УДК 37.01:007+004.82+004.85+519.713
DOI: 10.25559/SITITO.18.202202.287-299

Научная статья

Об объектно-ориентированной реализации метода ветвей и границ для задачи коммивояжёра. Часть I

Б. Ф. Мельников
Совместный университет МГУ – ППИ, г. Шэньчжэнь, Китайская Народная Республика
Адрес: 517182, Китайская Народная Республика, провинция Гуандун, г. Шэньчжэнь, р-н Лунган, Даюньсиньчэн, ул. Гоцзидасюеюань, д. 1
bormel@mail.ru

Аннотация

Многие задачи дискретной оптимизации очень сложны для решения на компьютере – откуда и идёт их название «труднорешаемые». Точнее, сложны для решения (для описания алгоритмов, для программирования) возможные подходы к быстрому решению этих задач – переборное же решение, как правило, программируется просто, но работает соответствующая программа гораздо медленнее. Настоящую статью, как и предыдущую на эту тему, можно назвать «методической». В предыдущей статье на эту тему мы описали наш подход к методу ветвей и границ, прежде всего – для классической задачи коммивояжёра. Мы рассматривали как классические эвристики этого метода, так и их современную интерпретацию. У настоящей статьи основная цель – совсем иная, чем была в предыдущей: сейчас мы рассматриваем именно программные аспекты реализации этого метода. Сам программный комплекс мы выполнили на языке Си++ – и при этом пытались использовать все такие возможности языка, которые можно применять для программирования практически любых алгоритмов дискретной математики; конечно, в первую очередь мы имеем в виду объектно-ориентированные возможности. В конце статьи (в части II) мы приводим некоторые результаты вычислительных экспериментов. А в заключении мы рассматриваем несколько интересных тем для дальнейшего исследования студентами: обе статьи (предыдущая и настоящая) описывают только начало очень большого количества возможных тем, связанных с применением метода ветвей и границ для задач дискретной оптимизации, в частности – с объектно-ориентированной реализацией этого метода.

Ключевые слова: оптимизационная задача, задача коммивояжёра, эвристический алгоритм, метод ветвей и границ, Си++

Автор заявляет об отсутствии конфликта интересов.

Для цитирования: Мельников Б. Ф. Об объектно-ориентированной реализации метода ветвей и границ для задачи коммивояжёра. Часть I // Современные информационные технологии и ИТ-образование. 2022. Т. 18, № 2. С. 287-299. doi: https://doi.org/10.25559/SITITO.18.202202.287-299

Original article

About the Object-Oriented Implementation of the Branch and Boundary Method for the Travelling Salesman Problem. Part I

B. F. Melnikov
Shenzhen MSU – BIT University, Shenzhen, People's Republic of China
Address: 1 Guoji-daxueyuan St., Dayunxincheng, Longgang District, Shenzhen 517182, Guangdong Province, People's Republic of China
bormel@mail.ru

Abstract

Many discrete optimization problems are very difficult to solve on a computer; hence they are titled "intractable". More precisely, the possible approaches to solving these problems quickly are difficult to solve (for describing algorithms, for programming), and the iterative solution, as a rule, is programmed simply, but the corresponding program works much slower. This paper, like the previous one on this topic, can be called "methodical". In the previous paper on this topic, we described our approach to the method of branches and boundaries, primarily for the classical traveling salesman problem. We considered both the classical heuristics of this method and their modern interpretation. The main purpose of this article is quite different from that of the previous article: now we consider the software aspects of the implementation of this method. We executed the software package in C++ – and at the same time tried to use all such language features that can be used for programming practically any algorithms of discrete mathematics; of course, first and foremost we mean object-oriented capabilities. At the end of the article (in Part II), we present some results of computational experiments. And in conclusion, we consider several interesting topics for further study by students: both papers (the previous one and the present one) describe only the beginning of a very large number of possible topics related to the application of the branches and bounds method for discrete optimization problems, in particular, with the object-oriented implementation of this method.

Keywords: optimization problem, traveling salesman problem, heuristic algorithm, branches and bounds method, C++

The author declares no conflict of interest.

For citation: Melnikov B.F. About the Object-Oriented Implementation of the Branch and Boundary Method for the Travelling Salesman Problem. Part I. *Sovremennye informacionnye tehnologii i IT-obrazovanie = Modern Information Technologies and IT-Education*. 2022; 18(2):287-299. doi: https://doi.org/10.25559/SITITO.18.202202.287-299

Введение

Настоящая статья может рассматриваться как продолжение [1] (причём о таком возможном продолжении мы в [1] неоднократно говорили); обе эти статьи можно назвать «методическими». При этом слово «продолжение» уместно, поскольку мы рассматриваем ту же самую предметную область: реализацию алгоритма метода ветвей и границ (МВГ) для задачи коммивояжёра (ЗКВ)[1]. Однако можно также сказать (и это не будет ошибкой), что настоящая статья – на совсем иную тему: мы здесь рассматриваем реализацию алгоритмов дискретной математики с помощью объектно-ориентированного программирования, а возможно – и общий подход к подобной реализации.

Для реализации мы используем язык Си++[2] – язык, который, без сомнения, «займёт чистое первое место» по любому естественному «комплексному критерию», включающему в себя:

- в первую очередь, конечно же, возможность применения объектно-ориентированного программирования[3];
- удобство оформления сложных структур данных и возможность применения абстракции данных[4];
- удобство реализации алгоритмов – а именно тех алгоритмов, которые предназначены для работы с дискретными математическими объектами, для оптимизационных задач и т. п.;
- красивая и понятная запись исходного текста программ[5];
- возможность применения множественного наследования[6];
- эффективность получаемого кода исполняемых программ[7];
- возможность малого изменения текста программы – при изменении (причём иногда значительном) реализуемого алгоритма;
- доступность трансляторов и связанных с ними сред разработки;
- удобство («дружественность») последних.
- Конечно же, по одиночным критериям есть и другие «лидеры рейтинга»:
- Ассемблер и даже обычный («неоплюсевший») Си могут дать более эффективные исполняемые программы;
- столь разные языки, как широко известный Питон и значительно менее известный язык системы GAP, дают некоторую экономию времени разработки алгоритмов и реализации соответствующих программ[8];
- современные «улучшения» Си++ (например, введение в язык концепции диапазонов) позволяют алгоритмам напрямую работать с контейнерами – и тем самым часто упрощают саму запись алгоритмов;
- Рубин позволяет очень удобно записывать работу с метаклассами – а эту работу как в обычном Си++, так и в уже упомянутом Питоне можно назвать не очень удачной;
- многие языки (мы даже не будем приводить конкретные названия и ссылки) дают возможность более удобной разработки параллельных алгоритмов и тем более реализации соответствующих программ;
- и т. д.

Однако, повторим, мы имеем в виду некоторый «комплексный критерий».

Теперь, в качестве «немного запоздалого эпиграфа», приведём цитату Б. Страуступа, уже упомянутого ранее в сноске[9]:

«Я всегда мечтал о том, чтобы работа с компьютером была не сложнее пользования телефоном. Моя мечта стала реальностью. Теперь я уже не знаю, как пользоваться всеми возможностями телефона».

По-видимому, несложно догадаться, какое отношение приведённая цитата имеет к языку Си++ вообще – и к нашему проекту в частности: да, язык Си++ действительно стал таким, что за всеми его возможностями просто «не угнаться». Но! Мы употребляем[10] только очень ограниченное подмножество языка Си++ – впрочем, подмножество, включающее все нужные нам

[1] Более того, как мы отмечали в [1], основным в той статье являлось описание самого МВГ – в то время как предметной областью можно было считать конкретную задачу дискретной оптимизации, т. е. ЗКВ.

[2] По-видимому, несколько чаще используемая в русском языке запись названия языка программирования как С++ хуже: она противоречит либо нормам русского языка, либо сложившейся традиции прочтения названия. Ведь если читать по-английски – то надо читать «плас-плас» вместо «плюс-плюс», а если по-латински (либо «по-математически») – то надо «цэ» вместо «си». Впрочем, что-либо «переделывать», конечно, поздно – но такая несогласованность объясняет употребление нами записи «Си++».

[3] На этом пункте можно было бы и закончить? Он «включает в себя» практически все последующие?

[4] Которую мы понимаем «по Страуструпу»: отделение несущественных деталей реализации подпрограммы от характеристик, существенных для её корректного использования.

[5] Мы считаем, что этот пункт, конечно, «коррелирует» с двумя предыдущими – но всё-таки сильно от них отличается. По этому поводу приведём цитату классиков программирования (Ч. Хоар и А. Шень, [2]): «эстетическая прелесть программы – это не архитектурное излишество, а то, что отличает в программировании успех от неудачи».

[6] Его применение часто критикуется – но при этом чаще всего напрасно. Если соблюдать некоторые ограничения (в частности – следить за невозможностью появления т.н. ромбовидного наследования), то множественное наследование может привести к очень красиво записанным алгоритмам. («Transparent» – тот редкий случай, когда сто́ит употребить английский термин.) При этом подобные ограничения, по-видимому, даже не надо указывать формально: каждый программист может легко сформулировать их сам для себя.

[7] Что бы ни говорили «классики» и современные специалисты (Ben-Ari M. Understanding Programming Languages. 1st. ed. Wiley, 1996. 376 p. и мн. др.), какие бы аргументы они ни приводили – а эффективность получаемых exe-файлов «всё-таки» очень даже зависит от используемых для их получения языков программирования («А всё-таки она вертится».)

[8] GAP – Groups, Algorithms, Programming – a System for Computational Discrete Algebra [Электронный ресурс] // GAP-system, 2022. URL: https://www.gap-system.org (дата обращения: 16.05.2022); мы «объединили» эти языки в общий пункт именно потому, что такая экономия времени разработки объясняется одними и теми же причинами – использованием высокоуровневых библиотек.

[9] Это – «a folklore story». По-видимому, эта мысль была изложена в интервью, данном Страуступом журналу Computer 1 января 1998 г.; по крайней мере, так написано в цитированиях на многочисленных сайтах – но первоисточник найти не удаётся. (Впрочем, чаще всего эта цитата приводится без конкретных ссылок.)

[10] Причём употребляем как в этом проекте, так и во многих других. Можно упомянуть, например, наши предыдущие публикации в этом журнале [4], [5] – несмотря на то, что в первой из них мы не привели конкретных текстов программы, а во второй – привели, но не на Си++, а на «идеологически близком Шарпе».

возможности объектно-ориентированного языка программирования[11], достаточные для того, чтобы описывать необходимые конструкции реализации алгоритмов.

Итак, основная цель настоящей статьи – совсем иная чем у «первой части» [1]: привести реализацию сложных алгоритмов дискретной математики (в первой части мы описывали такие алгоритмы). И по этому поводу важно отметить следующее. Даже в очень хороших книгах по алгоритмизации, включая[12], имеется один общий недостаток: его кратко можно назвать как недостаточное описание соответствующих программ «для реальных условий», прежде всего – для больших размерностей. Например (но не только), мы имеем в виду вопросы выделения и освобождения памяти при работе с очень большими размерностями[13]. «Отдавать это на откуп» выполняемым программам, создаваемым трансляторами с Ява-подобных языков[14]? Понятно, что от этого выполняемый код становится существенно менее эффективным – что можно увидеть, сравнивая время работы сложных программ (написанных на разных языках программирования «одинаково»), в частности – программ для рассматриваемого в настоящей статье МВГ для ЗКВ. Поэтому мы считаем, что о выделении и освобождении памяти нужно «думать самим» – что наиболее удобно, по-видимому, тоже на Си++.

Мы заканчиваем содержательную часть введения напоминанием о том, что для понимания настоящей статьи, конечно, крайне желательно знакомство с [1]. Теперь приведём краткое содержание статьи по разделам.

В разделе 2 приведены самые общие замечания про представляемую программу (т.е. про используемые нами структуры данных и алгоритмы); всё это будет ра́звито[15] в последующем тексте статьи. В разделе 3 описаны использованные в программе вспомогательные классы. Среди методов этих классов особо отметим реализацию возможности слежения за тем, чтобы при в процессе решения задачи не получались бы «малые циклы» – мы уже кратко упоминали об этой проблеме в [1]. А в разделе 4 начато подробное рассмотрение класса для подзадачи – конечно же, самого важного (и самый сложного) класса настоящего проекта.

Разделы с номерами 5 и более составляют часть II настоящей статьи. В разделе 5 мы продолжаем рассматривать класс для подзадачи, и приводим основной метод этого класса; можно сказать, что он является самым важным не только для рассматриваемого класса SubTask, но и для всего описываемого проекта. А «по большому счёту» этот метод – самый важный (ключевой) для всего подхода к реализации МВГ (для любой задачи дискретной оптимизации, т. е. далеко не только для ЗКВ).

Из подзадач «собирается»[16] класс для всей задачи – рассматриваемый в разделе 6. В разделе 7 приводятся некоторые результаты вычислительных экспериментов. А в заключении (раздел 8) приводятся интересные исследовательские проблемы для дальнейшего выполнения студентами: ведь [1] вместе с настоящей статьёй описывают только начало очень большого количества возможных работ, связанных с объектно-ориентированной реализацией метода ветвей и границ.

Общие замечания про программу (её структуры данных и алгоритмы)

Как мы уже отмечали во введении, мы стараемся использовать объектно-ориентированные возможности[17], удобные для описания алгоритмов решения математических задач, в частности – задач дискретной оптимизации. Процитируем, однако, ещё раз А. Шеня – про свою книгу он приводит такую антирекламу[18]:

«…в ней нет ни слова об объектно-ориентированном программировании, открывшем новую эпоху в построении дружественных и эффективных программных систем, и о современных библиотеках объектов и классов…»

---

[11] Иными словами – мы выбираем только немного «добавок» языка Си++; впрочем, если говорить об «обычном Си» – то в нём мы тоже применяем очень ограниченный набор примитивов.

[12] Кроме неё, для обучения студентов автор постоянно использует столь разные книги, как [2] (и, конечно, предыдущие их издания). См. Lipski W. Kombinatoryka dla programistów. Wydawnictwa Naukowo-Techniczne, Warszawa, 1989. 181 p.; Sedgewick R. Algorithms in C, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching. 3rd ed. Addison-Wesley Professional, 1997. 720 p. Все эти книги, конечно, можно назвать очень хорошими, хотя написаны они «в разном стиле» – т.е. с совершенно разных точек зрения на алгоритмизацию. Но! во всех – присутствует основной недостаток, который мы и отмечаем для издания: Cormen T. H., Leiserson Ch. E., Rivest R. L., Stein C. Introduction to Algorithms. 3rd ed. Cambridge, MA: The MIT Press, 2009. 1292 p.

[13] Приведём по этому поводу ссылку на недавнюю публикацию автора настоящей статьи – [12]. В конкретной прикладной задаче, рассматриваемой в той статье, необходимо работать с графами, состоящими из нескольких тысяч вершин; при этом даже простейшие алгоритмы на графах, которые, казалось бы, «вдоль и поперёк исследованы» уже несколько десятков лет назад, оказываются трудновыполнимыми для требуемых условий: очень много вершин графа, очень много (также несколько тысяч) независимых запусков программы с примерно одинаковыми входными данными…

[14] Кстати – конечно же, не надо уродовать русский язык «па́йтонами», «джа́вами» и т.п.; можно, например, «брать пример» с немецкого языка, в котором нормальное прочтение названий этих языков программирования существенно более популярно, чем английское. Несмотря на то, что в немецкоязычной Википедии «правильных немецких названий» языков программирования «с ходу» найти не удаётся (так, сайт про Яву там посвящён исключительно ей самой) – но из обсуждений на различных форумах немецких программистов можно заключить, что прочтение названия «ява» используется гораздо чаще. Как и, конечно, «цэ-плюс-плюс» – о чём мы уже говорили выше. Причём всё абсолютно аналогично – во французском языке: «жава» (а не «джава») и т.п.

[15] Добровольская В. РазвитОй, рАзвитый, развИтый – как разграничивать эти формы? [Электронный ресурс] // Аргументы и факты в Молдове. 25.10.2013. URL: https://aif.md/razvitoj-razvityj-razvityj-kak-razgranichivat-eti-formy (дата обращения: 16.05.2022).

[16] В [1] мы уже отмечали, что для «доступных» размерностей (примерно 70-100) при решении задачи случайного варианта ЗКВ максимум числа подзадач, одновременно находящихся в очереди, обычно превышает 200 000. Подробнее см. далее, в части II, – здесь лишь отметим, что, говоря об очереди, мы имеем в виду не обязательно FIFO-структуру в точности, но, возможно, некоторый близкий к ней вариант.

[17] Поиск в Яндексе (апрель 2022 г.) по словам «объектно-ориентированные возможности» дал только 1 точное совпадение, а всего менее результатов. (При этом замена слова «возможности» на одно из следующих слов / сочетаний: «программирование», «базы данных», «информационные технологии», «языки», «информационные интерфейсы» – даёт до нескольких миллионов результатов). Несмотря на это, мы считаем употребление такого сочетания – «объектно-ориентированные возможности» – вполне приемлемым.

[18] Конкретно – см. Shen A. Algorithms and Programming: Problems and Solutions. Springer Undergraduate Texts in Mathematics and Technology. Springer, New York, NY, 2010. 272 p. doi: https://doi.org/10.1007/978-1-4419-1748-5

Такой юмор, конечно, можно оценить… Но отметим, что как раз алгоритмы, приведённые в [2] (словом «программы» их вряд ли можно назвать) – конечно, хороши… Но для того, чтобы их довести до реальных программ – как раз и требуются «современные библиотеки объектов и классов». Более того, ещё лучше (гораздо лучше) – применять объектно-ориентированные возможности уже на этапе реализации алгоритма (а иногда – даже на этапе его разработки). Однако на всё это можно возразить, что нашу задачу тоже можно назвать «игрушечной»? Конечно, да[19], но, с нашей точки зрения (и в отличие от [2]), у нас – именно такие «игрушки», которые нужны для понимания не только «математики алгоритмов» но и – в неменьшей степени – их хорошей реализации[20].

Перейдём к непосредственному описанию организации всей нашей программы. В отличие от того, что для иллюстрации МВГ обычно изображают на рисунках[21] один подобный рисунок мы привели в [1]), мы для организации программы не используем структуры типа дерево[22] – однако мы всё-таки имеем его (дерево) в виду, но только мысленно, «в голове». А реально вместо дерева мы храним список его листьев – причём мы «нарушаем правила упорядочивания», которые для элементов этого списка можно вывести согласно[23]; более того, мы и в [1] описали «наши правила» далеко не полностью – подробнее см. далее, раздел 6. Однако отметим, что любой естественный алгоритм упорядочивания можно рассматривать как особую эвристику – и, по-видимому, только практическое исследование времени выполнения программы может дать ответ на вопрос, какая из этих эвристик лучше[24].

Чисто формально листья вышеупомянутого дерева[25] удобнее всего хранить в виде массива указателей на подзадачи[26] – но, конечно, возможны и другие варианты (список таких указателей и мн. др.). Конечно, нужно рассматривать массив с динамически изменяющейся размерностью. Вообще, как мы уже отмечали, подзадач получается «очень-очень много»[27] – немного подробнее см. ниже, в разделе 7 о результатах вычислительных экспериментов; поэтому естественным является размещение данных в динамической памяти[28].

Теперь становится понятной общая работа нашей программы. Во-первых («базис» алгоритма) – на основе входных данных (которые представляют собой просто матрицу) формируется задача, содержащая массив подзадач; первоначально этот массив состоит из единственной подзадачи (описывающей эти входные данные). Однако каждая подзадача – это не только сам матрица[29], но и некоторая дополнительная информация, которую можно полностью осознать на основе [1]:

- размерность подзадачи;
- граница (для инициализированной подзадачи –);
- номера строк и столбцов в матрице подзадачи (для инициализированной подзадачи – от до размерности и для строк, и для столбцов);
- полная информация об уже построенной части пути (подробнее см. ниже, разделы 4–5).

Во-вторых – шаг алгоритма:

1) выбираем задачу из массива (обычно самую первую);
2) если её не надо дорешивать сразу (т.е. обычно – в том случае, когда её размерность «не очень мала»), то:
   - выбираем в ней разделяющий элемент;
   - пытаемся для этого элемента разделить выбранную задачу на левую и правую подзадачи;
   - если удалось провести такое разделение – то формируем эти подзадачи, после чего – включаем их «назад» в список;
3) а если подзадачу надо сразу дорешивать, то:
   - дорешиваем; при этом получаем текущее решение, а также соответствующий ему цикл-тур;
   - сравниваем полученное решение с текущим псевдооптимальным;
   - возможно – заменяем на новые это текущее псевдооптимальное решение и соответствующий ему тур.

Всё! (Конец неформального описания алгоритма; а конец его работы – это пустой массив подзадач, т.е. невозможность выполнить подшаг 1.)

Итак, как мы уже отмечали, далее мы приводим практически полный текст работающей программы. Но наша цель, конечно, не в том, чтобы студент её просто повторил; мы можем сформулировать такие цели для студента (немного об этом мы уже писали во введении):

- познакомиться со стилем объектно-ориентированного описания математических алгоритмов;
- на основе приведённой программы – реализовать МВГ для некоторых других задач дискретной оптимизации (например, в нашем случае имеется много общего с описанием программы в [5]);

[19] Чтобы довести нашу программу до практического использования (например, при применении ЗКВ в задачах управления ядерным реактором, конкретных ссылок приводить не будем) – нужно, конечно, ещё немало сделать.

[20] Повторим, что это – мнение автора.

[21] Goodman S. E., Hedetniemi S. T. Introduction to the Design and Analysis of Algorithms. New York: McGraw-Hill, 1977. 371 p.

[22] Кстати, в будущем мы собираемся опубликовать статью про подход к объектно-ориентированному представлению деревьев произвольной структуры, про их использование в грамматическом разборе и других предметных областях, – но, повторим, для настоящего проекта они не нужны.

[23] Goodman S. E., Hedetniemi S. T. Introduction to the Design and Analysis of Algorithms. New York: McGraw-Hill, 1977. 371 p.

[24] Да и то, конечно, «неокончательный» ответ: надо не забывать, например, про большое количество вариантов ЗКВ, о которых было кратко рассказано в [1].

[25] Мы последний раз называем их таким образом.

[26] Про возможное применение стандартных «библиотечных» классов см. далее.

[27] Даже при «небольших» размерностях исходной задачи.

[28] «Динамические аллокаторы неизбежно приходят в состояние катастрофической фрагментации» – так?
Кириенко П. Динамическая память в системах жёсткого реального времени [Электронный ресурс] // Habr. 02.02.2020. URL: https://habr.com/ru/post/486650 (дата обращения: 16.05.2022).
При выполнении наших программ мы ничего подобного не наблюдали!

[29] Кстати, абсолютно аналогично – для всех остальных задач дискретной оптимизации, решаемых с помощью МВГ.

- также на её основе – попробовать «продвинуться» в решении некоторых более сложных программистских задач (выполняемых на основе нашей) – мы некоторые из них описали в заключении.

Далее переходим к непосредственному описанию программы.

Вспомогательные классы

Сначала приведём общие константы для всех функций и классов[30].

```
const int DIM_ALL = 99; // общая размерность всей задачи
const int DIM_ARR_SUB = 250000; // размерность массива указателей на подзадачи
const int INFTY = 999999; // бесконечность для присваивания
const int PEREBOR = 2; // если размерность меньше-или-равна, то полный перебор
const int TURBO = 6;
    // если размерность меньше-или-равна, то включаем в начало массива подзадач
const int MAX_VALUE = 999; // максимально возможное значение в исходной матрице
```

Некоторые из этих констант очевидны (приведённых комментариев в тексте программы достаточно) – но некоторые, по-видимому, всё-таки менее понятны; поясним их.

DIM_ARR_SUB – это размерность массива подзадач (точнее, указателей на подзадачи). Сам массив хранится в оперативной памяти – его сохранения на диск мы для простоты не делаем. Заметим, что на «обычных» современных персональных компьютерах значение этой размерности, равное , может уже не проходить трансляцию. (Но , по-видимому, почти всегда возможно – и мы использовали это значение для получения результатов вычислительных экспериментов, см. раздел 7.)

PEREBOR; смысл этой константы, по-видимому, почти полностью понятен из небольшого комментария в приведённом выше тексте программы. Если размерность подзадачи меньше или равна этой константе, то для её решения выполняется полный перебор, иначе – обычное выполнение очередного шага алгоритма МВГ. В нашей программе установлен самый простой – и не самый оптимальный! – вариант, PEREBOR = 2, но желающим (студентам и др.) при желании сто́ит попробовать и бо́льшие значения (однако, по-видимому, всё-таки не более 6). Конечно, с формальной точки зрения при этом возникнет факториальная сложность перебора, и, что не менее важно, сложнее будет написать функцию обработки – но всё это должно компенсироваться более быстрым выполнением всей программы[31].

TURBO; если значение размерности рассматриваемой подзадачи не превышает этого значения, то задачу сразу после формирования включаем в массив не по порядку значения границы – а в самое его начало. Это, среди прочего, даёт возможности реализовать как очень простой вариант построения ППЗ, так и некоторые другие эвристики.

MAX_VALUE; смысл этой константы, конечно, очевиден – но всё-таки приведём такой комментарий: при других вариантах случайного распределения значений матрицы[32] желательно полученные значения округлять до натуральных чисел, причём «обреза́ть» неположительные числа, а также значения, превосходящие эту константу.

Но в самом простом случае последняя константа применяется так:

```
int IntRnd() {
    return rand() % (MAX_VALUE+1);
}
```

(каждый вызов даёт значение очередного элемента матрицы ЗКВ – для её случайного варианта[33]).

Теперь перейдём к вспомогательным классам.

Описываемый первым класс Array[34] предназначен «для всего общего», что имеется в следующих двух классах, – в свою очередь предназначенных для:

---

[30] В текстах программ мы используем «стандартные цвета», предлагаемые программой Notepad для файлов языка Си++.
What is Notepad++ [Электронный ресурс] // Notepad++, 2022. URL: https://notepad-plus-plus.org (дата обращения: 16.05.2022).

[31] Однако отметим, что это не такая простая задача, как кажется на первый взгляд – и вот почему:
- во-первых, многие элементы рассматриваемой квадратной матрицы размерности, скажем, 6 (т.е. при значении константы PEREBOR = 6) в процессе решения равны ;
- во-вторых, входные номера строк и столбцов рассматриваемой для матрицы размерности 6 не дают возможность рассмотрения всех формируемых на основе этой матрицы циклов.

Итак, задача сама по себе очень интересная (создание быстрого алгоритма для подобных вычислений) – но, по-видимому, лишена какого бы то ни было практического смысла: непосредственные измерения времени выполнения показывают, что, например, даже для исходной размерности равной на решение таких «малых» задач тратится существенно менее процессорного времени.

[32] Точнее – при других вариантах ЗКВ. (См. термины в [1] – мы больше не будем заострять внимание на рассмотренной там терминологии.)

[33] А для других вариантов ЗКВ необходимы более сложные вспомогательные алгоритмы. Например, как мы отмечали в [1], возможный способ генерации случайных величин с нормальным законом распределения может быть построен на основе

Самарин А. Генераторы непрерывно распределенных случайных величин [Электронный ресурс] // Habr. 02.08.2015. URL: https://habr.com/ru/post/263993 (дата обращения: 16.05.2022).

Однако существуют «более простые» способы (при том, вполне приемлемые для практики) и, конечно, «более сложные».
- В качестве «более простого» можно использовать тот факт, что сумма н. о. р. с. в. с равномерным законом распределения на практике неотличима от нормального распределения, и др. А после получения таких сумм – либо с помощью несложных математических формул, либо с помощью суммы нескольких таких «нормально распределённых с. в.» – можно добиться нужной нам дисперсии, и, следовательно, нужного .
- Наоборот, в качестве «более сложного» способа можно использовать, например, т. н. преобразование Бокса-Мюллера и др.

[34] По-видимому, его можно назвать классом-прототипом. Однако чисто формально мы не оформляли его как абстрактный класс: это просто незачем.
Конечно, его – как и некоторые другие рассматриваемые здесь классы, в частности, массив указателей, – можно было бы реализовать на основе какой-либо библиотеки стандартных классов; например, для библиотеки MFC (которая, с точки зрения автора, «вовсе не устарела»). Для этих двух примеров возможна такая реализация:
- класс CWordArray (или его наследник) – для рассматриваемого здесь класса Array;
- CPtrArray (или его наследник) – для класса, описывающего поле Zadachi, представляющее собой массив указателей в рассматриваемом далее классе Task.

Однако, как мы уже отмечали во введении, мы выполнили реализацию алгоритма без вспомогательных библиотек (использованные нами stdlib.h, ctime и т. п. таковыми не считаем: их логично рассматривать в качестве элементов языка Си++).

- номеров строк/столбцов (класс Numbers ниже);
- формируемого/сформированного пути (класс Path ниже).

А в качестве такого «общего» (общих структур данных и общих алгоритмов) отметим:

- создание/удаление массивов неизвестной заранее размерности;
- обычные алгоритмы работы с такими массивами;
- индексация начиная с , очень удобная для математических алгоритмов[35];
- печать (вывод в файл);
- и др.

```
class Array { // просто массив; индексация с 1
protected:
    int nDim; // размерность этой подзадачи
    int* Arr;
    int MakeIndexSimple(int n) { return n-1; }
    void InitMemory() { Arr = new int[nDim]; }
public:
    Array(int nDim) { this->nDim = nDim; InitMemory(); }
    ~Array() { delete[] Arr; }
    void Set(int nInd, int N) { Arr[MakeIndexSimple(nInd)] = N; }
    int Get(int nInd) { return Arr[MakeIndexSimple(nInd)]; }
    void InitNull() {
        for (int i=1; i<=nDim; i++) Set(i,0);
    }
    void InitCopy(Array* arr) {
        for (int i=1; i<=nDim; i++) Set(i,arr->Get(i));
    }
    friend ostream& operator<<(ostream& os, Array& arr);
};
```

Вот некоторые комментарии к приведённому описанию класса Array.

- Самый важный комментарий – про размещение данных. Понятно, что такой вариант размещения элементов массива («на указателе») мы используем вследствие того, что размерность этого массива заранее неизвестна. А иначе – крайне нежелательно: (под)задач получается очень много, размерность создаваемой подзадачи заранее неизвестна…
- Комментарий про индексацию; он, кстати, связан с комментарием предыдущим. Кратко об индексации массивов начиная с (а не с) было сказано выше: она очень удобна для математических алгоритмов. Аналогично размещению данных «на указателе», такую индексацию мы используем для всех наших структур данных:
  - как в настоящем проекте — так и во многих других;
  - для тех структур данных, в которых рассматриваемый массив является единственным значимым элементом структуры (т.е. эта структура данных фактически ради него и заводится) — и для тех, где массив является «одним из многих».
- Про MakeIndexSimple(): эта функция сделана для следующих двух несвязанных вещей[36]:
  - во-первых – как раз для осуществления индексации начиная с ;
  - во-вторых – для «поддержки» инкапсуляции: мы осуществляем доступ к элементам массива (точнее, массивов-наследников) только с помощью пары методов Set() / Get(), а они оба работают с элементами массива только с помощью MakeIndexSimple().

Для нынешней реализации этот метод, конечно, может быть оформлен и как private. Немного забегая вперёд отметим, что в настоящем проекте в точности такой подход будет использован ещё один раз (а именно, для класса Subtask и «содержащейся в нём» матрицы элементов) – но, повторим, он очень удачен для практически любых математических объектов, описывающих массивы либо содержащих массивы внутри себя в качестве по́ля.

- Заранее отметим, что размерность конкретных массивов-наследников будет равна либо текущей размерности подзадачи, либо общей размерности задачи (в нашей реализации задаваемой с помощью константы).
- Применение protected, видимо, понятно: мы уже говорили, что рассматриваемый класс фактически является классом-прототипом.
- Про метод InitCopy(): мы «на указателе» размещаем не только массив как поле рассматриваемого нами класса – но и сам объект этого класса внутри классов «более сложных»; именно поэтому нам удобен такой «модифицированный конструктор копии».

Теперь перейдём к классам-наследникам описанного класса Array, используемым в нашей программе. Сначала приведём класс для записи перестановок (), Path:

```
class Path : public Array { // перестановка; размерность DIM_ALL; индексация с 1
public:
    Path() : Array(DIM_ALL) { }
    bool OK(); // является ли "большим" циклом
};
```

Итак – перестановки, ведь именно перестановкой проще всего задавать тур коммивояжёра. Поэтому здесь смысл -го элемента массива – это город, «в который мы поедем из -го»[37]. И понятно, что размерность здесь максимально возможная, поэтому родительский конструктор вызывается с максимально возможным значением параметра – константой DIM_ALL. При этом класс позволяет работать как с формируемой перестановкой, так и с уже сформированной.

Также понятно, что нам нужны только те перестановки, кото-

[35] По-видимому, индексация начиная с – это «самая большая беда» любых Си-подобных языков. Конечно же, для математических алгоритмов подобная индексация очень неудобна; например, вряд ли кому из «чистых математиков» придёт в голову в формулах линейной алгебры нумеровать строки и столбцы матриц начиная с …
Кстати, единственное известное автору исключение – это задачи, относящиеся к реализации т. н. длинной арифметики: в них, наоборот, удобна индексация начиная с . При этом мы в -й разряд (-й элемент массива) помещаем значение -ричной цифры (где – применяемое основание системы счисления), т.е. умножаем эту цифру на .

[36] «Одним махом семерых побивахом».

[37] Напомним, что сам массив «находится» в родительском классе.
Ещё отметим, что возможен и другой вариант представления перестановок, нередко используемый, например, в некоторых реализациях алгоритмов на графах: в нём значение -го элемент массива – порядок появления в ней -го города при поездке по циклу; в этом случае при удачном представлении данных и удачной реализации алгоритмов «малые циклы» (о которых речь пойдёт далее) не появляются совсем.

рые не могут быть разложены в произведение независимых циклов[38] (по [1] – «малых циклов»[39], будем и дальше употреблять такой термин). Поэтому, казалось бы, нужны специальные алгоритмы проверки отсутствия подобных «малых циклов»? Но, на самом деле, однозначного ответа на последний вопрос мы не дадим:

- с одной стороны – да, такие алгоритмы, конечно, существуют;
- но можно обойтись без них[40] – уже при получении какого-либо допустимого решения проверяя, не получили ли мы цикл, разложимый в произведение нескольких независимых;
- мы применяем «промежуточный вариант» – иногда проставляя значения для предотвращения «малых циклов», но оставляя окончательную проверку до момента получения допустимого решения[41];
- а действия, связанные с таким «промежуточным вариантом», мы не выполняем в рассматриваемом классе Path – а выносим в классы, использующие объекты этого типа (см. далее); в связи с этим приведённое выше описание класса Path довольно простое.

Метод класса OK() – проверка того, что цикл уже достроен, причём он неразложим в произведение нескольких независимых; таким образом, этот метод должен «отвергать не-циклы», т.е. следующие случаи:

- во-первых, оставшиеся нули в массиве: их наличие свидетельствует о том, что (под)задача ещё не досчитана;
- во-вторых, дублирование элементов массива[42];
- в-третьих, «малые» циклы (пример для 5 элементов – массив значений).

Однако проще сказать не «что должен отвергать», а «что должен принимать»; а принимать метод должен только «большие» циклы, из nDimAll элементов.

Наша реализация такова:

```
bool Path::OK() {
    int N = Get(1); // очередная вершина в цикле
    for (int K=1; K<=DIM_ALL; K++) { // номер вершины в цикле
        if (N<=1) return false;
            // 0 или менее - недостроен цикл; 1 - только после выхода из цикла
        N = Get(N);
    }
    return N==1;
}
```

Переходим к классу для записи номеров строк / столбцов текущей матрицы. Понятно, что здесь размерность массива заранее неизвестна – поэтому она задаётся параметром конструктора.

```
class Numbers : public Array { // номера строк/столбцов; размерность задаётся
public:
    Numbers(int nDim) : Array(nDim) { }
    int GetByNumber(int nNum); // 0 - если не нашли
};
```

Смысл оставшегося метода GetByNumber() и работа этого метода понятны на основе названия и приведённого небольшого комментария. А в теле этого метода мы используем самый обычный линейный поиск – а как иначе?

Также только на основе названия и приведённых комментариев понятен основной смысл последнего вспомогательного класса – класса для формирования результатов наших вычислений:

```
class Rezults {
private:
    int nIter; // всего было итераций
    int nSubT; // всего было подзадач (макс. число)
    int nomOpt; // номер на котором получили оптимум
public:
    Rezults() { nIter = 0; nSubT = 0; nomOpt = 0; }
    void NewIter(int newSubT) {
        nIter++;
        nSubT = max(nSubT,newSubT);
        if (nIter%1000==0) cout << *this;
    }
    void NewOpt() { nomOpt = nIter; }
    friend ostream& operator<<(ostream& os, Rezults& rez);
};
```

Объект этого класса будет использован в разделе 7, где мы и поясним его ещё немного подробнее (т.е. поясним смысл вычисляемых полей).

Класс для подзадач – общее описание и вспомогательные методы

В этом разделе подробно рассмотрен класс для подзадачи – конечно же, самый важный (и самый сложный) класс настоящего проекта.

```
class SubTask {
private:
    int nDim; // размерность этой подзадачи
    int nGran; // граница этой подзадачи
    Numbers* Lin; // номера строк
    Numbers* Col; // номера столбцов
    int* Matr; // сами элементы!
    Path* Next; // следующий город в "уже построенном" пути
    Path* Prev; // предыдущий город в "уже построенном" пути
```

Этот класс «не поместился на одну картинку», поэтому на первом приведённом рисунке – только его поля. К большинству из них комментарии не нужны (всё понятно на основе сказанного выше, названий самих полей, а также небольших комментариев в тексте программы) – дополнительно приведём только такие:

- массив Next используется для получения перестановки городов (в частости, для псевдооптимальных решений): если из города I мы едем в город J (т.е. установили

[38] Общая алгебра. Т. 1 / Под общ. ред. Л. А. Скорнякова. М.: Наука, 1990. С. 90.

[39] Включая и «малые циклы» длины . Впрочем, отсутствие последних нам гарантирует первоначальное проставление значений всюду на главной диагонали матрицы.
См. также соответствующие замечания в [16], [17], также посвящённых реализации метода ветвей и границ («мультиэвристического подхода») в разных задачах дискретной оптимизации.

[40] Причём «от слова совсем».

[41] Это объясняет «серый нуль» на рисунке [1, рис. 7] – в той статье мы объяснили его существование не полностью. Более того, в книге: Goodman S. E., Hedetniemi S. T. Introduction to the Design and Analysis of Algorithms. New York: McGraw-Hill, 1977. 371 p., способы полного решения этой проблемы не приводятся.

[42] Это – своего рода «перестраховка», впрочем, обычная для программирования (в реальных условиях при выполнении программы у нас такой ситуации ни разу не возникало).

необходимость такой поездки), то полагаем Next[I]=J;

- массив Prev – для получения обратной (к Next) перестановки: если из города I мы едем в город J, то полагаем Prev[J]=I.

А теперь применим необычный стиль изложения – приведём «комментарии к отсутствующим полям»! А именно, обычно в вариантах реализации МВГ ([5] и др.) бывают поля с условными названиями Yes и No:

- Yes – массив (или список) выбранных перемещений; при этом сам способ записи перемещения (поездки) произволен, а новое выбранное перемещение добавляется в массив при формировании новой правой подзадачи[43];
- No – аналогично, массив запрещённых («табуированных») перемещений; новое табуированное перемещение добавляется в массив при формировании новой левой подзадачи.

В общем случае – это поля, представляющие собой выбранные и табуированные разделяющие элементы соответственно. Отсутствие этих полей в нашем случае объясняется тем, что именно ЗКВ даёт возможность хранить о них информацию в само́й матрице – но, во-первых, это надо делать очень аккуратно, и, во-вторых, это не относится к другим задачам дискретной оптимизации, решаемым с помощью МВГ[44].

(Забегая вперёд, отметим также следующее. Имеется ещё и «самое главное отсутствующее» – это само дерево! Его хранить в памяти совершенно не нужно – мы как в этой программе, так и в подобных ей храним только его текущие вершины-листья. Всё это нужно будет реализовать в самом главном[45] классе Task. Мы, конечно, рассмотрим этот класс далее – но для понимания смысла класса SubTask и его объектов знать эту информацию желательно уже сейчас.)

Продолжим описание класса SubTask – в нашей реализации присутствуют следующие методы (см. рисунок ниже). Комментарии к ним такие.

- Второй из конструкторов – SubTask(int* Matr). Он работает только раз (при инициализации всей задачи) – а при порождении новой подзадачи мы его не вызываем (подробности см. ниже). Конечно же, мы обычно инициализируем эту первую подзадачу с помощью данных, взятых из некоторого текстового файла – однако там находятся только эти данные, т.е. элементы самой матрицы, а не все элементы подзадачи (и поэтому здесь мы не используем ввод из потока).
- MinLinDva() ищет второй по минимальности элемент в строке (это необходимо для варианта выбора «оптимального нуля», описанного в работе[46]); аналогично для столбца.
- SetInftyByNumbers() – наш метод, при других вариантах реализации он может быть не нужен; он устанавливает бесконечность согласно номерам строк и столбцов исходной матрицы (имеющимся в массивах номеров) – а не по номерам строк и столбцов матрицы подзадачи.

```
private:
    int MakeIndex(int nX, int nY) { return (nX-1)*nDim + (nY-1); }
    void InitMemoryMatrix() { Matr = new int[nDim*nDim]; }
public:
    SubTask(int nDim); // только выделение памяти без инициализации значений
    SubTask(int* Matr);
        // инициализация по уже имеющимся элементам (размерность DIM_ALL)
    ~SubTask();
    int GetLin(int I) { return Lin->Get(I); }
    int GetCol(int J) { return Col->Get(J); }
    void SetGran(int N) { nGran = N; }
    int GetGran() { return nGran; }
    int GetDim() { return nDim; }
    void Set(int nI, int nJ, int N) { Matr[MakeIndex(nI,nJ)] = N; }
    int Get(int nI, int nJ) { return Matr[MakeIndex(nI,nJ)]; }
    friend ostream& operator<<(ostream& os, SubTask& st);
private:
    int MinLin(int nX); // минимальный в строке
    int MinLinDva(int nX); // второй по минимальности в строке
    int MinCol(int nY); // минимальный в столбце
    int MinColDva(int nY); // второй по минимальности в столбце
    bool SetInftyByNumbers(int III, int JJJ);
        // устанавливаем "обратную бесконечность"
```

Реализацию нескольких простых методов опускаем. Остальные рассмотрим в том же порядке, что и в приведённом описании.

```
SubTask::SubTask(int nDim) {
    this->nDim = nDim;
    InitMemoryMatrix();
    Lin = new Numbers(nDim);
    Col = new Numbers(nDim);
    Next = new Path;
    Prev = new Path;
}
```

К первому конструктору, по-видимому, дополнительные комментарии не нужны.

Теперь второй конструктор:

```
SubTask::SubTask(int* Matr) {
    this->nDim = DIM_ALL;
    InitMemoryMatrix();
    nGran = 0; // ещё не вычислили
    Lin = new Numbers(DIM_ALL);
    Col = new Numbers(DIM_ALL);
    Next = new Path; Next->InitNull();
    Prev = new Path; Prev->InitNull();
    for (int i=1; i<=DIM_ALL; i++) {
        Lin->Set(i,i);
        Col->Set(i,i);
        for (int j=1; j<=nDim; j++) Set(i,j,Matr[MakeIndex(i,j)]);
    }
    Reduction();
}
```

Про параметры конструкторов. Поскольку первый конструктор будет вызываться постоянно (каждый раз при порождении новой подзадачи – кроме самой первой), то устанавливаемая размерность заранее неизвестна, она передаётся как параметр этого конструктора. А для второго конструктора – наоборот, размерность максимально возможная, она, как мы уже отмечали, задаётся константой размерности исходной задачи. В цикле мы переписываем элементы из Matr-параметра

[43] Сто́ит отметить, что в китайских учебниках по алгоритмам левая и правая (под)задачи обычно изображаются на дереве «наоборот»: левая справа, а правая слева. И это, по-видимому, более правильно! ведь наше мышление (да и работа программы) сначала рассматривает/решает правую подзадачу – а читаем-то мы слева направо (китайцы тоже)! Однако в литературе и на английском, и на русском языке названия «левая» и «правая», а также, соответственно, способ их отображения, к сожалению, стали стандартом – и мы не будем от него отходить.

[44] По-видимому, в случае хотя бы небольшого сомнения в правильности алгоритма, находящегося в процессе разработки, такие массивы (списки) сто́ит заводить.

[45] Но не самом важном! Как мы уже отмечали, самый важный класс – это рассматриваемый нами сейчас.

[46] Goodman S. E., Hedetniemi S. T. Introduction to the Design and Analysis of Algorithms. New York: McGraw-Hill, 1977. 371 p.

в Matr-поле – и сто́ит отметить, что при этом для Matr-параметра мы используем функцию MakeIndex(), первоначально для такого типа не предназначенную.

Продолжим описания методов класса SubTask. SetInftyByNumbers() – установка бесконечности вместо «серого нуля» на [1, рис. 7]:

```
bool SubTask::SetInftyByNumbers(int III, int JJJ) { // в обратном порядке
    int nX = Lin->GetByNumber(JJJ);
    if (nX<=0) return false;
    int nY = Col->GetByNumber(III);
    if (nY<=0) return false;
    Set(nX,nY,INFTY);
    return true;
}
```

При этом важно отметить следующее:

- устанавливается только одно значение, равное , – без применения дополнительных алгоритмов, которые, как мы уже отмечали, будут относиться уже к классам, описываемым далее;
- параметры передаются «в прямом порядке» (строка, потом столбец), а используются «в обратном»: всё правильно, мы используем установку значения для элемента, симметричного относительно главной диагонали исходной матрицы к элементу рассматриваемому;
- параметры суть номера первоначальной матрицы – поэтому используем метод GetByNumber().

Продолжим описания методов. Вот их последняя группа:

```
public:
    void ReductionLin(int nX); // по строке
    void ReductionCol(int nY); // по столбцу
    void Reduction(); // сначала по строкам, потом по столбцам
    bool BestNull(int& nXdel, int& nYdel);
    SubTask* MakeRight(int nXdel, int nYdel);
        // как обычно; nXdel и nYdel - удаляемые; модифицируем себя под левую!
    int Solve(Path*& Otvet);
        // вызываем только для малых размерностей (PEREBOR);
        // Otvet - заводится новый объект!
};
```

Комментарии такие.

- Reduction() – редукция по работе[47];
- BestNull() – выбор «наилучшего нуля», также по работе[48];
- MakeRight() – конечно же, это самый важный метод; он по заданному (ранее выбранному) нулю делает правую подзадачу («рождая» её с помощью вызова конструктора и возвращая указатель на неё) – после чего модифицирует себя под левую подзадачу;
- Solve() – решение «до конца» задачи малой размерности (после этого полученное значение тура будет сравниваться с текущим псевдооптимальным – но это уже будет сделано в классе Task); заранее отметим, что текст метода приводить не будем – некоторые пояснения были приведены выше при описании общих констант программы.

Далее – снова реализация.

```
void SubTask::ReductionLin(int nX) {
    int Min = MinLin(nX);
    if (Min==0) return;
    for (int j=1; j<=nDim; j++) {
        if (Get(nX,j)>=INFTY/2) continue;
        Set(nX,j,Get(nX,j)-Min);
    }
    nGran += Min;
}
```

Для редукции, по-видимому, не нужны никакие комментарии – но всё-таки текст одной такой функции из трёх мы приводим:

```
bool SubTask::BestNull(int& nXdel, int& nYdel) {
    int Max = -1;
    for (int i=1; i<=nDim; i++) for (int j=1; j<=nDim; j++) {
        if (Get(i,j)>0) continue;
        int nnn = MinLinDva(i) + MinColDva(j);
        if (nnn<=Max) continue;
        Max = nnn;
        nXdel = i; nYdel = j;
    }
    return Max >= 0;
}
```

Итак, BestNull() – алгоритм выбора «лучшего нуля» и возврат найденной пары координат с помощью параметров – реализован в точности по работе[49]; напомним, что всё это повторено, причём с бо́льшим числом примеров, в [1]. Здесь очень важно, что координаты возвращаются «именно так, как в рассматриваемой матрице» (т.е. их возможные значения – от до её размерности nDim) – а не «так, как в матрице исходной» (от до максимально возможной размерности DIM_ALL).

Как мы уже сказали во введении, мы продолжим описание программы в части II настоящей статьи. При этом начнём мы с самого важного метода – организующего правую подзадачу и изменяющего «себя» под подзадачу левую.

---

[47] Там же.

[48] Goodman S. E., Hedetniemi S. T. Introduction to the Design and Analysis of Algorithms. New York: McGraw-Hill, 1977. 371 p.
Как было сказано в [1], мы заменяем понятие «лучший нуль» так, чтобы наилучшим образом изменилась бы (т.е. увеличилась бы) граница левой подзадачи – вместо подобного увеличения разности границ, которое считается за значительно большее время.
Здесь может быть интересна такая аналогия про сослагательные наклонения в английском языке (как ни странно, многие студенты её воспринимают). Что бы ни говорили лингвисты – с нашей точки зрения их (этих сослагательных наклонений) в английском языке два – для реального и нереального желаний, https://grammarway.com/ru/subjunctive-mood и мн. др.
В теории алгоритмов подобное нереальное желание – это когда в алгоритме QuickSort заявляется, что хочется поставить рассматриваемый элемент на среднее место в массиве (а вероятность того, что он действительно является медианным, равна лишь); при этом и без выполнения такого желания алгоритм «всё-таки» работает неплохо. В то время как в нашем случае – желание реальное, поскольку оба возможных алгоритма вычисления «лучшего нуля» часто дают одни и те же ответы.

[49] Goodman S. E., Hedetniemi S. T. Introduction to the Design and Analysis of Algorithms. New York: McGraw-Hill, 1977. 371 p.

Список использованных источников

[1] Мельников Б. Ф., Мельникова Е. А. О классической версии метода ветвей и границ // Компьютерные инструменты в образовании. 2021. № 1. С. 21-44. doi: https://doi.org/10.32603/2071-2340-2021-1-21-45

[2] Shen A. Algorithms and Programming: Problems and Solutions. Springer Undergraduate Texts in Mathematics and Technology. Springer, New York, NY, 2010. 272 p. doi: https://doi.org/10.1007/978-1-4419-1748-5

[3] Melnikov B., Trenina M., Melnikova E. Different Approaches to Solving the Problem of Reconstructing the Distance Matrix Between DNA Chains // Modern Information Technology and IT Education. SITITO 2018. Communications in Computer and Information Science ; V. Sukhomlin, E. Zubareva (eds.). Vol. 1201. Springer, Cham, 2020. P. 211-223. doi: https://doi.org/10.1007/978-3-030-46895-8_17

[4] Мельников Б. Ф., Мельникова Е. А., Пивнева С. В. Пасьянс «Маджонг»: научный проект для старшеклассников с элементами искусственного интеллекта // Компьютерные инструменты в образовании. 2018. № 5. С. 41-51. doi: https://doi.org/10.32603/2071-2340-2018-5-41-51

[5] Абрамян М. Э., Мельников Б. Ф., Мельникова Е. А. Таблица состояний конечного автомата: научный проект для старшеклассников // Компьютерные инструменты в образовании. 2019. № 2. С. 87-107. doi: https://doi.org/10.32603/2071-2340-2019-2-86-107

[6] Gera R., Hedetniemi S., Larson C. Graph Theory: Favorite Conjectures and Open Problems – 1. Problem Books in Mathematics. Springer, Cham, 2016. 291 p. doi: https://doi.org/10.1007/978-3-319-31940-7

[7] Gera R., Haynes T.W., Hedetniemi S. Graph Theory: Favorite Conjectures and Open Problems – 2. Problem Books in Mathematics. Springer, Cham, 2018. 281 p. doi: https://doi.org/10.1007/978-3-319-97686-0

[8] Hromkovič J. Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics // Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin, Heidelberg, 2004. 538 p. doi: https://doi.org/10.1007/978-3-662-05269-3

[9] Gutin G., Punnen A. P. The Traveling Salesman Problem and Its Variations. Combinatorial Optimization. Vol. 12. Springer, Boston, MA, 2007. 830 p. doi: https://doi.org/10.1007/b101971

[10] Dorigo M., Gambardella L. M. Ant colonies for the travelling salesman problem. Biosystems. 1997. Vol. 43, issue 2. P. 73-81. doi: https://doi.org/10.1016/S0303-2647(97)01708-5

[11] Мельников Б. Ф., Мельникова Е. А. Кластеризация ситуаций в алгоритмах реального времени для задач дискретной оптимизации // Системы управления и информационные технологии. 2007. № 2(28). С. 16-20. URL: https://elibrary.ru/item.asp?id=11717006 (дата обращения: 16.05.2022).

[12] Оптимизационные задачи, возникающие при проектировании сетей связи высокой размерности, и некоторые эвристические методы их решения / А. Г. Булынин, Б. Ф. Мельников, В. Ю. Мещанин, Ю. Ю. Терентьева // Информатизация и связь. 2020. № 1. С. 34-40. doi: https://doi.org/10.34219/2078-8320-2020-11-1-34-40

[13] Melnikov B., Dudnikov V., Pivneva S. Heuristic Algorithm and Results of Computational Experiments of Solution of Graph Placement Problem // Modern Information Technology and IT Education. SITITO 2017. Communications in Computer and Information Science ; V. Sukhomlin, E. Zubareva (eds.). Vol. 1204. Springer, Cham, 2021. doi: https://doi.org/10.1007/978-3-030-78273-3_16

[14] Мельников Б., Эйрих С. Подход к комбинированию незавершенного метода ветвей и границ и алгоритма имитационной нормализации // Вестник Воронежского государственного университета. Серия: Системный анализ и информационные технологии. 2010. № 1. С. 35-38. URL: https://www.elibrary.ru/item.asp?id=15199645 (дата обращения: 16.05.2022).

[15] Макаркин С., Мельников Б. Геометрические методы решения псевдогеометрической версии задачи коммивояжера // Стохастическая оптимизация в информатике. 2013. Т. 9, № 2. С. 54-72. URL: https://www.elibrary.ru/item.asp?id=20960443 (дата обращения: 16.05.2022).

[16] Баумгертнер С. В., Мельников Б. Ф. Мультиэвристический подход к проблеме звездно-высотной минимизации недетерминированных конечных автоматов // Вестник Воронежского государственного университета. Серия: Системный анализ и информационные технологии. 2010. № 1. С. 5-7. URL: https://elibrary.ru/item.asp?id=15199639 (дата обращения: 16.05.2022).

[17] Мельников Б. Ф., Панин А. Г. Параллельная реализация мультиэвристического подхода в задаче сравнения генетических последовательностей // Вектор науки Тольяттинского государственного университета. 2012. № 4(22). С. 83-86. URL: https://elibrary.ru/item.asp?id=18755384 (дата обращения: 16.05.2022).

[18] Мельников Б. Ф., Мельникова А. А. Бесконечные деревья в алгоритме проверки условия эквивалентности итераций конечных языков. Часть I // International Journal of Open Information Technologies. 2021. Т. 9, № 4. С. 1-11. URL: https://elibrary.ru/item.asp?id=45595955 (дата обращения: 16.05.2022).

[19] Мельников Б. Ф., Мельникова А. А. Бесконечные деревья в алгоритме проверки условия эквивалентности итераций конечных языков. Часть II // International Journal of Open Information Technologies. 2021. Т. 9, № 5. С. 1-11. URL: https://elibrary.ru/item.asp?id=45671876 (дата обращения: 16.05.2022).

[20] Мельников Б. Ф., Мельникова Е. А. О классическом варианте метода ветвей и границ // Компьютерные инструменты в образовании. 2022. № 2. С. 41-58. doi: https://doi.org/10.32603/2071-2340-2022-2-41-58

[21] Melnikov B. F., Meshchanin V. Y., Terentyeva Y. Y. Implementation of optimality criteria in the design of communication networks // Journal of Physics: Conference Series. 2020. Vol. 1515. Article number: 042093. doi: https://doi.org/10.1088/1742-6596/1515/4/042093
[22] Pokorni S., Janković R. Reliability estimation of a complex communication network by simulation // 2011 19thTelecommunications Forum (TELFOR) Proceedings of Papers. IEEE Computer Society, 2011. P. 226-229. doi: https://doi.org/10.1109/TELFOR.2011.6143532
[23] Yu H., Oh J. Anytime 3D Object Reconstruction Using Multi-Modal Variational Autoencoder // IEEE Robotics and Automation Letters. 2022. Vol. 7, issue 2. P. 2162-2169. doi: https://doi.org/10.1109/LRA.2022.3142439
[24] Geldenhuys J., van der Merwe B., van Zijl L. Reducing Nondeterministic Finite Automata with SAT Solvers // Finite-State Methods and Natural Language Processing. FSMNLP 2009. Lecture Notes in Computer Science ; A. Yli-Jyrä, A. Kornai, J. Sakarovitch, B. Watson (eds.) Vol. 6062. Springer, Berlin, Heidelberg, 2010. doi: https://doi.org/10.1007/978-3-642-14684-8_9
[25] Yo-Sub Han. State Elimination Heuristics for Short Regular Expressions // Fundamenta Informaticae. 2013. Vol. 128, issue 4. P. 445-462. doi: https://doi.org/10.3233/FI-2013-952
[26] Grandcolas S., Pain-Barre C. A hybrid metaheuristic for the two-dimensional strip packing problem // Annals of Operations Research. 2022. Vol. 309, issue 1. P. 79-102. doi: https://doi.org/10.1007/s10479-021-04226-6

Поступила 16.05.2022; одобрена после рецензирования 25.06.2022; принята к публикации 04.07.2022.

Об авторе:
Мельников Борис Феликсович, профессор факультета вычислительной математики и кибернетики, Совместный университет МГУ – ППИ (517182, Китайская Народная Республика, провинция Гуандун, г. Шэньчжэнь, р-н Лунган, Даюньсиньчэн, ул. Гоцзидасюеюань, д. 1), доктор физико-математических наук, профессор, **ORCID: http://orcid.org/0000-0002-6765-6800,** bormel@mail.ru

Автор прочитал и одобрил окончательный вариант рукописи.

References

[1] Melnikov B.F., Melnikova E.A. On the classical version of the branch and bound method. *Kompjuternye instrumenty v obrazovanii* = Computer Tools in Education journal. 2021; (1):21-44. (In Russ., abstract in Eng.) doi: https://doi.org/10.32603/2071-2340-2021-1-21-45
[2] Shen A. Algorithms and Programming: Problems and Solutions. *Springer Undergraduate Texts in Mathematics and Technology.* Springer, New York, NY; 2010. 272 p. (In Eng.) doi: https://doi.org/10.1007/978-1-4419-1748-5
[3] Melnikov B., Trenina M., Melnikova E. Different Approaches to Solving the Problem of Reconstructing the Distance Matrix Between DNA Chains. In: Sukhomlin V., Zubareva E. (eds.) *Modern Information Technology and IT Education. SITITO 2018. Communications in Computer and Information Science*. Vol. 1201. Springer, Cham; 2020. p. 211-223. (In Eng.) doi: https://doi.org/10.1007/978-3-030-46895-8_17
[4] Melnikov B.F., Melnikova E.A., Pivneva S.V. Mahjongg Solitaire: a high-school student scientific project, containing elements of artificial intelligence. *Kompjuternye instrumenty v obrazovanii* = Computer Tools in Education journal. 2018; (5):41-51. (In Russ., abstract in Eng.) doi: https://doi.org/10.32603/2071-2340-2018-5-41-51
[5] Abramyan M.E., Melnikov B.F., Melnikova E.A. Finite automata table: a science project for high school students. *Kompjuternye instrumenty v obrazovanii* = Computer Tools in Education journal. 2019; (2):87-107. (In Russ., abstract in Eng.) doi: https://doi.org/10.32603/2071-2340-2019-2-86-107
[6] Gera R., Hedetniemi S., Larson C. Graph Theory: Favorite Conjectures and Open Problems – 1. *Problem Books in Mathematics.* Springer, Cham; 2016. 291 p. (In Eng.) doi: https://doi.org/10.1007/978-3-319-31940-7
[7] Gera R., Haynes T.W., Hedetniemi S. Graph Theory: Favorite Conjectures and Open Problems – 2. *Problem Books in Mathematics.* Springer, Cham; 2018. 281 p. (In Eng.) doi: https://doi.org/10.1007/978-3-319-97686-0
[8] Hromkovič J. Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. *Texts in Theoretical Computer Science. An EATCS Series*. Springer, Berlin, Heidelberg; 2004. 538 p. (In Eng.) doi: https://doi.org/10.1007/978-3-662-05269-3
[9] Gutin G., Punnen A.P. The Traveling Salesman Problem and Its Variations. *Combinatorial Optimization.* Vol. 12. Springer, Boston, MA; 2007. 830 p. (In Eng.) doi: https://doi.org/10.1007/b101971
[10] Dorigo M., Gambardella L.M. Ant colonies for the travelling salesman problem. *Biosystems*. 1997; 43(2):73-81. (In Eng.) doi: https://doi.org/10.1016/S0303-2647(97)01708-5
[11] Melnikov B.F., Melnikova E.A. *Klasterizatsiya situatsii v algoritmakh realnogo vremeni dlya zadach diskretnoi optimizatsii* [Situation Clustering in Real-Time Algorithms for Discrete Optimization Problems]. *Sistemy upravleniya i informatsionnye tekhnologii*. 2007; (2):16-20. Available at: https://elibrary.ru/item.asp?id=11717006 (accessed 16.05.2022). (In Russ., abstract in Eng.)

[12] Bulynin A.G., Melnikov B.F., Meshchanin V.Y., Terentyeva Y.Y. Optimization problem, arising in the development of high-dimensional communication networks, and some heuristic methods for solving them. *Informatizacija i svjaz'* = Informatization and Communication. 2020; (1):34-40. (In Russ., abstract in Eng.) doi: https://doi.org/10.34219/2078-8320-2020-11-1-34-40

[13] Melnikov B., Dudnikov V., Pivneva S. Heuristic Algorithm and Results of Computational Experiments of Solution of Graph Placement Problem. In: Sukhomlin V., Zubareva E. (eds.) *Modern Information Technology and IT Education. SITITO 2017. Communications in Computer and Information Science.* Vol. 1204. Springer, Cham; 2021. (In Eng.) doi: https://doi.org/10.1007/978-3-030-78273-3_16

[14] Melnikov B., Eyrih S.N. On the approach to combining truncated branch-and-bound method and simulated annealing. *Proceedings of Voronezh State University. Series: Systems Analysis and Information Technologies.* 2010; (1):35-38. Available at: https://www.elibrary.ru/item.asp?id=15199645 (accessed 16.05.2022). (In Russ., abstract in Eng.)

[15] Makarkin S.B., Melnikov B.F. *Geometricheskie metody reshenija psevdogeometricheskoj versii zadachi kommivojazhera* [Geometric methods for solving the pseudo-geometric version of the traveling salesman problem]. *Stohasticheskaya optimizaciya v informatike* = Stochastic optimization in informatics. 2013; 9(2);54-72. Available at: https://www.elibrary.ru/item.asp?id=20960443 (accessed 16.05.2022). (In Russ., abstract in Eng.)

[16] Baumgertner S.V., Melnikov B.F. Multi-heuristic approach for the star-height minimization of non-deterministic finite automata. *Proceedings of Voronezh State University. Series: Systems Analysis and Information Technologies.* 2010; (1):5-7. Available at: https://elibrary.ru/item.asp?id=15199639 (accessed 16.05.2022). (In Russ., abstract in Eng.)

[17] Melnikov B.F., Panin A.G. The parallel implementation of the multiheuristical approach in the nucleotide sequence comparison problem. *Science Vector of Togliatti State University*. 2012; (4):83-86. Available at: https://elibrary.ru/item.asp?id=18755384 (accessed 16.05.2022). (In Russ., abstract in Eng.)

[18] Vakhitova A. The basis automaton for the given regular language. *The Korean Journal of Computational and Applied Mathematics.* 1999; 6(3):617-624. (In Eng.) doi: https://doi.org/10.1007/BF03009953

[19] Hashiguchi K. Algorithms for determining the smallest number of nonterminals (states) sufficient for generating (accepting) a regular language. In: Albert J.L., Monien B., Artalejo M.R. (eds.) *Automata, Languages and Programming. ICALP 1991. Lecture Notes in Computer Science.* Vol. 510. Springer, Berlin, Heidelberg; 1991. (In Eng.) doi: https://doi.org/10.1007/3-540-54233-7_170

[20] Melnikov B., Melnikova E. On the Classical Version of the Branch and Bound Method. *Kompjuternye instrumenty v obrazovanii* = Computer Tools in Education journal. 2022; (2):41-58. (In Eng.) doi: https://doi.org/10.32603/2071-2340-2022-2-41-58

[21] Melnikov B.F., Meshchanin V.Y., Terentyeva Y.Y. Implementation of optimality criteria in the design of communication networks. *Journal of Physics: Conference Series.* 2020; 1515:042093. (In Eng.) doi: https://doi.org/10.1088/1742-6596/1515/4/042093

[22] Pokorni S., Janković R. Reliability estimation of a complex communication network by simulation. *2011 19thTelecommunications Forum (TELFOR) Proceedings of Papers.* IEEE Computer Society; 2011. p. 226-229. (In Eng.) doi: https://doi.org/10.1109/TELFOR.2011.6143532

[23] Yu H., Oh J. Anytime 3D Object Reconstruction Using Multi-Modal Variational Autoencoder. *IEEE Robotics and Automation Letters.* 2022; 7(2):2162-2169. (In Eng.) doi: https://doi.org/10.1109/LRA.2022.3142439

[24] Geldenhuys J., van der Merwe B., van Zijl L. Reducing Nondeterministic Finite Automata with SAT Solvers. In: Yli-Jyrä A., Kornai A., Sakarovitch J., Watson B. (eds.) *Finite-State Methods and Natural Language Processing. FSMNLP 2009. Lecture Notes in Computer Science.* Vol. 6062. Springer, Berlin, Heidelberg; 2010. (In Eng.) doi: https://doi.org/10.1007/978-3-642-14684-8_9

[25] Yo-Sub Han. State Elimination Heuristics for Short Regular Expressions. *Fundamenta Informaticae*. 2013; 128(4):445-462. (In Eng.) doi: https://doi.org/10.3233/FI-2013-952

[26] Grandcolas S., Pain-Barre C. A hybrid metaheuristic for the two-dimensional strip packing problem. *Annals of Operations Research.* 2022; 309(1):79-102. (In Eng.) doi: https://doi.org/10.1007/s10479-021-04226-6

Submitted 16.05.2022; approved after reviewing 25.06.2022; accepted for publication 04.07.2022.

About the author:

Boris F. Melnikov, Professor of the Faculty of Computational Mathematics and Cybernetics, Shenzhen MSU – BIT University (1 Guojidaxueyuan St., Dayunxincheng, Longgang District, Shenzhen 517182, Guangdong Province, People's Republic of China), Dr.Sci. (Phys.-Math.), Professor, **ORCID: http://orcid.org/0000-0002-6765-6800,** bormel@mail.ru

The author has read and approved the final manuscript.