точных методов представлено в таблице 1.

Таблица 1 –

Метод	Теоретическая сложность	Практическая эффективность	Оптима для
B&B с LP	$O(2^n)$	Низкая	Малые задачи (n < 50)
B&C	$O(2^n)$	Средняя	Средние задачи (n < 200)
Митчелл-Борчерс	$O(n^{3.5})$ + экспоненциальная (Симплекс)	Высокая	Крупные задачи (n > 200)

6.4. Алгоритм Беккера

На первом шаге выбирается индекс, максимизирующий стоимость, представленную формулой 4.

$$q_i = \frac{\sum_{k=1}^{n} c_{ik}}{\sum_{k=1}^{n} c_{ki}} i = 1 \dots n \quad (4)$$

Он помещается на первую позицию в перестановке. Затем этот индекс вместе с соответствующими столбцом и строкой удаляется, и из полученной подматрицы. Вычисляются новые значения q_i для оставшихся индексов. Эти шаги повторяются до тех пор, пока список индексов не станет пустым, что приводит к вычислительным затратам $O(n^3)$. А простая вариация этого алгоритма - вычислить значения q_i только один раз в начале работы алгоритма, отсортировать эти значения в неувеличивающемся порядке, чтобы получить перестановку индексов. При использовании этого варианта решение может быть вычислено за $O(n^2)$.

6.5. Алгоритм локального поиска

Алгоритм локального поиска является одним из базовых методов решения задачи линейного упорядочивания (LOP). Этот алгоритм начинает работу с произвольного решения и последовательно улучшает его, исследуя окрестность текущей перестановки с помощью операций вставки. На каждом шаге вычисляется изменение целевой функции, если находится лучшее решение, оно принимается в качестве нового текущего состояния. Процесс продолжается до тех пор, пока в окрестности не

останется улучшающих перемещений, что соответствует достижению локального оптимума.

Шаги алгоритма:

1. **Инициализация начального решения**

   Начальная перестановка, полученная случайно или с помощью алгоритма

2. **Вычисление целевой функции**

   Для текущей перестановки σ вычисляется сумма весов:

$$S = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} W[\sigma(i), \sigma(j)] \quad (5)$$

3. **Определение окрестности**

   Соседние решения генерируются с помощью операции вставки элемента на новую позицию. Для элемента на позиции i рассматриваются все возможные позиции $j \neq i$.

4. **Вычисление изменения целевой функции (дельта)**

   При перемещении элемента $x = \sigma(i)$ в позицию j:

   Если $j < i$:

$$\Delta = \sum_{k=i+1}^{j} (W[x, \sigma(k)] - W[\sigma(k), x]) \quad (6)$$

   Если $j > i$:

$$\Delta = \sum_{k=i+1}^{j} (W[\sigma(k), x] - W[x, \sigma(k)]) \quad (7)$$

5. **Поиск улучшения**

   Для каждого элемента x находится позиция j, дающая максимальное Δ. После этого выбирается перемещение с наибольшим дельта $\Delta > 0$.

6. **Обновление решения**

   Переместите элемент на новую позицию, обновите перестановку σ и сумму S.

7. **Критерий остановки**

   Если улучшений нет (все $\Delta \leq 0$), алгоритм завершается. Текущее решение — локальный оптимум.

Алгоритм локального поиска для LOP демонстрирует высокую эффективность на небольших и средних размерностях задачи, однако его главным ограничением является зависимость от начального решения и риск застревания в локальных оптимумах. Для улучшения результатов часто применяются модификации, такие как мультистартовый поиск или гибридизация с метаэвристиками (например, табу-поиском). Несмотря на простоту, этот метод остается популярным инструментом для приближенного решения LOP благодаря своей прозрачности и гибкости настройки.

6.6. Алгоритм поиска с запретами

Табу-поиск (Tabu Search, TS) — это метаэвристический алгоритм, расширяющий идеи локального поиска за счёт механизмов избегания циклов и выхода из локальных оптимумов. В отличие от стандартного локального поиска, TS использует список табу, запрещающий возврат к недавно посещённым решениям, и стратегии диверсификации, что позволяет исследовать новые области пространства решений.

Шаги алгоритма:

1. **Инициализация**

   Задаётся начальное решение случайным образом или через алгоритм. Инициализируется пустой список табу, который хранит запрещённые перестановки. Определяется критерий останова через максимально допустимое количество итераций.

2. **Генерация окрестности**

   Для текущей перестановки σ рассматриваются все возможные вставки элемента x на новую позицию $j \neq x$. Вычисляется изменение целевой функции ΔS по формулам (6) и (7).

3. **Выбор лучшего допустимого хода**

   Среди всех соседей выбирается тот, что даёт максимальное положительное улучшение ΔS. Если данный ход в списке табу, то он игнорируется и

выбирается следующий по убыванию приращения ΔS сосед. Выполняется ход. Если обнаружено застревание в локальном минимуме, то применяются стратегии диверсификации (например, пертурбация или рестарт из случайной позиции).

4. **Обновление списка табу**

   Запрещается обратное перемещение элемента на несколько итераций (например, 7-10). Список табу обновляется по принципу FIFO.

5. **Критерий остановки**

   Алгоритм завершается при достижении лимита итераций или отсутствии решений.

Самой дорогостоящей по времени является операция генерации окрестности $O(n^2)$. За счёт ограничения на количество итераций общая сложность алгоритма в худшем случае будет ограничена (формула 8).

$$O(k \cdot n^2), \tag{8}$$

где k — максимальное число итераций.

Табу-поиск демонстрирует высокую эффективность в решении LOP, особенно для больших матриц, благодаря способности избегать застревания в локальных оптимумах. Однако его производительность сильно зависит от настройки параметров (длины списка табу, стратегии диверсификации). В сравнении с локальным поиском, TS требует больше вычислений, но обеспечивает лучшее качество решений, что делает его предпочтительным выбором для сложных экземпляров LOP. Для дальнейшего ускорения можно комбинировать TS с другими метаэвристиками (например, реактивным поиском).

6.7. Алгоритм поиска с рассеиванием

Scatter Search (SS) — это популяционная метаэвристика, которая комбинирует решения из элитного множества для генерации новых перспективных кандидатов. В задаче линейного упорядочивания (LOP) SS эффективен благодаря способности сохранять разнообразие решений и комбинировать их лучшие фрагменты. В отличие от генетических алгоритмов, SS использует детерминированные методы выбора и

комбинирования решений, что делает его менее зависимым от случайности и более стабильным для задач с выраженной структурой, такой как LOP.

Шаги алгоритма:

1. **Инициализация популяции**

   Генерация начального множества решений для включения их в популяцию P. Размер популяции обычно небольшой (5-20 решений).

2. **Улучшение решений**

   Каждое решение из популяции локально оптимизируется (например, алгоритмом локального поиска). Как итог, имеется набор элитных решений E.

3. **Построение Reference Set (RS)**

   Выбор b лучших по целевой функции решений из E для включения в RS. Добавление d разнообразных решений (например, с максимальным расстоянием до других) в RS.

4. **Генерация новых решений (Subset Generation)**

   Комбинирование новых решений из пар в RS с помощью операторов линейной комбинации, плавной трансформации решений и других.

5. **Улучшение новых решений**

   Применение локального поиска к сгенерированным решениям.

6. **Обновление Reference Set**

   Замена худших решений в RS на новые улучшенные кандидаты. Критерием может выступать, как целевая функция или разнообразие (функция Хэмминга), так и их взвешенная сумма с динамически меняющимися весами.

7. **Критерий остановки**

   Достигнуто максимальное число итераций без улучшений или достигнут лимит времени.

Общая сложность вычисляется по формуле (9).

$$O(k \cdot |RS^2| \cdot n^3), \quad (\mathbf{9})$$

где k — максимальное число итераций. Для больших n критически важна оптимизация операторов комбинирования.

Scatter Search демонстрирует высокую эффективность для LOP, особенно в случаях, где важно сочетание интенсификации (глубокий поиск вокруг элитных решений) и диверсификации (за счёт включения разнообразных кандидатов). Его сила — в способности систематически комбинировать решения, а не полагаться на случайные мутации. Однако алгоритм требует тщательной настройки

6.8. Итеративный локальный поиск

Итеративный локальный поиск (Iterated Local Search, ILS) — это метаэвристика, объединяющая локальный поиск с механизмами выхода из локальных оптимумов. В задаче линейного упорядочивания (LOP) ILS эффективен благодаря чередованию фаз интенсификации (глубокий поиск вокруг текущего решения) и диверсификации (возмущение для исследования новых областей). Алгоритм особенно полезен для задач средней и большой размерности, где классический локальный поиск застревает в субоптимальных решениях.

Шаги алгоритма:

1. Генерация начального решения

   Создание исходной перестановки σ_0.
2. Локальный поиск

   Применения локального поиска к σ_0. Результат – локальный оптимум σ^*.
3. Perturbation (Возмущение)

   Контролируемая модификация σ^*, чтобы выйти из его окрестности. Для LOP могут быть поменяны местами случайные пары элементов. Сдвиг блоков. Важно, чтобы сила возмущения была минимальной, но достаточной для выхода из локального оптимума.
4. Критерий принятия решения

   Новое решение $\acute{\sigma}$ принимается, если оно лучше текущего $\left(S(\acute{\sigma}) > S(\sigma_{best})\right)$. В вероятностных версиях может приниматься и решение хуже, где эта вероятность вычисляется по специальному закону.

5. Критерий остановки

   Достигнуто максимальное число итераций или отсутствие улучшений в течение t итераций (например, $t = 50$).

Итеративный локальный поиск — мощный метод для решения LOP, особенно для матриц размерности $n \geq 100$. Он легко адаптируется под разные алгоритмы пертурбаций, сочетает глубокий поиск и глобальное исследование, устойчив за счёт выхода из локальных минимумов.

Таким образом, выбор метода зависит от требований к точности, доступных ресурсов и специфики данных. Современные исследования LOP акцентируют развитие гибридных подходов, сочетающих скорость эвристик и точность математических моделей.

6.9. Меметический алгоритм

Меметический алгоритм (Memetic Algorithm, MA) представляет собой гибридный подход, комбинирующий генетический алгоритм с локальной оптимизацией. В контексте задачи линейного упорядочения (LOP) данный метод демонстрирует высокую эффективность благодаря способности сочетать глобальный поиск в пространстве решений с локальным улучшением особей. Особенно результативен MA при работе с задачами большой размерности ($n \geq 1000$), где традиционные точные методы становятся неприменимыми.

Шаги алгоритма:

1. Инициализация популяции
2. Оценка приспособленности:

$$S(\pi_i) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} W[\pi_i(i), \pi_i(j)]$$

3. Селекция: турнирный отбор или метод рулетки
4. Рекомбинация: Ordred Crossover (OX), Partially Mapped Crossover (PMX)
5. Мутация: инверсия или транспозиция с вероятностью $p_m \approx 0.1$
6. Локальный поиск
7. Обновление популяции

Общая сложность: $O(N \cdot n^3)$ на поколение.

Меметический алгоритм демонстрирует превосходные результаты при решении LOP, особенно для задач большой размерности. Ключевые преимущества:

- На $5 - 15\%$ лучшее качество решений по сравнению с классическими метаэвристиками
- Эффективное сочетание глобального и локального поиска
- Гибкость настройки под конкретные условия задачи

6.10. Итеративный локальный поиск на ограниченной окрестности вставки

Restricted Insert Neighbourhood Iterative Local Search (RIN-ILS) — это модификация классического итеративного локального поиска, разработанная специально для задач большой размерности. В контексте LOP алгоритм сочетает мощь ILS с оптимизированной стратегией исследования окрестности, где вместо полного перебора всех возможных вставок рассматривается только их ограниченное подмножество. Этот подход особенно эффективен для матриц с n ≥ 1000, где традиционные методы становятся вычислительно непрактичными.

Шаги алгоритма:

1. Инициализация: генерация начального решения σ_0
2. Локальный поиск с ограниченной окрестностью

   Для текущего решения π:

   a. Ограниченная окрестность вставок: Для каждого элемента рассматриваются только k ближайших позиций ($k \ll n$, типично $k = log_2 n$)

   b. Быстрое вычисление ΔS: Использование префиксных сумм для вычисления изменения целевой функции за $O(1)$ на операцию

   c. Первая улучшающая стратегия: Принятие первого найденного улучшения вместо полного перебора
3. Пертурбация
4. Критерии принятия решения

a. Детерминированное принятие только улучшающих решений
b. Ограниченный откат к предыдущим решениям при длительном застое

5. Остановка по количеству итераций.

Общая сложность: $O(m \cdot n \log n)$, где m — число итераций

Предложенный алгоритм RIN-ILS демонстрирует значительное превосходство над традиционными методами решения LOP для задач большой размерности. Ключевые преимущества подхода заключаются в:

- Эффективности – сокращении вычислительной сложности до $O(n \log n)$ за счёт ограниченной окрестности вставок;
- Масштабируемости – возможности обработки матриц с $n \geq 10^4$;
- Гибкости – адаптивном механизме возмущений, автоматически регулирующем интенсивность диверсификации.

6.11. Генетические алгоритмы

Генетические алгоритмы (ГА) представляют собой мощный метаэвристический подход для решения NP-трудных задач комбинаторной оптимизации, таких как LOP. Основанные на принципах естественного отбора, ГА эффективно исследуют пространство возможных перестановок, комбинируя элементы лучших решений и внедряя контролируемые случайные изменения. В контексте LOP генетические алгоритмы особенно полезны для задач средней и высокой размерности $(50 \leq n \leq 2000)$, где традиционные точные методы становятся неприменимыми из-за вычислительной сложности.

Шаги алгоритма:

1. **Инициализация популяции**: Создание начального множества решений (популяции) размером N. Используются как случайные перестановки, так и решения, построенные жадными алгоритмами (например, сортировка по убыванию суммы строк матрицы весов W).
2. **Оценка приспособленности**: Для каждой особи популяции вычисляется целевая функция, определяющая её качество.

3. **Селекция**: Отбор родительских пар для скрещивания с использованием методов турнирного отбора или рулетки, где вероятность выбора особи пропорциональна её приспособленности.
4. **Кроссовер (рекомбинация)**: Применение операторов скрещивания (например, Order Crossover или Partially Mapped Crossover) для генерации потомков, сохраняющих части структуры родителей.
5. **Мутация**: Внесение случайных изменений (инверсия, транспозиция или сдвиг элементов) с малой вероятностью (обычно 0.01–0.1) для поддержания генетического разнообразия.
6. **Замена популяции**: Формирование нового поколения путём комбинации лучших особей предыдущего поколения (элитизм) и вновь созданных потомков.
7. **Критерий остановки**: Алгоритм завершает работу при достижении максимального числа поколений, исчерпания времени или отсутствия улучшений в течение заданного числа итераций.

Вычислительная сложность генетического алгоритма для LOP определяется несколькими факторами. Инициализация популяции требует $O(N \cdot n^2)$ операций при использовании жадных алгоритмов. Каждая оценка приспособленности имеет сложность $O(n^2)$, что для всей популяции даёт $O(N \cdot n^2)$ на поколение. Операторы кроссовера и мутации выполняются за $O(n)$, но их применение ко всей популяции увеличивает сложность до $O(N \cdot n)$. Таким образом, общая сложность для K поколений составляет $O(K \cdot N \cdot n^2)$. Для больших $n(n \geq 1000)$ критически важны оптимизации: ограничение размера популяции ($N \leq 100$), использование кэширования значений целевой функции и параллельная обработка особей.

Генетические алгоритмы демонстрируют высокую эффективность в решении задач LOP, сочетая глобальный поиск с возможностью адаптации к специфике конкретных экземпляров. Их ключевые преимущества включают способность избегать локальных оптимумов за счёт механизмов мутации и рекомбинации, а также хорошую параллелизуемость. Однако производительность ГА существенно зависит от

корректного выбора параметров (размера популяции, вероятности мутации, операторов кроссовера), что требует тщательной настройки. Для дальнейшего повышения эффективности рекомендуется гибридизация с локальными методами поиска (мемети-ческие алгоритмы) или использование машинного обучения для адаптивного управления параметрами. Применение ГА особенно оправдано в задачах, где требуется нахождение приближённых решений высокой точности за разумное время, таких как анализ крупномасштабных экономических данных или биоинформатические исследования.

6.12. Алгоритм искусственной иммунной системой

Крёмер, Платос и Снасел (2013) разработали новый био-вдохновленный метаэвристический метод для решения задачи линейного упорядочения (LOP), основанный на принципах искусственных иммунных систем (AIS). Данный подход адаптирует алгоритм B-клеток, имитирующий процесс клональной селекции в биологических иммунных системах, предлагая альтернативу традиционным генетическим алгоритмам. Метод демонстрирует особую эффективность в поддержании разнообразия популяции решений и исследовании пространства поиска, что делает его подходящим для задач LOP средней и высокой размерности.

Шаги алгоритма:

1. **Инициализация**: Создание популяции B-клеток (кандидатных решений), представленных в виде случайных перестановок индексов матрицы.
2. **Оценка аффинности**: Вычисление качества каждой B-клетки с использованием целевой функции LOP (сумма весов над главной диагональю).
3. **Клональная селекция**: Отбор B-клеток с наивысшей аффинностью для клонирования, где количество копий пропорционально их качеству.
4. **Гипермутация**: Применение операторов мутации к клонам, причем интенсивность мутаций обратно пропорциональна аффинности родительской клетки.
5. **Метадинамика**: Замена наихудших решений новыми случайными B-клетками для поддержания разнообразия популяции.

6. **Завершение**: Повторение шагов 2-5 до выполнения критериев останова (например, достижение лимита итераций или требуемого качества решения).

Алгоритм AIS имеет временную сложность $O(G \cdot P \cdot n^2)$, где G - число поколений, P - размер популяции, n - размерность задачи. Этапы клонирования и селекции требуют $O(P \cdot logP)$ операций на поколение из-за необходимости сортировки. Гипермутация выполняется за $O(P \cdot C \cdot n)$, где C - среднее число клонов на клетку. Объем используемой памяти составляет $O(P \cdot n)$. Хотя асимптотическая сложность сравнима с генетическими алгоритмами, AIS обычно сходится быстрее благодаря механизмам сохранения разнообразия, что делает его особенно эффективным для структурированных задач LOP с $n \leq 500$.

Предложенный AIS-подход обладает тремя ключевыми преимуществами: (1) адаптивные мутации предотвращают преждевременную сходимость, (2) метадинамика поддерживает разнообразие решений, (3) механизм клональной селекции обеспечивает целенаправленный локальный поиск. Эксперименты показывают превосходство над стандартными генетическими алгоритмами, особенно для матриц со сложной структурой. Перспективные направления включают гибридизацию AIS с методами локального поиска и разработку GPU-реализаций для сверхбольших задач LOP. Данный био-вдохновленный метод представляет собой ценный инструмент для решения задач оптимизации, когда приоритетом является качество решения.

6.13. Меметический алгоритм с многородительской рекомбинацией

Ye, Wang, Lü и Hao (2014) разработали инновационный меметический алгоритм для решения задачи линейного упорядочения (LOP), отличающийся применением многородительского оператора рекомбинации. Этот подход преодолевает ограничения классических двухродительских генетических алгоритмов за счет одновременного использования нескольких родительских решений (обычно 3-5) для генерации потомков. Особенностью метода является интеллектуальный механизм обновления популяции, учитывающий как качество решений, так и их пространственное

распределение, что обеспечивает баланс между интенсификацией и диверсификацией поиска.

Шаги алгоритма:

1. **Инициализация популяции**: Создание стартовой популяции из N решений с использованием жадных алгоритмов и случайных перестановок.
2. **Многородительская рекомбинация**: Для каждой группы из k родителей ($k \geq 3$) построение потомка через:
    - Анализ общих элементов в одинаковых позициях
    - Консенсусное заполнение конфликтных позиций
    - Случайное назначение оставшихся элементов
3. **Локальная оптимизация**: Применение ограниченного поиска с вставками к каждому потомку (5-10 итераций).
4. **Дистанционно-качественный отбор**: Обновление популяции на основе:
    - Ранжирования по целевой функции
    - Минимального попарного расстояния Хэмминга
5. **Адаптивное управление**: Автоматическая регулировка:
    - Числа родителей (3-5) на основе диверсификации
    - Интенсивности локального поиска по качеству потомков

Вычислительная сложность алгоритма складывается из трех основных компонентов: операция рекомбинации требует $O(k \cdot n^2)$ операций для k родителей из-за анализа позиционных конфликтов, локальная оптимизация занимает $O(L \cdot n^3)$ на потомка (при $L \leq 10$ итерациях), а обновление популяции - $O(N^2 \cdot n)$ вследствие вычисления попарных расстояний Хэмминга. Общая сложность на поколение составляет $O\big(M \cdot (k \cdot n^2 + L \cdot n^3)\big) + O(N^2 \cdot n)$, где M - количество потомков. При типичных параметрах ($N = 50, M = 30, k = 4, L = 5, n = 200$) алгоритм демонстрирует практическую эффективность, однако для задач с $n > 500$ требуются дополнительные оптимизации для снижения вычислительных затрат.

Предложенный алгоритм демонстрирует существенные преимущества: многородительская рекомбинация ускоряет исследование пространства решений на 30-40%, гибридный критерий отбора эффективно предотвращает преждевременную сходимость, а адаптивное управление параметрами минимизирует потребность в ручной настройке. Экспериментальные результаты на тестах LOLIB подтверждают повышение качества решений на 5-8% относительно традиционных меметических алгоритмов. Перспективы дальнейшего развития включают интеграцию методов машинного обучения для автоматизации выбора параметров и создание параллельных реализаций для обработки сверхбольших задач ($n \geq 10^4$), что особенно актуально для современных приложений анализа данных.

6.14. Алгоритм TREE

Алгоритм TREE представляет собой специализированный метод локального поиска для задачи линейного упорядочения, оптимизированный для работы с окрестностью вставок. Его ключевая особенность — использование древовидной структуры данных для эффективного анализа возможных перестановок.

Шаги алгоритма:

1. Построение деревьев для каждой вершины графа, где листья соответствуют возможным позициям вставки
2. Вычисление значений γ (стоимость обратных дуг) для всех узлов дерева
3. Поиск вершины с минимальным значением γ в корне дерева
4. Определение оптимальной позиции вставки через обход дерева от корня к листу
5. Обновление текущего решения и соответствующих структур данных
6. Повторение до достижения локального оптимума

Временная сложность составляет $O(n^2)$ для построения деревьев и $O(n)$ для каждой итерации поиска. Общая сложность — $O(k \cdot n^2)$, где k — число итераций до сходимости. Алгоритм особенно эффективен для плотных графов благодаря оптимизированным операциям обновления.

TREE демонстрирует превосходную производительность для задач средней размерности ($n \leq 500$), обеспечивая точный локальный поиск с гарантией нахождения оптимума в заданной окрестности. Его основное преимущество — сбалансированность между качеством решений и вычислительными затратами.

6.15. Алгоритм великого потопа

Алгоритм "Великого потопа" (GDA) представляет собой метаэвристический подход для решения задач оптимизации, вдохновленный природным процессом адаптации к изменяющимся условиям. В контексте задачи линейного упорядочения (LOP) GDA демонстрирует особую эффективность благодаря способности избегать локальных оптимумов за счет динамического управления уровнем принятия решений ("уровнем воды").

Iwl - начальный уровень воды, dryratio - коэффициент испарения.

Шаги алгоритма:

1. Генерация начального решения s с использованием конструктивной эвристики Беккера
2. Инициализация параметра $waterlevel = iwl \times cost(s)$
3. Пока не выполнен критерий остановки:
    1. Случайный выбор вершины v для вставки
    2. Обход дерева TREE для v с целью поиска допустимой позиции:
        1. Начало от корня дерева $\left(x = r, \gamma_m in = \gamma(r)\right)$
        2. Построение множества допустимых дочерних узлов Y
        3. Случайный выбор узла $y \in Y$
        4. Повторение процедуры до достижения листа
    3. Вставка v в найденную позицию → новое решение s'
    4. Если $cost(s') \leq waterlevel$:
        1. Обновление решения: $s = s'$
        2. Расчет величины испарения: $dry = dryratio \times \left(waterlevel - cost(s')\right)$
        3. Уменьшение уровня воды: $waterlevel -= dry$
    5. Если пройдено n итераций без улучшений:
        1. Сброс $waterlevel = iwl \times cost(s)$

Вычислительная сложность алгоритма GDA складывается из трех основных компонентов: конструктивная эвристика требует $O(n^2)$ операций, обход деревьев TREE выполняется за $O\big(deg(v)\big)$ для каждой вершины, а общая сложность одной итерации в худшем случае составляет $O(n^2)$. Критическое влияние на производительность оказывают параметры iwl (начальный уровень воды) и $dryratio$ (коэффициент испарения), которые требуют тщательной эмпирической настройки для различных типов задач LOP, что особенно важно при работе с матрицами большой размерности ($n \geq 500$).

Алгоритм GDA демонстрирует высокую эффективность при решении сложных задач LOP с многочисленными локальными оптимумами, показывая на 5-15% лучшие результаты по сравнению с классическими методами. Его ключевые преимущества включают: (1) гарантированное нахождение допустимых решений за счет интеграции с TREE-структурами, (2) адаптивный механизм управления уровнем принятия решений через динамическое изменение waterlevel, и (3) устойчивость к застреванию благодаря периодическому сбросу параметров. Наибольшую эффективность алгоритм проявляет при работе с задачами размерности 500-5000 элементов, обеспечивая оптимальный баланс между качеством получаемых решений и вычислительными затратами, что делает его предпочтительным выбором для сложных практических приложений.

7. ПРИМЕНЕНИЯ

Задача линейного упорядочивания (Linear Ordering Problem, LOP) направлена на поиск оптимальной последовательности элементов, максимизирующей сумму весов над главной диагональю матрицы. Эта задача имеет широкое применение в различных областях, где требуется ранжирование объектов с учётом их взаимного влияния, приоритетов или структурных зависимостей. Универсальность LOP делает её инструментом для анализа данных, оптимизации процессов и принятия решений в междисциплинарных исследованиях.

В экономике LOP используется для анализа межотраслевых взаимодействий. Например, матрица весов может отражать объёмы поставок между секторами

экономики. Оптимальное упорядочивание позволяет определить последовательность отраслей, максимизирующую совокупный экономический эффект. В финансах LOP применяется для ранжирования активов по их волатильности или корреляции, что помогает строить сбалансированные инвестиционные портфели.

В социологических исследованиях LOP помогает ранжировать социальные факторы по степени влияния на определённый показатель (например, уровень образования на доход). В психологии задача используется для анализа результатов тестов, где важно определить порядок вопросов или реакций, минимизирующий когнитивную нагрузку. Например, упорядочивание стимулов в эксперименте для выявления скрытых поведенческих паттернов.

В биоинформатике LOP применяется для анализа данных генной экспрессии. Матрица весов может отражать силу взаимодействия между генами, а оптимальное упорядочивание помогает выявить цепочки регуляторных процессов. В генетике LOP используется для построения филогенетических деревьев, где важно определить последовательность мутаций, объясняющую эволюционное развитие видов.

В IT LOP востребована в задачах ранжирования поисковой выдачи, рекомендательных систем и обработки естественного языка. Например, матрица весов может кодировать релевантность документов запросу, а оптимальный порядок элементов улучшает точность выдачи. В машинном обучении LOP используется для обучения моделей ранжирования (Learning to Rank), где алгоритмы учатся предсказывать порядок объектов, максимизирующий пользовательскую удовлетворённость.

В логистике LOP помогает оптимизировать маршруты доставки, где матрица весов отражает время или стоимость перемещения между точками. Упорядочивание пунктов позволяет минимизировать общие затраты. В управлении цепями поставок LOP используется для расстановки приоритетов между этапами производства, чтобы сократить простои и повысить эффективность.

В археологии задача линейного упорядочения (LOP) играет ключевую роль в решении проблем стратиграфии — определения хронологической последовательности культурных слоёв и артефактов. Например, матрица Харриса, используемая для

визуализации стратиграфических отношений, кодирует информацию о взаимном расположении слоёв: если слой A перекрывает слой B, это указывает, что A моложе B. Однако при наличии множества пересекающихся слоёв и фрагментарных данных ручной анализ становится крайне трудоёмким. LOP позволяет автоматизировать этот процесс, преобразуя матрицу Харриса в матрицу весов, где элемент $W[i,j]$ отражает частоту случаев, когда слой i должен предшествовать слою j. Решение LOP даёт оптимальную последовательность слоёв, максимизирующую согласованность стратиграфических данных. Это особенно важно при анализе крупных раскопок, таких как многослойные поселения или некрополи, где точное датирование слоёв критично для реконструкции исторических процессов. Алгоритмы на основе LOP (например, метаэвристики) успешно применяются для обработки таких данных, сокращая время анализа и минимизируя субъективные ошибки, что способствует более объективному восстановлению хронологии археологических памятников.

LOP служит мощным инструментом для решения задач, требующих учёта парных взаимодействий и структурных зависимостей. Её применение охватывает экономику, социологию, биоинформатику, IT и логистику, демонстрируя междисциплинарную ценность. С развитием алгоритмов оптимизации и роста объёмов данных роль LOP будет только возрастать, особенно в областях, где критически важен анализ сложных систем и принятие решений на основе множества параметров.

8. ПРАКТИЧЕСКАЯ ЧАСТЬ

9. ЗАКЛЮЧЕНИЕ

Проведённое исследование задачи линейного упорядочения (LOP) подтвердило её ключевую роль в комбинаторной оптимизации и междисциплинарных приложениях. Будучи NP-сложной задачей, LOP требует применения разнообразных методов, адаптированных к специфике конкретных областей. В работе систематизированы теоретические основы LOP, включая её связь с проблемами ранжирования, графовыми моделями и матричными представлениями.

Сравнительный анализ методов решения продемонстрировал, что точные алгоритмы (например, ветвей и границ) эффективны для малых размерностей, тогда как

эвристики (жадные стратегии) и метаэвристики (табу-поиск, ILS) позволяют находить приближённые решения для задач с большими данными. Это особенно актуально в контексте современных вызовов, связанных с обработкой Big Data и интеграцией LOP в системы искусственного интеллекта.

Результаты исследования подчёркивают, что выбор метода решения LOP должен учитывать не только вычислительную сложность, но и специфику данных. Например, в задачах с высокой размерностью (машинный перевод, рекомендательные системы) метаэвристики демонстрируют лучший баланс между точностью и скоростью.

Перспективным направлением дальнейших исследований является разработка гибридных алгоритмов, сочетающих преимущества точных и приближённых методов, а также адаптация LOP для работы с динамическими данными в реальном времени. Это позволит расширить применение задачи в таких актуальных сферах, как анализ социальных сетей, управление цепями поставок и персонализированная медицина.

Таким образом, LOP остаётся важным инструментом для решения сложных оптимизационных задач, а её изучение способствует развитию как теоретической информатики, так и прикладных наук.

10.СПИСОК ЛИТЕРАТУРЫ

1) Статья с сайта: Fabio Duarte Amount of Data Created Daily (2025) [электронный ресурс] // explodingtopics URL: https://explodingtopics.com/blog/data-generated-per-day (дата обращения: 06.05.2025)
2) Aparicio J., Landete M., Monge J. F. A linear ordering problem of sets //Annals of Operations Research. – 2020. – Т. 288. – №. 1. – С. 45-64.
3) Cameron T. R., Charmot S., Pulaj J. On the linear ordering problem and the rankability of data //arXiv preprint arXiv:2104.05816. – 2021.
4) Alcaraz J. et al. The linear ordering problem with clusters: a new partial ranking //Top. – 2020. – Т. 28. – С. 646-671.