Компьютерные инструменты в образовании, 2021
№ 1: 21–44
УДК: 37.01:007+004.82+004.85+519.713
http://cte.eltech.ru
doi:10.32603/2071-2340-2021-1-21-45

О КЛАССИЧЕСКОЙ ВЕРСИИ МЕТОДА ВЕТВЕЙ И ГРАНИЦ

Мельников Б. Ф.[1], доктор физико-математических наук, профессор,
✉ bf-melnikov@yandex.ru
Мельникова Е. А.[2], кандидат физико-математических наук, доцент,
✉ ya.e.melnikova@yandex.ru

[1]Совместный университет МГУ – ППИ, район Лунган, Даюньсиньчэн, улица Гоцзидасюеюань, д. 1, 517182, Провинция Гуандун, Шэнчжэнь, Китай
[2]Российский государственный социальный университет, ул. Вильгельма Пика, д. 4, стр. 1, 129226, Москва, Россия

Аннотация

В литературе описано очень много задач, которые могут быть названы задачами дискретной оптимизации: от шифровки информации в Интернете (включая создание программ для криптовалют) до поиска групп «по интересам» в социальных сетях. Часто эти задачи очень сложны для решения на компьютере, откуда и идёт их название «труднорешаемые». Точнее, сложны для решения (описания алгоритмов, программирования) возможные подходы к быстрому решению этих задач, переборное же решение, как правило, программируется просто, но работает соответствующая программа гораздо медленнее.

Практически каждую из таких труднорешаемых задач можно назвать математической моделью. При этом часто и сама модель, и алгоритмы, предназначенные для её решения, создаются для одной предметной области, но могут найти применение и во многих других областях. Примером такой модели является задача коммивояжёра. Особенностью задачи является то, что при относительной простоте её постановки нахождение оптимального решения (оптимального маршрута) является весьма сложным и относится к так называемому классу NP-полных проблем. Более того, согласно имеющейся классификации, задача коммивояжёра является примером оптимизационной проблемы, входящей в самый сложный подкласс этого класса.

Однако основной предмет статьи — это не задача, а метод её решения, метод ветвей и границ. Он состоит из нескольких связанных между собой эвристик, и в монографической литературе подобная мультиэвристичность метода ветвей и границ ранее отмечена, по-видимому, не была: разработчики алгоритмов и программ это должны были понимать сами. При этом сам метод может быть применён с небольшими изменениями и ко многим другим задачам дискретной оптимизации.

Итак, классический вариант метода ветвей и границ — основной предмет настоящей статьи, но почти столь же важен и второй её предмет — задача коммивояжёра, тоже в классической её постановке. Речь идёт о применении метода ветвей и границ при решении задачи коммивояжёра, причём в отношении этого применения также можно употребить прилагательное «классическое». Однако в дополнение к классической версии этой реализации мы рассматриваем новые эвристики, связанные с необходимостью разработки алгоритмов реального времени.

Ключевые слова: *оптимизационные задачи, задача коммивояжёра, эвристические алгоритмы, метод ветвей и границ, алгоритмы реального времени.*

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

1. ВВЕДЕНИЕ

Классический вариант метода ветвей и границ (МВГ) — основной предмет настоящей статьи, но почти столь же важен и второй её предмет — задача коммивояжёра (ЗКВ), тоже в классической её постановке. Иными словами, речь идёт о применении МВГ при решении ЗКВ, причём про это применение также (в третий раз!) можно употребить прилагательное «классическое». Однако в дополнение к классической версии этой реализации мы рассматриваем ещё и новые эвристики, связанные с необходимостью разработки алгоритмов реального времени.

Более точно — мы рассматриваем такие алгоритмы реального времени, которые в каждый определённый момент работы имеют лучшее (на данный момент) решение, при этом пользователь может просматривать эти псевдо-оптимальные решения в режиме реального времени, а последовательность таких решений в пределе даёт оптимальное решение[1].

Итак, мы в настоящей статье будем рассматривать задачу и алгоритм одновременно, но всё-таки «немного более важным» будем считать алгоритм. Совсем кратко можно сказать, что алгоритм метода ветвей и границ необходим по той причине, что он без больших изменений применим и во многих других задачах дискретной оптимизации.

2. О РАССМАТРИВАЕМЫХ ЗАДАЧЕ И АЛГОРИТМЕ

В литературе описано много задач (сотни? тысячи? больше?), которые могут быть названы *задачами дискретной оптимизации*. Про их практическое применение «не писал только ленивый»: от шифровки информации в Интернете (включая создание программ для криптовалют) до поиска групп «по интересам» в социальных сетях. Часто эти задачи очень сложны для решения на компьютере, откуда и идёт название «труднорешаемые»[2]. Точнее, сложны для решения (описания алгоритмов, программирования) возможные подходы к более-менее *быстрому* решению этих задач. Переборное же решение[3], как правило, программируется просто, но работает соответствующая программа гораздо медленнее.

С другой стороны, практически каждую из таких труднорешаемых задач можно назвать *математической моделью*. При этом часто и сама модель, и алгоритмы, предна-

[1] Русское название для таких алгоритмов, по-видимому, ещё не устоялось, и мы будем далее говорить «anytime-алгоритмы».

[2] Этот термин, «труднорешаемые задачи», впервые был применён, по-видимому, в названии русского перевода [1] — перевода книги, ставшей классической. К сожалению, термин не до конца утвердился в русском языке: Word и подобные ему программы слово «труднорешаемые» подчёркивают красным цветом.

[3] По-английски — “brute force method”. Калька — «метод грубой силы» — в русских публикациях иногда применяется, но, по-видимому, свидетельствует о нелюбви автора такой публикации к русскому языку. А слово «перебор», хотя также иногда подчёркивается Word-подобными программами, в русской литературе появилось в подобном смысле ещё до массового издания книг по теоретической информатике: так часто называется метод решения математических задач, причём нередко — даже «школьного уровня».

значенные для её решения, создаются для одной предметной области, но могут найти применение и во многих других областях. Примером такой задачи (модели!) является задача коммивояжёра ([1–4] и мн. др.[4]). Особенностью задачи является то, что при относительной простоте её постановки нахождение оптимального решения (оптимального маршрута) является весьма сложным и относится — причём как в обобщённой её постановке, так и для большинства вариаций — к так называемому классу NP-полных проблем. Более того, согласно классификации, приведённой в [4] и др., задача коммивояжёра является примером оптимизационной проблемы, входящей в *самый сложный подкласс*[5], так называемый подкласс NPO(V).

Однако мы отвлеклись. Ведь в названии статьи основная часть — про реализации некоторого алгоритма… Ответить на это можно так: да, *основной вопрос статьи* — про варианты реализации так называемого метода ветвей и границ; при этом алгоритм этого метода имеет очень много общего для самых разных задач дискретной оптимизации, а рассматриваем мы его на примере ЗКВ.

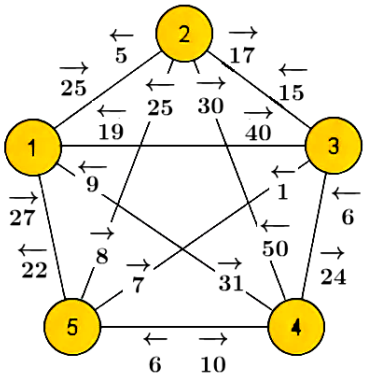

Рис. 1. Исходные данные

Приведём формулировку этой задачи. Задан полный граф (мы обычно будем рассматривать ориентированные графы — орграфы) из N вершин[6], каждая дуга которого помечена неотрицательным числом — мы всегда будем рассматривать только целые пометки[7]. Задача заключается в нахождении *цикла*, проходящего через все N вершин графа[8], причём у этого цикла стоимость (сумма пометок его дуг) должна быть минимально возможной. Пример входных данных — сразу отметим, что он взят из [2] — приведён на рис. 1.

Снова вернёмся к представлению входных данных. Далее мы для удобства вместо графовых представлений всегда будем использовать матричные. Итак, имеется матрица, часто обозначаемая C (см. рис. 2, который фактически повторяет данные, приведённые на рис. 1). Каждый элемент матрицы c_{ij} равен стоимости дуги, идущей из i-й вершины графа в j-ю, то есть стоимости «прямого проезда» из i-го города в j-й. Нам

	1	2	3	4	5
1	999	25	40	31	27
2	5	999	17	30	25
3	19	15	999	6	1
4	9	50	24	999	6
5	22	8	7	10	999

Рис. 2. Исходные данные, матрица

[4] Интересно отметить, что один из авторов книги [2] активно работает до сих пор: совсем недавно вышла его очень интересная монография-двухтомник [5, 6].

[5] Он содержит все оптимизационные задачи, для которых — при некотором дополнительном «естественном» предположении, например, «пресловутом» неравенстве P ≠ NP, — временна́я сложность всех возможных полиномиальных алгоритмов не может быть ограничена никакой полилогарифмической функцией. Другим примером подобной задачи — задачи самого сложного подкласса — является проблема максимальной клики.

[6] Вместо слова «вершина» мы также будем употреблять термины «точка» и «город». Очень важно, что в конкретных моделях в качестве «вершин» могут выступать не только физические объекты, но и объекты абстрактные (например — комбинаторные, в частности — перестановки), а также, например, процессы и т. п.

[7] Иногда рассматриваются графы, которые, вообще говоря, не являются полными. Такая возможность не усложняет задачу: например, в нашем случае мы просто можем рассматривать полный граф, в котором пометки «отсутствующих» дуг — очень большие числа.

Независимо от последнего, часто рассматриваются и неориентированные графы — об этом далее.

[8] Через каждую — ровно один раз; называется «тур коммивояжёра».

будет удобно считать, что $c_{ii} = \infty$ для всех i; это вовсе не «противоречит здравому смыслу» (ведь проехать из любого i-го города в него же, казалось бы, можно бесплатно): значение ∞ фактически запрещает переходы, ненужные в задаче, в частности, запрещает бесконечные циклы. Как мы видим, на рисунке бесконечность представляется числом 999; в реальных программах (к примеру, для размерностей порядка 100) этого значения для представления ∞, конечно, «не хватит», но очевидно, что требуемые значения для представления бесконечности всё-таки не очень велики.

Теперь от представления данных опять вернёмся к алгоритмам — и опять временно. Нашу задачу коммивояжёра, как и большинство задач дискретной оптимизации, всё-таки можно решать и «переборно», однако без особых успехов[9]. Именно поэтому для ЗКВ требуются сложные алгоритмы, и метод ветвей и границ является одним из таких.

Повторим, что основной предмет статьи — это не задача, а метод её решения, метод ветвей и границ. Он состоит из нескольких связанных между собой *эвристик*[10], и в монографической литературе подобная «мультиэвристичость» метода ветвей и границ ранее отмечена, по-видимому, не была: разработчики алгоритмов и программ должны были это «понимать сами». При этом сам метод может быть применён — с весьма небольшими изменениями — и ко многим другим задачам дискретной оптимизации.

3. О ВАРИАНТАХ ЗАДАЧИ КОММИВОЯЖЁРА И НЕКОТОРЫХ ЭВРИСТИЧЕСКИХ АЛГОРИТМАХ ЕЁ РЕШЕНИЯ

Чтобы *совсем немного* подробнее рассказать о других сложных алгоритмах, предназначенных для решения как ЗКВ в частности, так и задач дискретной оптимизации

[9] Здесь интересен такой «пример из 1990-х». Примерно в 1992 г. в компьютерных классах вузов и средних школ были машины с тактовой частотой около 10 МГц. На таком компьютере за реальное время (около 45 минут, то есть примерно за 1 академический час) удавалось переборно решить ЗКВ размерностью 13.

С тех пор прошло 30 лет, и теперь у средних современных ПК тактовая частота примерно в 300 раз больше. (Понятно, что скорость работы программы зависит не только от тактовой частоты компьютера, но всё-таки от неё в первую очередь.) Желающие могут проверить, что за примерно такое же время переборно можно решить ЗКВ размерностью 15: для такой проверки подойдёт практически любая программа, перебирающая все возможные перестановки и по каждой из них формирующая тур коммивояжёра.

Эти результаты хорошо согласуются с элементарной теорией: число перестановок факториально зависит от размерности, и в нашем случае произведение «двух новых размерностей» 14·15 примерно равно увеличению тактовой частоты доступных компьютеров. (Точнее, 13·14: один из городов, например первый, можно считать зафиксированным.)

[10] Фактически, эвристическим алгоритмам были посвящены наши предыдущие публикации в этом журнале [7, 8]: слово «эвристика» входила в ключевые слова обеих этих публикаций.

А метод ветвей и границ, как мы отметим ниже, — тоже эвристический алгоритм. И в связи с этим будет полезным текст «про эвристики вообще», взятый из [4] (цитата до конца этой сноски, перевод авторов настоящей статьи).

«В общем смысле эвристические алгоритмы являются непротиворечивыми алгоритмами решения оптимизационных задач. Они основаны на какой-нибудь ясной (но при этом обычно достаточно простой) идее — стратегии поиска в множестве всех допустимых решений; однако обычно такая стратегия не гарантирует, что какое-нибудь оптимальное решение найдётся. В этом смысле иногда говорят об эвристике локального поиска или жадной эвристике, даже если метод приводит только к аппроксимационному алгоритму.

А в узком смысле эвристика является методом получения непротиворечивого алгоритма, который для типичных частных случаев (входов) рассматриваемых оптимизационных проблем даёт допустимые решения приемлемого качества за приемлемое (например полиномиальное) время. Однако при этом не существует — или мы не знаем — строгого доказательства того, что алгоритм ведёт себя именно подобным образом; цель эвристики — «пообещать» хорошее поведение алгоритма для типичных входов рассматриваемой оптимизационной задачи».

вообще, нам сначала придётся ненадолго вернуться к представлению данных, точнее, к *вариантам* входных данных[11].

Говоря про входные данные, мы сначала приведём важное замечание по терминологии, которую, в отличие от некоторых других публикаций на русском языке, мы согласовываем с классическими публикациями, в первую очередь, с [1].

- *Вариант проблемы* (“subproblem” в английской литературе) определяется некоторым подмножеством всех возможных входов («аналогом для множеств» можно считать знак $\subseteq$). Иногда применяемые в русской литературе альтернативные термины для этого понятия — фактически кальки, «подпроблема» и «подзадача» — представляются значительно менее удачными: в литературе на русском языке они должны употребляться в другом смысле.
- А *частный случай проблемы* (“problem instance”) — это один конкретный возможный вход («аналог» — знак $\in$).

Именно про варианты проблемы (варианты ЗКВ в нашем случае) мы и будем говорить до окончания этого раздела. А по формируемым согласно описаниям вариантов ЗКВ алгоритмам генерации могут быть сформированы различные частные случаи ЗКВ — либо для срочной обработки некоторым алгоритмом, либо для записи в специальную базу данных, соответствующую этому варианту.

Итак, варианты ЗКВ. Мы уже отмечали, что нередко рассматриваются и *неориентированные* графы — и понятно, что при таком представлении данных в качестве матричного аналога должна рассматриваться матрица, симметричная относительно главной диагонали (понятно, что матрица, приведённая на рис. 2, таковой не является). Таким образом, можно рассматривать так называемый *симметричный* вариант ЗКВ[12] и представление данных для неё может быть либо матрицей, симметричной относительно главной диагонали, либо какой-либо иной структурой данных, представляющей верхнетреугольную матрицу. Иными словами, стоимость проезда между любыми двумя городами не зависит от направления. В любом случае, будем для симметричного варианта употреблять термин «ребро», а не «дуга».

В *метрической* ЗКВ выдвигается дополнительное требование о выполнении для любых трёх точек неравенства треугольника: для любых различных i, j и k

$$c_{ij} + c_{jk} \geqslant c_{ik},$$

при этом мы не забываем, что, вообще говоря, $c_{ij} \neq c_{ji}$ и т. п.[13]

Подвариантом как симметричного, так и метрического варианта ЗКВ[14], чаще всего применяемым в литературе по алгоритмизации, в том числе для создания *эвристических* алгоритмов, является так называемый *геометрический (Евклидовый)* вариант: в нём точки расположены на плоскости (например внутри единичного квадрата), а стоимость каждого ребра равна расстоянию между соответствующими точками. Понятно, что возможным

[11] Не будет ошибкой сказать, что мы до конца этого раздела будем рассматривать — очень кратко — возможные *алгоритмы генерации входных данных*.

[12] А «симметричной ЗКВ» будем называть какой-либо частный случай этого варианта. Аналогично и для других вариантов ЗКВ.

[13] А ещё интересно отметить, что в теории так называемых аппроксимационных алгоритмов часто рассматриваются ещё и специальная *количественная* характеристика, описывающая максимальную по матрице «степень нарушения» неравенства треугольника (см. [4] и др.)

[14] Крайне нежелательно говорить «частным случаем симметричного варианта ЗКВ» и т. п. (хотя в литературе такое иногда случается): слова «частный случай» желательно считать *термином*, определённым выше.

способом генерации частного случая такого варианта является случайное бросание точек в единичный квадрат (с применением равномерного распределения случайной величины) и передача на вход алгоритма решения ЗКВ вычисленной по таким правилам матрицы.

Отметим, что именно для геометрической ЗКВ создаются многие алгоритмы решения (см., например, две известные монографии [9, 10][15], а также главу в книге, которую, по-видимому, можно назвать самой известной книгой по алгоритмизации [11]). Мы ещё ненадолго вернёмся к алгоритмам для метрической ЗКВ, но уже сейчас заметим, что рассматриваемый нами метод ветвей и границ для неё всё-таки не очень подходит.

Теперь, после некоторых сведений про варианты ЗКВ, можно осознать такую «историю»[16]. В 1990-х годах для эвристического решения геометрической ЗКВ разрабатывались несколько *групп* алгоритмов, среди которых были «муравьиные» ([13] и др.) и «опоясывающие»[17]. Сначала про первые, муравьиные — будем писать без кавычек. При поиске путей к источникам пищи «настоящие» муравьи помечают пройденный путь специальным веществом — феромоном. А другие муравьи, попадая на такой путь, с большей вероятностью начинают двигаться по уже пройденному пути, и эта вероятность тем больше, чем больше концентрация феромона, которую каждый из муравьёв-последователей ещё более увеличивает. При применении муравьиных алгоритмов к частному случаю нашей задачи коммивояжёра они (муравьи) будут «ползать» по всем дугам графа[18], при этом *без нашего «приказа»* выбирая, в первую очередь, не только и не столько дуги с минимальной стоимостью (это совсем несложно), сколько «относительно дешёвые» пары дуг, тройки таких дуг и т. д. В конце концов при этом выстраивается требуемый тур коммивояжёра. И, как практически во всех эвристических алгоритмах, получаемые в разных частных случаях туры обычно не дают оптимальный результат, но дают результат, близкий к оптимальному.

Для понимания работы муравьиных алгоритмов (в значительно более простой постановке, чем подобные муравьиные алгоритмы для задачи коммивояжёра) рассмотрим рис. 3. Пусть (на левом рисунке) необходимо найти путь минимальной длины из самой верхней точки в самую нижнюю. Запускаем «сверху вниз» несколько «муравьёв» (A, B, C и D на участке 1 среднего рисунка) — каждый из которых на развилках будет выбирать дальнейшее движение случайным образом (в нашем случае — с вероятностью $\frac{1}{2}$ в каждую сторону, причём для простоты ещё и запретим любые движения в обратном направлении). Считаем, что каждый из муравьёв оставляет феромон равномерно по всей пройденной им дороге, и поэтому на правом рисунке весь феромон принадлежит единственному отрезку участка 1, этот отрезок выделен жирной линией.

Далее, на участке 2 половина муравьёв повернула налево (налево от нас, левая часть участка 2 — это A и B), а вторая половина — направо (C и D). Общий феромон двух «левых»

[15] Вторая из них формально является сборником статей, но фактически её можно рассматривать как полноценную «коллективную монографию».

[16] «Притча во языцех», “a folklore story”.

Точных ссылок на изложении этой истории авторам найти не удалось, однако приведённые по этому поводу факты (конкретнее — значения времени выпонения алгоритмов) могут быть проверены на основе публикаций групп исследователей, алгоритмов, построенных на основе этих публикаций, и т. п. Например, эти факты проверялись автором диссертации [12], и «результаты проверки» имеются во введении этой диссертации.

[17] Само название придумано авторами настоящей статьи, поэтому, в отличие от муравьиных алгоритмов, это назавание будем всё время писать в кавычках. Также сразу отметим, что точных ссылок на какую-то одну работу приводить не будем: подобные алгоритмы рассмотрены, например, в нескольких статьях из [10].

[18] Или рёбрам, если рассматривается неориентированный граф.

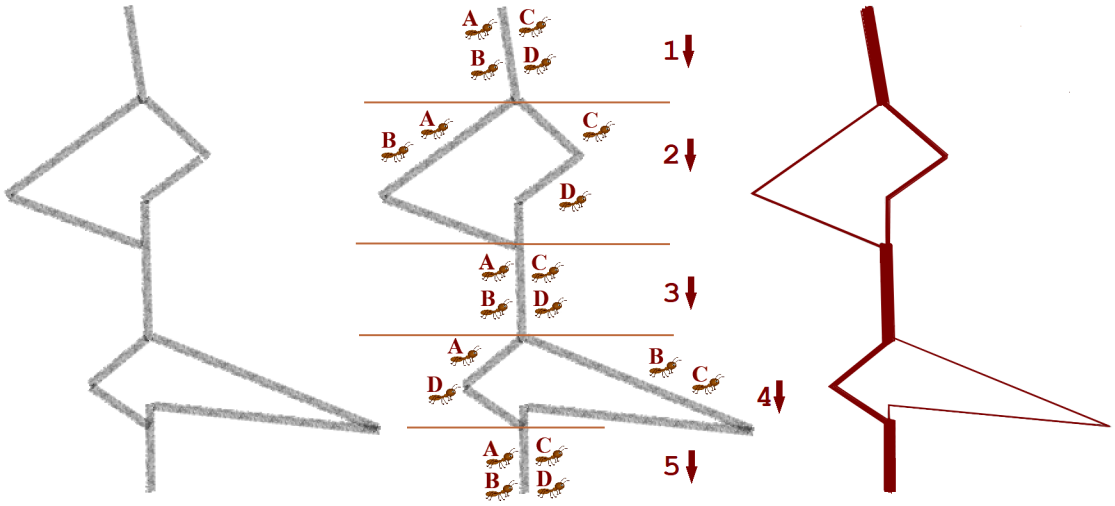

Рис. 3. Схематичное изображение муравьиного алгоритма

муравьёв «размазывается»[19] по более длинному участку, чем соответствующий общий феромон двух «правых», поэтому на участке 2 правого рисунка правая часть помечена более жирными линиями, чем левая.

И так далее. Оптимальный путь показан на правой картинке более жирными линиями. Понятно, что рассмотренный нами пример тривиален, но он хорошо отражает общую идею, и *нечто подобное* применяется и в значительно более сложных случаях, в том числе и в рассматриваемой нами задаче коммивояжёра — особенно в её геометрическом варианте.

Теперь перейдём к «опоясывающим» алгоритмам, причём здесь мы рассмотрим именно геометрическую ЗКВ (а не какую-то имеющую с ней аналогию, но сильно упрощённую задачу). Итак, пусть нам надо решить ЗКВ для частного случая, приведённого на рис. 4а.

Для решения мы с помощью некоторого вспомогательного алгоритма образуем «внешний контур» — красный на рис. 4б. Из оставшихся точек делаем «второй контур» — синий (двойная линия) на рис. 4б, и так далее. На рис. 4б разными цветами показаны все построенные контуры[20]. При этом отметим, что самый внутренний контур не обязан быть контуром в привычном нам смысле: он может вырождаться или в отрезок (как в нашем примере), или в точку; однако это не усложняет алгоритмы.

После построения всех контуров мы начинаем их *соединять*. Начало этого процесса показано на рис. 4в, продолжение — на рис. 4г, 4д. Мы не будем подробно обсуждать этот алгоритм: из рисунков видно, что выбираются две пары точек, находящихся на соседних контурах, и путём замены отрезков эти контуры соединяются. Понятно, что и этот алгоритм, и *апостериорная обработка* получаемого цикла коммивояжёра могут иметь очень много различных конкретных интерпретаций, но основной процесс здесь описан. Возможный окончательный вариант ответа алгоритма показан на рис. 4е.

[19] Необходимые формулы для такого «размазывания» придумать очень просто. Например, можно учитывать либо общее количество дуг / рёбер, либо суммарную длину пути. Важно, что *создание* такой формулы — это часто основная часть предмета конкретного муравьиного алгоритма.

[20] Очень интересная задача для выполнения студентами младших курсов: а сколько в среднем образуется таких контуров для N точек? (N очень велико — скажем, 1000000.) И далее — а каково распределение этой случайной величины (числа контуров), то есть, упрощая последнее, насколько вероятно большое отличие числа контуров от его вычисленного среднего значения?

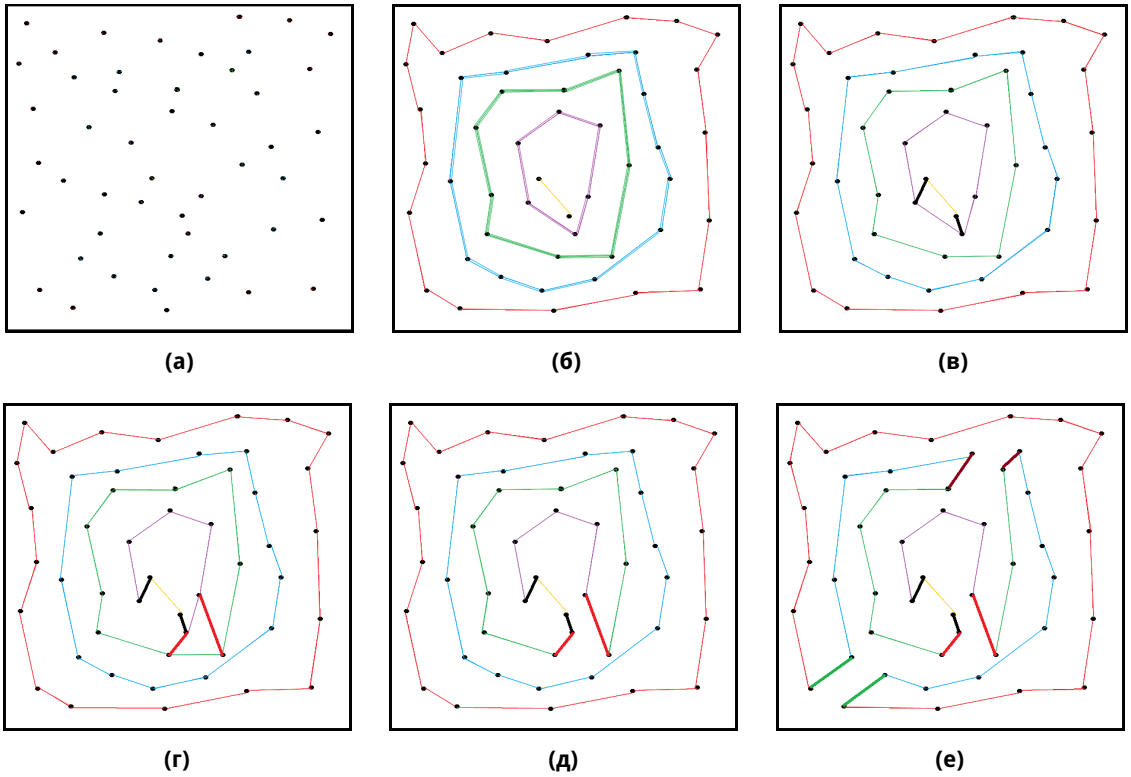

Рис. 4. Об «опоясывающем» алгоритме

Теперь вернёмся к «ранее начатой нами истории». Её окончание состоит в том, что «опоясывающие» алгоритмы на практике оказались существенно более эффективными, чем муравьиные, причём они дают возможность за реальное время получать приемлемые результаты (решения, близкие к оптимальным) для частных случаев ЗКВ с десятками тысяч (сотнями тысяч? миллионами?) точек.

Почему же тогда мы *не всегда* применяем «опоясывающие» алгоритмы? Ответ на этот вопрос очень прост: вся приведённая «история» оказывается верной только для геометрического варианта ЗКВ, а на практике часто приходится рассматривать и другие варианты. Итак, продолжим описание вариантов ЗКВ.

Случайный вариант[21] формируется путём применения равномерного распределения не к местоположению точек, а к генерируемым элементам матрицы; *может быть*, именно таким способом был получен рассматриваемый нами пример, приведённый на рис. 2[22]. А основной алгоритм настоящей статьи — метод ветвей и границ — может быть применён, конечно, к любому частному случаю любого варианта ЗКВ, но очень удобно считать, что мы применяем его к случайному варианту ЗКВ.

Последним рассмотрим *псевдогеометрический* вариант ЗКВ. Алгоритм генерации её частных случаев (как и в предыдущих ситуациях, совпадающий *с определением самого́ варианта*) заключается в следующем:

[21] Отметим, что его формулировка, как и для некоторых других вариантов ЗКВ, одновременно является и *алгоритмом генерации*.

[22] И вообще, про *любой* частный случай ЗКВ мы можем полагать, что он был сгенерирован как случай случайного варианта ЗКВ. Забегая вперёд, отметим, что то же самое можно сказать и о псевдогеометрическом варианте ЗКВ, который мы рассмотрим далее.

— в качестве входных данных выбирается не только размерность матрицы, но и некоторое значение σ;
— обычным образом генерируется «предварительная» матрица для геометрического варианта ЗКВ с выбранной размерностью; при этом возможен как симметричный случай (если заранее есть соответствующее условие), так и «несимметричный»;
— каждый элемент матрицы (или её верхнетреугольной части — при рассмотрении симметричного случая) умножается на случайную величину, выбираемую с нормальным распределением[23] с $\mu = 1$ и выбранным значением σ; при этом получающиеся отрицательные (либо неположительные) значения (см. рис. 5) просто «не засчитываются» и пересчитываются;
— возможно дополнительное требование — выполнение неравенства треугольника для всей получившейся матрицы (либо не очень большая «степень нарушения» этого неравенства); в этом случае должны быть применены какие-то дополнительные алгоритмы генерации, которые мы обсуждать не будем.

Конечно, про любой частный случай ЗКВ можно сказать, что он *был* сгенерирован с помощью алгоритма для псевдогеометрического варианта ЗКВ (мы это уже отмечали выше), но, в отличие, например, от случайного варианта ЗКВ, для псевдогеометрического варианта можно сказать следующее: при соответствующем выборе значения σ могут быть получены не только любые частные случаи, но и *различные варианты* ЗКВ:

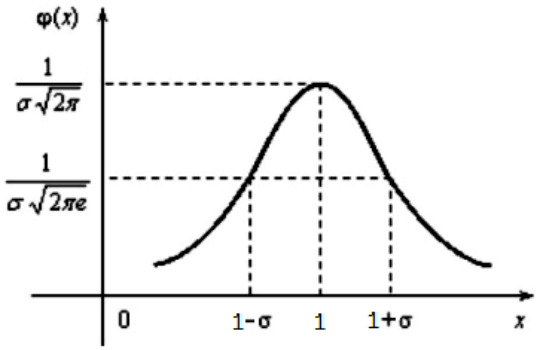

Рис. 5. Нормальное распределение, $\mu = 1$

— при $\sigma = 0$ мы получаем геометрический вариант ЗКВ,
— а при $\sigma = \infty$ (то есть достаточно большое число) мы получаем случайный вариант ЗКВ[24].

Для псевдогеометрического варианта задачи коммивояжёра также удаётся успешно применять МВГ.

Отметим также широкое применение приложений ЗКВ, встречающихся в литературе[25]; среди этих приложений отметим:

— оптимизацию в сетях [14, 15],
— оптимизацию маршрутов,
— приложения в кристаллографии,
— обработку печатных плат,
— управление роботами,
— исследование ДНК-цепочек [16, 17],
— и др.

[23] Возможный способ генерации таких случайных величин может быть построен, например, на основе изложенного здесь: https://habr.com/ru/post/263993/

[24] Повторим, что мы пересчитываем получающиеся отрицательные значения, а из рис. 5 можно видеть, что при таком возможном пересчёте математическое ожидание числа попыток получения неотрицательного значения элемента матрицы коммивояжёра меньше 2; этого, конечно, достаточно для использования такого алгоритма генерации на практике.

[25] Всё это можно было бы сказать в самом начале настоящей статьи. Мы приводим список возможных приложений здесь, поскольку для каждого из них применим МВГ, который начинаем рассматривать подробно.

Итак, мы считаем, что метод ветвей и границ вовсе не «устарел», он возможен для самых разных задач дискретной оптимизации. Как уже неоднократно говорилось, мы в настоящей статье рассмотрим его на примере задачи коммивояжёра.

4. ОПИСАНИЕ КЛАССИЧЕСКОГО ПОДХОДА ДЛЯ МЕТОДА ВЕТВЕЙ И ГРАНИЦ (ИЗЛОЖЕНИЕ ГУДМАНА-ХИДЕТНИЕМИ С ДОПОЛНЕНИЯМИ)

В качестве мини-предисловия к разделу приведём такую небольшую цитату из [2]:

«Алгоритмы ветвей и границ для задачи коммивояжера могут быть сформулированы разными способами. Авторы излагаемого алгоритма — Литл, Мерти, Суини и Карел. Это своего рода классика».

Однако авторам настоящей статьи ссылок на перечисленных авторов найти не удалось. Более того, по нашему мнению, на самом деле «классика» — это как раз книга Гудмана и Хидетниеми [2]! Но у изложения МВГ по [2], с нашей точки зрения, есть и недостатки[26], которые мы надеемся «исправить» нашей публикацией. В частности, в [2] не сказано, что *весь* алгоритм МВГ является эвристикой, а сами «малые» эвристики не разделяются на те, которые относятся к МВГ вообще[27], и те, которые относятся непосредственно к ЗКВ. Текст этого раздела имеет мало общего с [2]: только общий подход совпадает с изложенным в этой книге.

Для примера классического подхода к методу ветвей и границ будем рассматривать матрицу исходных данных, приведённую выше на рис. 2. Как мы уже отмечали, пример тоже взят из [2], причём он очень интересен[28].

Далее будем рассматривать и дерево поиска[29]. Строгое определение этого объекта (дерева поиска) приводить не будем — рассмотрим его в конкретной нашей задаче. Каждая вершина этого дерева — некоторое множество туров[30], корень дерева поиска помечен множеством «всех возможных туров», то есть в нашей задаче с 5 городами корень помечен множеством всех 4! возможных туров. Понятно, что в общем случае для любой «несимметричной» задачи с N городами корень помечен «полным множеством» — множеством всех $(N-1)!$ возможных туров.

Для такого дерева поиска рассмотрим так называемое *ветвление*[31]. Ветви (рёбра или дуги — здесь это непринципиально), выходящие из любой вершины (в частности, из корня), определяются выбором одной дуги[32] исходного графа, или — что то же самое — элемента рассматриваемой матрицы, скажем, дуги (элемента матрицы) (i, j)[33]. Наша цель состоит в том, чтобы разделить множество всех туров на два (под)множества:

[26] Что логично: книга-то издана более 40 лет назад.

[27] Их, *возможно*, сто́ит назвать «более важными».

[28] Забегая вперёд, отметим, что во-первых, пример интересен тем, что оптимальное решение получается «не там, где его ожидали» (при генерации вариантов случайной ЗКВ такие примеры для размерностей 5–6 получаются редко), во-вторых, мы досчитываем этот пример до конца (что, как и в предыдущей сноске, тоже логично: размерность-то совсем небольшая), но не забываем, что в реальных вычислениях примеры не обязательно досчитываются до конца (подробнее ниже).

[29] *Неявно* мы его уже рассматривали в наших предыдущих публикациях [7, 8].

[30] Точнее так: *пометкой* любой вершины является некоторое множество туров. По-видимому, все читатели знакомы с тем, что пометками вершин графа могут быть числа, но, как мы видим, пометками могут быть и существенно более сложные объекты!

[31] Это — эвристика *всего МВГ*!

[32] Всюду в тексте книги в подобных ситуациях «ребро» вместо «дуги» — это менее удачно.

[33] Вот что важно: мы пока не говорим, *как* выбирать эту дугу (i, j). Это в задаче коммивояжёра и есть *разделяющий* (разветвляющий, разрешающий) элемент.

— одно, которое, *весьма вероятно*, содержит оптимальный тур,
— и другое, которое, *весьма вероятно*, его не содержит.

Для этого мы выбираем такую дугу (i, j), что она, *как мы надеемся*, входит в оптимальный тур. После этого мы разделяем множество всех туров на два множества, которые мы обозначим $\{i, j\}$ и $\overline{\{i, j\}}$:

— в множество $\{i, j\}$ входят все туры, *содержащие* дугу (i, j) (и только они),
— в $\overline{\{i, j\}}$ — все туры, *не содержащие* (i, j) (и только они).

Уже сейчас должно быть понятно, что последовательное применение ветвления даст ответ (оптимальный тур). Но как выбирать дугу (i, j)[34] ? Мы вернёмся к этому вопросу немного позже.

В [2] *одновременно* с ветвлением рассматривается ещё один вспомогательный алгоритм, который, согласно приведённым ранее описаниям, также можно назвать эвристикой[35]: так называемая редукция[36]. По этому поводу снова приведём небольшую цитату.

«Если вычесть некоторую константу h из каждого элемента какой-то строки или столбца матрицы стоимостей C, то стоимость любого тура при новой матрице C' ровно на h меньше стоимости того же тура при матрице C. Поскольку любой тур должен содержать дугу из данной строки или данного столбца, то стоимость всех туров уменьшается на h … Пусть t — оптимальный тур при матрице стоимостей C. Тогда стоимостью тура будет

$$z(t) = \sum_{(i,j)\in t} c_{ij}\,.$$

Если C' получается из C редукцией строки (или столбца), то t должен остаться оптимальным туром и в C', причём

$$z(t) = h + z'(t),$$

где $z'(t)$ — стоимость этого же тура для C'».

После всего изложенного становится понятным описание ветвления, приведённое на рис. 6 (он выполнен близко к [2], однако мы его «немного улучшили», причём не только обозначения, см. ниже). На нём обозначено первое ветвление — получающееся:

— сначала — выбором разделяющего элемента $(3,5)$;
— далее — разделением на его основе исходной задачи на подзадачи $\{3,5\}$ (правая) и $\overline{\{3,5\}}$ (левая)[37];

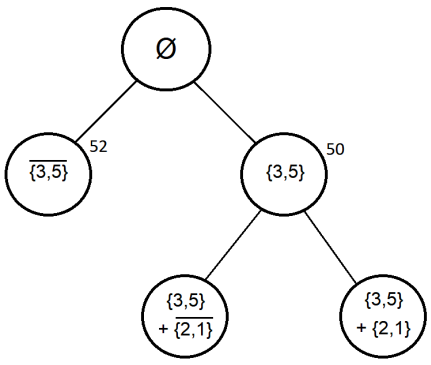

Рис. 6. Ветвление: начало выполнения МВГ

[34] По-видимому, это *самый важный момент*, причём не только для ЗКВ, но и вообще для любой реализации МВГ, то есть для любой задачи дискретной оптимизации.

[35] И, в отличие от предыдущей эвристики, то есть ветвления, этот вспомогательный алгоритм, наоборот, является *не* эвристикой всего МВГ, а «внутренней» эвристикой задачи коммивояжёра.

[36] В русском переводе [2] название другое — «приведение», не очень удачное, по мнению авторов настоящей статьи.

[37] Интересно отметить, что в некоторых книгах на китайском языке левая и правая подзадачи «меняются сторонами». По-видимому, такое «китайское» изложение более логично: ведь мы читаем слева направо, и поэтому естественно сначала выполнять ту подзадачу, которая на рисунке изображена слева (а ведь применяя МВГ, мы стараемся сначала выполнять ту, которая справа)… Однако мы, конечно, не будем отступать от традиций, принятых в английских изложениях и соответствующих русских переводах, а также в наших предыдущих публикациях.

— потом — выбором разделяющего элемента $(2,1)$ для правой подзадачи;
— и, наконец, — разделением на его основе правой подзадачи на подзадачи $\{3,5\}+\{2,1\}$ и $\{3,5\}+\overline{\{2,1\}}$.

Использованные обозначения, по-видимому, больших пояснений не требуют, однако необходимы следующие комментарии к самому алгоритму.

- Обозначение $\varnothing$ в качестве пометки корня означает, что в исходной задаче (точнее, в исходной *под*задаче, подзадаче-корне) ограничений ещё нет.
- Числа справа от вершин дерева — это границы (именно это слово и входит в название МВГ). Отметим по этому поводу, что сам алгоритм в [2] сформулирован недостаточно чётко: в каком порядке должно идти его выполнение — сначала ветвление или сначала редукция[38]?
- Эти границы вычисляются путём редукции; при этом мы всегда будем выполнять сначала редукцию по строкам, а потом — редукцию по столбцам[39].
- Для 2-го уровня[40] границы не считаем (ниже они нам не понадобятся).

Итак, приведём две получившиеся матрицы, соответствующие подзадачам $\{3,5\}$ и $\overline{\{3,5\}}$ и неявно уже использованные для получения границ на предыдущем рисунке. Здесь комментарии таковы:

	1	2	3	4	5	
1	999	0	15	6	2	25
2	0	999	12	25	20	5
3	13	9	999	0	999	6
4	3	44	18	999	0	6
5	15	1	0	3	999	7
				3		**52**

	1	2	3	4	
1	999	0	15	3	25
2	0	999	12	22	5
4	0	41	15	999	9
5	15	1	0	0	7
				3	**1+49=50**

Рис. 7. Начало МВГ: матрицы левой и правой подзадач

- Обе подзадачи приведены после редукции (о связанных с ней обозначениях см. ниже).
- В первой из получившихся подзадач (в правой) уменьшается размерность: мы же в ней точно уверены, что из 3-го города мы поедем в 5-й, поэтому мы из 3-го города не поедем никуда больше (и, следовательно, 3-я строка теперь не нужна). Аналогично, мы ниоткуда не приедем в 5-й город (и теперь не нужен 5-й столбец). Таким образом, размерность 5 превращается в размерность 4, и уже после этого мы делаем редукцию.
- Во второй из получившихся подзадач (в левой) размерность не уменьшается, но, поскольку поездка из 3-го города в 5-й запрещена (ещё говорят «табуирована»), мы проставляем ∞ как новое значение c_{35}. Как и в предыдущем случае, мы делаем редукцию, причём уже после описанных здесь действий.

[38] Мы ниже приведём возможный ответ на этот вопрос, то есть возможный алгоритм, сформулированный чётче.

[39] Приведём также *неформальное* объяснение редукции — сначала по строке. Была матрица коммивояжёра, и будем считать, что деньги за проезд коммивояжёр платит водителю автобуса, везущего его между городами. Но вот мэр i-го города решил тоже собирать деньги за проезд и со всех *уезжающих* из этого города водителей берёт налог, равный минимальной стоимости проезда из этого города. А водители автобусов перебрасывают этот налог на коммивояжёров-пассажиров, и понятно, что пассажиру всё равно, кому платить (всю сумму водителю, либо часть этой суммы водителю и часть мэру — общая сумма одна и та же); всё это и есть редукция по i-й строке. Аналогично — про столбцы и автобусы, *приезжающие* в j-й город.

Также про редукцию нужно сказать следующее. Применение редукции сначала по строкам, а потом по столбцам даёт, вообще говоря, другие результаты, чем её применение в обратном порядке.

[40] Как обычно в теории графов, уровни вершин дерева считаются, начиная с 0 (0-й уровень — корень).

- В связи с возможным уменьшением размерности мы должны знать номера строк и столбцов получаемых подзадач[41]. На рисунках эти номера набраны жирным шрифтом — слева для строк и сверху для столбцов.
- Продолжим говорить про редукцию. Как несложно догадаться, справа в каждой строке и внизу в каждом столбце мы ставим число, вычитаемое при редукции (нули пропускаем); сумма всех этих чисел — *увеличение границы* (по сравнению с подзадачей-родителем). Мы на картинках приводим эта сумму жирным шрифтом в правом нижнем углу матриц (значения 52 и 49 на рис. 7).
- 1+49 означает, что к увеличению границы добавляется ещё и стоимость выбранной дуги (вспомним, что в исходной матрице $c_{35} = 1$) — в левой задаче такого слагаемого не нужно.
- Элемент правой матрицы с координатами $(5, 3)$ (это номера строки и столбца *исходной* матрицы) мы изобразили на сером фоне — и вот почему. Мы могли бы сразу вместо этого элемента записать ∞ (ведь поездка «в обратном направлении» становится невозможной), но при этом мы столкнулись бы с такой проблемой: нужно было бы *детально* описать такой вспомогательный алгоритм «запрещения обратных поездок», ведь нам следовало бы следить за «непоявлением» «малых циклов» не только длины 2, но и 3, 4 и т. д. В настоящей статье мы *совсем* не будем описывать этот вспомогательный алгоритм, вернёмся к нему в следующей статье[42].
- На рисунках (в том числе в статье-продолжении) будет ещё по одному слагаемому (мы его будем приводить самым первым), представляющему собой границу подзадачи-родителя; в нашем примере мы для простоты (и не противореча [2]) не делали таких вычислений для подзадачи-корня. Таким образом, ниже и в левых, и в правых подзадачах у суммы для вычисления границ обычно будет 2 слагаемых (старая граница плюс значение редукции)[43].
- И немного про эвристики: мы действительно «надеемся», что оптимальное решение ожидается в правой подзадаче — это показывает и меньшая её размерность, и меньшая её граница, вычисленная нами.

Повторим, что *последовательное применение ветвления даст ответ (оптимальный тур)*. А удачные алгоритмы выбора разделяющего элемента[44] позволяют построить этот тур достаточно быстро. Но мы ещё не говорили, за счёт чего это происходит, или, переформулируя и частично отвечая на этот вопрос: надо осознавать, *зачем* нам нужны границы (то есть *как их применять*). По этому поводу снова приведём небольшую цитату из [2]:

«С каждой вершиной дерева мы связываем нижнюю границу стоимости любого тура из множества, представленного вершиной. Вычисление этих нижних границ — основной фактор, дающий экономию усилий в любом алгоритме типа ветвей и границ[45]. Поэтому особое внимание следует уделить получению как можно более точных границ. Причина

[41] Уже сейчас понятно, что при последовательном решении ЗКВ таких уменьшений размерности будет много, причём заранее непонятно, какие именно строки и столбцы будут удалены. Мы, кроме того, должны не только знать номера строк и столбцов получаемых подзадач, но и хранить эти номера при обработке данных. Мы предполагаем подробно написать об этом в следующей статье.

[42] Заметим лишь, что при его применении, вообще говоря, сильно изменяется порядок вычисления: например, на рис. 7 нам было бы необходимо выполнить *дополнительную* редукцию на 12 (в 3-м столбце), что привело бы к другому порядку границ в получающихся левой и правой подзадачах, и так далее.

[43] Здесь на правом рисунке также 2 слагаемых, но по другой причине: старая граница равна 0, однако есть ещё и ненулевая стоимость дуги.

[44] А также другие эвристики, к которым мы вернёмся в следующих разделах.

[45] Так по тексту книги. Но с точки зрения авторов настоящей статьи, основной фактор — это выбор разделяющего элемента. (Мы только что отмечали этот факт.)

этого следующая. Предположим, что мы построили конкретный полный тур со стоимостью m. Если нижняя граница, связанная с множеством туров, представленных вершиной v_k, равна $M \geqslant m$, то до конца процесса поиска оптимального тура не нужно рассматривать вершину v_k и все следующие за ней (её потомки)».

5. ТО, ЧТО МЫ ЕЩЁ «НЕДОРАССМАТРИВАЛИ»

В этом небольшом разделе мы просто перечислим «ещё недорассмотренные» вопросы, которые обычно рассматриваются одновременно с описанием алгоритма метода ветвей и границ. Мы считаем, что более логично рассмотреть их:

— или в оставшихся разделах настоящей статьи (в тех случаях, когда для их понимания нужны конкретные примеры, либо они выходят за рамки «стандартного МВГ»),
— или в статье-продолжении (это, прежде всего, вопросы, непосредственно связанные с созданием компьютерных программ).

Самый первый возникающий вопрос — почему мы пример, который начали рассматривать, не довели до конца? Ответов на него два: во-первых, он доведён до конца в [2][46], и, во-вторых, мы ниже доведём до конца этот же пример, но для немного модифицированного (улучшенного) алгоритма.

А *самый важный* из этих вопросов — эвристика выбора «оптимального нуля»[47]. О каких нулях идёт речь? Дело в том, что после редукции матрицы ЗКВ размерности N в ней получается не менее N элементов, равных 0[48]. Так почему же в примере, который мы начали рассматривать выше, мы выбирали элемент c_{35}? Сильно изменяя [2], можно сказать, что мы выбрали этот элемент потому, что он в исходной матрице минимален; такая «эвристика», конечно, возможна… но она действительно является «эвристикой в кавычках»: ведь после редукции соответствующий элемент (его логично обозначать c'_{35}) «ничем не лучше» и «ничем не хуже» любого из оставшихся нулей, точнее, нужен дополнительный алгоритм, *дополнительная эвристика*, для того чтобы из нулей выбрать «лучший». Подробнее такой вспомогательный алгоритм (ответ на поставленный вопрос) мы рассмотрим в следующем разделе при обработке конткретного частного случая ЗКВ, причём при его обработке с помощью основного алгоритма, являющегося, как мы уже сказали, *улучшением* приведённого в [2]; но кратко на поставленный вопрос ответим прямо сейчас.

Наша цель — упорядочить, «проранжировать» нули[49]. И можно сказать (это очередная эвристика), что лучше — тот из нулей, при котором увеличение границы левой подзадачи (пусть это L[50]) как можно больше превосходит увеличение границы правой подзадачи

[46] Причём, к сожалению, с некоторыми недочётами-опечатками.

[47] Терминология авторов настоящей статьи.

[48] Будем и далее называть такие элементы матрицы нулями.

И возникают несколько интересных вопросов, «находящихся между» дискретной математикой, комбинаторикой, математической статистикой, имитационным моделированием и программированием; приведём такие два. Сколько *в среднем* нулей возникает? Какова вероятность, что эти нули *сразу* дадут решение — то есть что они образуют цикл-ответ? (В этой сноске матрица имеет размерность N, которая и должна войти в ответ, причём возможны разные варианты ЗКВ, рассмотренные выше.)

[49] И отметим заранее, что «проранжировать нули» правильнее, чем «найти лучший нуль»; это можно объяснить следующим образом. Иногда применяются модифицированные эвристики, связанные с тем, что мы *предварительно* выбираем несколько «лучших» нулей и *уже потом* среди них с помощью *каких-то других* эвристик выбираем единственный, который и станет разделяющим элементом. Такими «другими» эвристиками может быть «похожесть» получаемой матрицы на некоторую рассматривавшуюся ранее [17], применение функций риска [18–20] и др.

[50] Обозначение не очень удачное: ведь L зависит от конкретного нуля. Но, конечно, смысл понятен.

(пусть это R); то есть самым удачным выбором нуля был бы такой, при котором

$$L - R \longrightarrow \max. \tag{5.1}$$

Последнюю эвристику легко объяснить: ведь, как мы уже говорили, для более быстрого решения задачи желательно, чтобы оптимальное решение с большей вероятностью оказывалось бы в правой подзадаче.

Однако на основе рис. 7 можно понять, что вычисление L выполняется значительно быстрее, чем вычисление R: на L тратится порядка N операций, а на вычисление R — порядка N^2; и мы такие вычисления должны повторять для каждого ветвления в МВГ для каждого нуля этого ветвления… Поэтому вместо (5.1) принято «от нашей бедности»[51] использовать такой упрощённый вариант:

$$L \longrightarrow \max.$$

Последняя запись, как несложно понять, представляет собой максимизацию суммы двух элементов: минимального элемента в соответствующей строке (в той, где стоит «рассматриваемый нуль», причём при выборе минимума мы «этот нуль» не считаем) и минимального элемента в соответствующем столбце. Это значение (для каждого нуля) будем называть badness[52].

Также мы пока не будем рассматривать вспомогательный алгоритм «слежения» за тем, чтобы при работе МВГ не получались бы «малые циклы» (таким циклом можно назвать, например, выбор элементов c_{12}, c_{23} и c_{31} при общей размерности матрицы не менее 4). Мы предполагаем вернуться к этому вопросу в следующей статье, где будем рассматривать программную реализацию МВГ.

И наконец отметим, что, конечно, МВГ только «в среднем»[53] улучшает время вычисления (по сравнению с переборными алгоритмами); время вычисления «в худшем случае» остаётся таким же[54]. Чтобы относительно быстро получать решения, близкие к оптимальному, часто используется так называемый *незавершённый* (truncated) МВГ; в нём фиксируется решение, соответствующее минимальной имеющейся в некоторый момент работы алгоритма границе, и при необходимости завершения вычислений (например, в связи с истечением отведённого времени) это решение выдаётся пользователю в качестве *псевдооптимального*. К незавершённому МВГ мы вернёмся в следующем разделе.

6. СОВРЕМЕННЫЕ ДОПОЛНЕНИЯ К КЛАССИЧЕСКОМУ ПОДХОДУ. ANYTIME-АЛГОРИТМЫ РЕАЛЬНОГО ВРЕМЕНИ И ПОСЛЕДОВАТЕЛЬНОСТЬ ПРАВЫХ ЗАДАЧ

В начале этого раздела приведём несколько слов про так называемый *anytime-алгоритмы*. Как уже было сказано во введении, ими являются такие алгоритмы

[51] В хорошем смысле этого слова: мы же в любых ситуациях должны экономить время вычислений, а особенно делать это в так называемых «узких моментах», то есть в тех местах алгоритмов, которые при работе программы выполняются значительно чаще остальных.

[52] По-видимому, удачного русского аналога нет; например, слово «штраф» вряд ли уместно. А слово “badness” мы используем в подобных ситуациях и во многих других предметных областях, например, для вышеупомянутого исследования ДНК-цепочек [16, 17] и др.

[53] Про оценки времени «в среднем» и «в худшем» см., например, [21].

[54] По-видимому, оно даже немного увеличивается, например, из-за необходимости постоянно вызывать вспомогательные алгоритмы, к примеру, алгоритм выбора разделяющего элемента.

реального времени, которые в каждый определённый момент работы имеют лучшее (на данный момент) решение, и при этом пользователь может просматривать эти псевдо-оптимальные решения (также в режиме реального времени), и при этом последовательность таких решений в пределе даёт оптимальное решение.

Видимо, более подробных определений не нужно. Но вот возможный *пример* их практического применения (он, кстати, подходит для любой труднорешаемой задачи). Пусть мы оцениваем время выполнения всей задачи в 3 месяца[55], а заказчику (начальнику, пользователю) хочется получить *хоть какое-то* допустимое решение (конечно, желательно, чтобы оно было более-менее близко к оптимальному) гораздо скорее, скажем, после 1 часа работы программы. Если же полное время выполнения всей задачи существенно больше 1 часа, то какие-то близкие к оптимальным решения начинают получаться «почти» за 3 месяца, скажем, за 2 месяца 28 дней[56]. Что же делать? Вот для этого и используют дополнительные эвристики, дающие получение пусть даже «более далёкого» решения (чем если бы работал обычный МВГ), зато очень быстро. Одной из таких эвристик (фактически — модификаций МВГ) является использование так называемых 1-деревьев[57], а другая — так называемую *последовательность правых задач* (ППЗ), см. ниже рис. 8.

Эта эвристика заключается в следующем. Каждый раз при выборе очередного разделяющего элемента в некоторой задаче (пусть это T; можно также сказать, при получении очередной правой задачи) мы при её применении фактически строим такую последовательность:

— саму задачу T,
— правую (под)задачу задачи T,
— правую задачу правой задачи задачи T,
— и так далее.

Естественно, каждый раз строятся (и включаются в список задач для потенциального решения в последующем) и соответствующие левые задачи:

— левая задача задачи T,
— левая задача правой задачи задачи T,
— и так далее.

Описанный процесс заканчивается:

— либо при получении тривиальной задачи (например, задачи нулевой размерности): в этом случае мы запоминаем её решение (границу, получаемый к моменту её постановки тур и другие *характеристики*) в качестве *текущего на данный момент времени псевдооптимального решения* нашего anytime-алгоритма;
— либо при получении в какой-либо задаче достаточно большой границы — например, большей, чем имеющееся на данный момент времени псевдо-оптимальное решение[58].

[55] Для труднорешаемых задач это совсем не редкость. Иногда общее время выполнения переборного алгоритма (да и рассматриваемого нами МВГ) в худшем случае оценивается значительно бóльшими временными интервалами — а решать задачу надо…

[56] И это — вовсе не преувеличение, а реальные оценки. Действительно, вспомним «классическое» описание МВГ (достаточно той части описания, которую мы уже рассмотрели): первыми выполняются подзадачи, имеющие меньшие значения границ и, вследствие этого, как правило, бóльшие размерности. При этом задачи с меньшими размерностями «отодвигаются в конец» что и приводит к подобному сроку получения самых первых результатов.

[57] См., например [22].

[58] Это псевдо-оптимальное решение было выбрано с помощью действий, описанных выше. Стоит ещё отме-

Отметим, что на практике описанный процесс построения ППЗ *не занимает много лишнего времени* и *не приводит к большому увеличению списка задач*, предназначенных для потенциального решения в будущем.

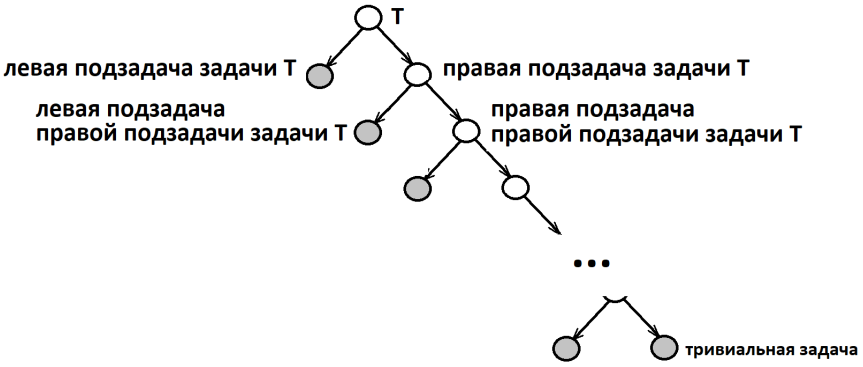

Рис. 8. Последовательность правых задач (серым цветом показаны подзадачи, включающиеся в список подзадач для дальнейшего решения)

Итак, мы описали простой процесс построения anytime-алгоритма на основе некоторого конкретного варианта МВГ. В английской литературе подобные алгоритмы считаются вариантами алгоритма, называемого *truncated* branch-and-bound method; мы о нём уже немного говорили в предыдущем разделе.

7. ПРИМЕР РАБОТЫ АЛГОРИТМА С ИСПОЛЬЗОВАНИЕМ ПОСЛЕДОВАТЕЛЬНОСТИ ПРАВЫХ ЗАДАЧ

В этом разделе мы рассмотрим другой порядок перебора подзадач — он следует из приведённого в предыдущем разделе алгоритма построения ППЗ. Но, чтобы было с чем сравнивать, мы будем использовать тот же самый пример исходной матрицы (рис. 2). Ещё раз отметим, что этот пример интересен вот чем: оптимальное решение в нём находится *не там, где ожидалось вначале* (то есть не в том множестве туров, которое вначале рассматривается как «более перспективное»).

Для удобства (и для того чтобы не рисовать много рисунков) мы будем далее обозначать возникающие (под)задачи следующим образом:

— «база»: исходную задачу будем обозначать ε[59];
— «шаг»: если некоторая задача обозначена X, то возникающие на её основе левая и правая подзадачи обозначаются соответственно $X0$ и $X1$[60].

Отметим, что эти обозначения нестандартные.

Приведём ещё несколько замечаний:

тить, что в этом вопросе не всё так очевидно. Иногда процесс построения ППЗ может быть прерван ранее — для получения оптимального решения некоторой промежуточной задачи, например, методом перебора (при небольшой размерности задачи) или каким-либо иным методом. Этот вопрос связан с подробными исследованиями МВГ.

[59] Это обычное обозначение пустого слова в теории формальных языков, и приписывание к нему любого слова u (по-научному называется «конкатенация») даёт в результате это же слово u.

[60] В частности, задачи ППЗ для исходной задачи обозначаются 1, 11, 111, … — до тех пор, пока не получится тривиальная задача.

- В описанной далее реализации алгоритма договорённости о других обозначениях и вспомогательных алгоритмах совпадают с применёнными ранее; например, редукцию мы будем всегда производить сначала по строкам, а потом по столбцам.
- Множество имеющихся в некоторый момент работы алгоритма (под)задач будем указывать очень кратко (мы предполагаем вернуться к этому вопросу в статье-продолжении).
- Также мы не будем рисовать текущие состояния дерева вывода: мы считаем, что приведённого на рис. 6 достаточно.
- Кроме того, редукцию будем всегда выполнять *сразу после формирования очередной (под)задачи* (в [2] «чёткого указания» по этому поводу нет); соответственно, на рисунках задачи будем обычно приводить парами — непосредственно полученную и редуцированную версии.
- Ещё сто́ит отметить следуюшее. Всё это — всё-таки *неполная* информация о подзадаче: не хватает *уже выбранного* множества дуг, а восстановить это множество по имеющейся (описанной здесь) информации, вообще говоря, невозможно. Однако получить *значение* оптимального тура (но не сам этот тур) можно и по имеющейся информации (перечисленной здесь). К *полному* представлению данных (необходимому для полного же решения ЗКВ) мы предполагаем вернуться в следующей статье.
- Как и ранее, вместо «подзадача» будем иногда говорить «задача» (смысл всегда будет понятен из контекста).

Итак . . .

Задача ε: в немного ином формате (согласованном с приведёнными замечаниями) повторим исходные данные.

	1	2	3	4	5	
1	999	25	40	31	27	
2	5	999	17	30	25	
3	19	15	999	6	1	
4	9	50	24	999	6	
5	22	8	7	10	999	
						0

	1	2	3	4	5	
1	999	0	15	3	2	25
2	0	999	12	22	20	5
3	18	14	999	2	0	1
4	3	44	18	999	0	6
5	15	1	0	0	999	7
				3		**0+47=47**

Рис. 9. Исходные данные, задача ε

Следующий шаг, понятно, совпадает с обсуждённым ранее: у нас пока только одна подзадача, и ранжировать нули надо для неё. Получаем такие значения badness, совпадающие с вычисленными в [2][61]; обозначаем эти значения b_{ij}, индексы совпадают с индексами нулей:

$$b_{12} = 2+1 = 3\,, \qquad b_{21} = 12+3 = 15\,, \qquad b_{35} = 2+0 = 2\,,$$
$$b_{45} = 3+0 = 3\,, \qquad b_{53} = 0+12 = 12\,, \qquad b_{54} = 0+2 = 2\,.$$

Таким образом, лучший нуль соответствует элементу матрицы с координатами $(2,1)$, и на основе этого элемента мы формируем первое ветвление МВГ; после него получаем следующие задачи 0 и 1.

	1	2	3	4	5	
1	999	0	15	3	2	
2	999	999	12	22	20	
3	18	14	999	2	0	
4	3	44	18	999	0	
5	15	1	0	0	999	
						47

	1	2	3	4	5	
1	999	0	15	3	2	
2	999	999	0	10	8	12
3	15	14	999	2	0	
4	0	44	18	999	0	
5	12	1	0	0	999	
	3					**47+15=62**

Рис. 10. Задача 0

[61] Начиная со следующего шага, совпадения работы алгоритма с [2] не будет.

Как мы видим, для задачи 1 мы можем привести только одну матрицу: для неё редукция не требуется, поскольку в каждой строке и в каждом столбце уже есть нуль. (Однако «большой цикл» при этом «пока» не получается; проще всего это объясняется тем, что в первой строке только один элемент равен 0, и этот элемент вместе с выбранным ранее образовывал бы «малый цикл».)

	2	3	4	5	
1	0	15	3	2	
3	14	999	2	0	
4	44	18	999	0	
5	1	0	0	999	
					47

Рис. 11. Задача 1

Начиная с этого момента, выполнение алгоритма МВГ идёт «по новому пути», отличающемуся от [2]: ведь мы, независимо ни от чего, делаем ветвление задачи 1[62] — это начало построения ППЗ.

Продолжим это построение. Для формирования задач 10 и 11 (входящих, как следует из рис. 8, в ППЗ) мы должны проранжировать нули задачи 1. Будем употреблять для всех матриц обозначения badness b_{ij} (аналогично обозначениям, применённым выше); при этом обратим внимание, что индексы i и j — это номера строк и столбцов *исходной* матрицы, приведённые на рисунках слева и сверху от матриц текущих. Таким образом[63],

$$b_{12} = 2+1 = 3\,, \qquad b_{35} = 2+0 = 2\,,$$
$$b_{45} = 18+0 = 18\,, \qquad b_{53} = 0+15 = 15\,, \qquad b_{54} = 0+2 = 2\,,$$

и мы для ветвления выбираем элемент с индексами $(4,5)$. Обе получающиеся задачи 10 и 11 приведены на следующем рис. 12:

	2	3	4	5	
1	0	15	3	2	
3	14	999	2	0	
4	44	18	999	999	
5	1	0	0	999	
					47

	2	3	4	5	
1	0	15	3	2	
3	14	999	2	0	
4	26	0	999	999	18
5	1	0	0	999	
					47+18=65

	2	3	4	
1	0	15	3	
3	14	999	2	
5	1	0	0	
				47

	2	3	4	
1	0	15	3	
3	12	999	0	2
5	1	0	0	
				47+2=49

Рис. 12. Задача 10 (две матрицы слева) и задача 11 (две матрицы справа)

Продолжим построение ППЗ[64]. Теперь необходимо проранжировать нули в задаче 11, и мы получаем следующее:

$$b_{12} = 3+1 = 4\,, \qquad b_{34} = 12+0 = 12\,,$$
$$b_{53} = 0+15 = 15\,, \qquad b_{54} = 0+0 = 0\,.$$

[62] Впрочем, и «идя по старому пути», мы бы на этом шаге производили бы ветвление задачи 1. Разница в том, что для этого пришлось бы сначала сравнивать границы. В общем случае мы сравниваем границы всех имеющихся «ещё не разветвлённых» подзадач.

[63] Заранее отметим следующее. И здесь, и в других подобных ситуациях большинство различных badness сохраняются, то есть они равны значениям badness с теми же индексами, посчитанными для матриц, относящихся к вершинам-родителям дерева поиска. «Многие, но не все» — и поэтому возникает следующая проблема: нужно применить как можно больше информации, относящейся к подзадаче вершины-родителя. Это похоже на некоторые из проблем, решавшихся нами в [17, 23], но к настоящему моменту до конца не доведенных.

[64] И ещё раз отметим, что мы не следим за появлением «малых циклов» — точнее, будем это проверять «вручную» при получении каждого очередного тура. Как это делать не «вручную», а автоматически, рассмотрим в статье-продолжении.

Для ветвления выбираем координаты (5,3); на рис. 13 приведены получающиеся в результате этого ветвления задачи 110 и 111. В задаче 111 дальнейшие ветвления бессмысленны[65]. Оба нуля матрицы этой задачи образуют «малые циклы», поэтому текущее псевдооптимальное решение образуется её ненулевыми элементами — вместе с нулями, уже выбранными ранее в процессе построения ППЗ[66]. Его стоимость равна 64 (см. рис. 13), а сам цикл таков:

	2	3	4	
1	0	15	3	
3	12	999	0	
5	1	999	0	
				49

	2	3	4	
1	0	15	3	
3	12	999	0	
5	1	999	0	
		15		49+15=64

	2	4	
1	0	3	
3	12	0	
			49

Рис. 13. Задача 110 (две матрицы слева) и задача 111

$$1 \to 4 \to 5 \to 3 \to 2 \to 1\,.$$

Оставшиеся непросчитанными подзадачи — 0, 10 и 110. Среди них выбираем задачу 0 как имеющую наименьшую границу 62 (в задачах 10 и 110 — 65 и 64)[67]. Далее «неинтересно»: эта граница *соответствует ответу* — поскольку существует последовательность нулей, образующая «большой цикл»:

$$1 \to 2 \to 3 \to 5 \to 4 \to 1\,.$$

Стоимость этого цикла меньше всех имеющихся границ, что даёт возможность *дальнейшие вычисления прекратить*.

8. ЗАКЛЮЧЕНИЕ

Как мы уже не раз отмечали в тексте статьи, мы предполагаем в ближайшем будущем опубликовать статью о возможной *программной реализации* представленного здесь материала (и, можно сказать, об *объектно-ориентированной* реализации). Заранее приведём несколько замечаний об этой реализации, некоторые из этих замечаний уже были в тексте выше, в первую очередь, в сносках. Итак (от более важных к менее важным).

- По-видимому, основное в этой реализации то, что дерево поиска мы «держим в голове», а в памяти храним не его, а *текущий список подзадач*.
- Точнее: в упрощённой реализации (которую мы, по-видимому, и приведём) будет не список подзадач, а массив, и не подзадач, а указателей на таковые.
- Вероятно, класс «подзадача» — *самый важный*; но, конечно, будет рассмотрен и *самый верхний* (по иерархии) класс — задача. Самый простой вариант описаний этого класса — нечто, содержащее несколько полей, одним из которых, конечно, является

[65] А в реальных программах мы перестаём применять ветвление для размерности 6: считаем ответ для получившейся матрицы переборным алгоритмом.

[66] Понятно, что и в реальных условиях подобные текущие псевдооптимальные решения получаются «быстрее, чем за 2 месяца 28 дней».

[67] По-видимому, ещё более интересной была *бы* ситуация, в котором наименьшую границу имела бы задача 10. Для неё нам снова пришлось *бы* провести процесс построения ППЗ. Подсчёт badness (для матрицы задачи 10) был *бы* таким:

$$b_{12} = 2+1 = 3\,,\quad b_{35} = 2+2 = 4\,,\quad b_{43} = 26+0 = 26\,,\quad b_{53} = 0+0 = 0\,,\quad b_{54} = 0+2 = 2\,,$$

после чего для ветвления мы выбрали *бы* элемент (4,3). И так далее…

Подобные ситуации постоянно возникают в реальных программах решения ЗКВ с помощью МВГ, и, например, рассматривая случайную ЗКВ размерности около 75, мы при реальном выполнении программ в процессе многих ветвлений обычно получаем до 200 000 подзадач.

список подзадач (ну, или массив указателей на них, см. выше). Однако авторам ещё больше нравится другое представление: задача — *наследник* списка подзадач! Это представление «противоречит принципам объектно-ориентированного программирования»? Видимо — да (ведь множество задач не является подмножеством множества списков), но тем же «принципам структурного программирования» Н. Вирта ([21] и мн. др.) тоже нельзя следовать бездумно, `goto` иногда очень полезно!

- Конечно, мы подробно рассмотрим полную информацию о само́й подзадаче (выше мы отмечали, что пока эту полную информацию ещё не привели).
- Мы рассмотрим, как «следить» за тем, чтобы при работе МВГ не получались «малые циклы», примеры которых были выше.
- Мы подробно рассмотрим и полную информацию о представлении (хранении) номеров строк и столбцов; отметим, что это представление, конечно, простое, но всё же не столь тривиальное, как кажется на первый взгляд.

Список литературы

1. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982. 416 с.
2. *Гудман С., Хидетниеми С.* Введение в разработку и анализ алгоритмов. М.: Мир, 1981. 364 с.
3. *Громкович Ю.* Теоретическая информатика. Введение в теорию автоматов, теорию вычислимости, теорию сложности, теорию алгоритмов, рандомизацию, теорию связи и криптографию. СПб.: БХВ-Петербург, 2010. 336 с.
4. *Hromkovič J.* Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. Berlin: Springer, 2004. 547 p.
5. *Gera R., Hedetniemi S., Larson C.* (Eds.) Graph Theory. Favorite Conjectures and Open Problems — 1. Berlin: Springer, 2016. 291 p.
6. *Gera R., Hedetniemi S., Larson C.* (Eds.) Graph Theory. Favorite Conjectures and Open Problems — 2. Berlin: Springer, 2018. 281 p.
7. *Мельников Б. Ф., Мельникова Е. А., Пивнева С. В.* Пасьянс «Махджонг»: научный проект для старшеклассников с элементами искусственного интеллекта // Компьютерные инструменты в образовании. 2018. № 5. С. 41–51.
8. 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.
9. *Applegate D., Bixby R., Chvatal V., Cook W.* The Traveling Salesman Problem: A Computational Study. NJ: Princeton University Press, 2007. 593 p.
10. *Gutin G., Punnen A.* (Eds.) Combinatorial Optimization. The Traveling Salesman Problem and Its Variations. Berlin: Springer, 2007. 830 p.
11. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы. Построение и анализ. М.: Вильямс, 2013. 1324 с.
12. *Ляликов В. Н.* Комплекс программ для исследования методов решения задачи о коммивояжере. Диссертация … кандидата технических наук. Специальность 05.13.18 — математическое моделирование, численные методы и комплексы программ. Ульяновск, 2008. 123 с.
13. *Dorigo M., Gambardella L. M.* Ant Colonies for the Traveling Salesman Problem // Biosystems. 1997. Vol. 43, № 2. P. 73–81.
14. *Булынин А. Г., Мельников Б. Ф., Мещанин В. Ю., Терентьева Ю. Ю.* Оптимизационные задачи, возникающие при проектировании сетей связи высокой размерности, и некоторые эвристические методы их решения // Информатизация и связь. 2020. № 1. С. 34–40.
15. *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 (4). P. 042093.
16. *Мельников Б. Ф., Тренина М. А.* Применение метода ветвей и границ в задаче восстановле-

ния матрицы расстояний между цепочками ДНК // International Journal of Open Information Technologies. 2018. Vol. 6, № 8. С. 1–13.
17. *Melnikov B., Trenina M., Melnikova E.* Different Approaches to Solving the Problem of Reconstructing the Distance Matrix Between DNA Chains // Communications in Computer and Information Science. 1201 CCIS. 2020. P. 211–223.
18. *Мельников Б. Ф., Давыдова Е. В., Ничипорчук Н. В., Тренина М .А.* Кластеризация ситуаций в алгоритмах решения задачи коммивояжера и ее применение в некоторых прикладных задачах. Часть II. Списочная метрика и некоторые связанные оптимизационные проблемы // Известия высших учебных заведений. Поволжский регион. Физико-математические науки. 2018. № 4 (48). С. 62–77.
19. *Мельников Б., Радионов А.* О выборе стратегии в недетерминированных антагонистических играх // Программирование. 1998. № 5. С. 55–63.
20. *Мельников Б., Романов Н.* Еще раз об эвристиках для задачи коммивояжера // Теоретические проблемы информатики и ее приложений. 2001. № 4. С. 81–89.
21. *Мельников Б. Ф., Пивнева С. В.* Принятие решений в прикладных задачах с применением динамически подобных функций риска // Вестник транспорта Поволжья. 2010. № 3 (23). С. 28а–33.
22. *Вирт Н.* Алгоритмы и структуры данных. М.: Мир, 1989. 360 с.
23. *Сигал И. Х., Иванова А. П.* Введение в прикладное дискретное программирование: модели и вычислительные алгоритмы. М.: Физматлит, 2007. 304 с.
24. *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.

Поступила в редакцию 16.01.2021, окончательный вариант — 18.02.2021.

Мельников Борис Феликсович, доктор физико-математических наук, профессор Совместного университета МГУ — ППИ в Шэньчжэне, Китай, ✉ bf-melnikov@yandex.ru

Мельникова Елена Анатольевна, кандидат физико-математических наук, доцент Российского государственного социального университета, ✉ ya.e.melnikova@yandex.ru

Computer tools in education, 2021
№ 1: 21–44
http://cte.eltech.ru
doi:10.32603/2071-2340-2021-1-21-45

On the Classical Version of the Branch and Bound Method

Melnikov B. F.[1], PhD, Professor, ✉ bf-melnikov@yandex.ru
Melnikova E. A.[2], PhD, Associated Professor, ✉ ya.e.melnikova@yandex.ru

[1]Shenzhen MSU – BIT University, No. 1, International University Park Road, Dayun New Town, Longgang District, Shenzhen, PRC, 517182, Guangdong Province, Shenzhen, China
[2]Russian State Social University, 4, build. 1, Wilhelm Pieck street, 129226 Moscow, Russia

Abstract

In the computer literature, a lot of problems are described that can be called discrete optimization problems: from encrypting information on the Internet (including creating programs for digital cryptocurrencies) before searching for “interests” groups in social

networks. Often, these problems are very difficult to solve on a computer, hence they are called “intractable”. More precisely, the possible approaches to quickly solving these problems are difficult to solve (to describe algorithms, to program); the brute force solution, as a rule, is programmed simply, but the corresponding program works much slower.

Almost every one of these intractable problems can be called a mathematical model. At the same time, both the model itself and the algorithms designed to solve it are often created for one subject area, but they can also be used in many other areas. An example of such a model is the traveling salesman problem. The peculiarity of the problem is that, given the relative simplicity of its formulation, finding the optimal solution (the optimal route). This problem is very difficult and belongs to the so-called class of NP-complete problems. Moreover, according to the existing classification, the traveling salesman problem is an example of an optimization problem that is an example of the most complex subclass of this class.

However, the main subject of the paper is not the problem, but the method of its solution, i.e. the branch and bound method. It consists of several related heuristics, and in the monographs, such a multi-heuristic branch and bound method was apparently not previously noted: the developers of algorithms and programs should have understood this themselves. At the same time, the method itself can be applied (with minor changes) to many other discrete optimization problems.

So, the classical version of branch and bound method is the main subject of this paper, but also important is the second subject, i.e. the traveling salesman problem, also in the classical formulation. The paper deals with the application of the branch and bound method in solving the traveling salesman problem, and about this application, we can also use the word “classical”. However, in addition to the classic version of this implementation, we consider some new heuristics, related to the need to develop real-time algorithms.

Keywords: *optimization problems, traveling salesman problem, heuristic algorithms, branch and bound method, real-time algorithms.*

Citation: B. F. Melnikov and E. A. Melnikova, “On the Classical Version of the Branch and Bound Method,” *Computer tools in education*, no. 1, pp. 21–44, 2021 (in Russian); doi: 10.32603/2071-2340-2021-1-21-45

References

1. M. R. Garey and D. S. Johnson, *Computers and Intractability*, Moscow: Mir, 1982 (in Russian).
2. S. E. Goodman and S. T. Hedetniemi, *Introduction to the Design and Analysis of Algorithms*, Moscow: Mir, 1981 (in Russian).
3. J. Hromkovič, *Theoretical computer science: introduction to Automata, computability, complexity, algorithmics, randomization, communication, and cryptography*, St. Petersburg, Russia: BKhV-Peterburg, 2010 (in Russian).
4. J. Hromkovič, *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*, Berlin: Springer, 2004.
5. R. Gera, S. Hedetniemi, and C. Larson, eds., *Graph Theory. Favorite Conjectures and Open Problems*, vol. 1, Berlin: Springer, 2016.
6. R. Gera, S. Hedetniemi, and C. Larson, eds., *Graph Theory. Favorite Conjectures and Open Problems*, vol. 2, Berlin: Springer, 2016.
7. B. F. Melnikov, E. A. Melnikova, and S. V. Pivneva, “Mahjongg Solitaire: a High School Student Scientific Project, Containing Elements of Artificial Intelligence,” *Computer tools in education*, no. 5, pp. 41–51, 2018 (in Russian); doi: 10.32603/2071-2340-2018-5-41-51
8. M. E. Abramyan, B. F. Melnikov, and E. A. Melnikova, “Finite Automata Table: a Science Project for High School Students,” *Computer tools in education*, no. 2, pp. 87–107, 2019 (in Russian); doi:10.32603/2071-2340-2019-2-87-107

9. D. Applegate, R. Bixby, V. Chvatal, and W. Cook, *The Traveling Salesman Problem: A Computational Study*, NJ: Princeton University Press, 2007.
10. G. Gutin and A. Punnen, eds. *Combinatorial Optimization. The Traveling Salesman Problem and Its Variations*, Berlin: Springer, 2007.
11. Th. H. Cormen, Ch. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, Moscow: Viljams, 2013 (in Russian).
12. V. N. Lyalikov, "Kompleks programm dlya issledovaniya metodov resheniya zadachi o kommivoyazhere. Dissertation of the candidate of technical sciences,", Ulyanovsk State Technical University, Ul'yanovsk, Russia, 2008 (in Russian).
13. M. Dorigo and L. M. Gambardella, "Ant Colonies for the Traveling Salesman Problem," *Biosystems*, vol. 43, no. 2, pp. 73–81, 1997.
14. G. Bulynin, B. F. Melnikov, V. Y. Meshchanin, and Y. Y. Terentyeva, "Optimization problem, arising in the development of high-dimensional communication networks, and some heuristic methods for solving them," *Informatizatsiya i svyaz'*, no. 1, pp. 34–40, 2020 (in Russian); doi: 10.34219/2078-8320-2020-11-1-34-40
15. B. F. Melnikov, V. Y. Meshchanin, and Y. Y. Terentyeva, "Implementation of optimality criteria in the design of communication networks," *Journal of Physics: Conference Series*, vol. 1515, no. 4, p. 042093, 2020; doi: 10.1088/1742-6596/1515/4/042093
16. B. F. Melnikov and M. A. Trenina, "The application of the branch and bound method in the problem of reconstructing the matrix of distances between DNA strings," *International Journal of Open Information Technologies*, vol. 6, no. 8, pp. 1–13, 2018 (in Russian).
bibitemme-tre2 B. Melnikov, M. Trenina, and E. Melnikova, "Different Approaches to Solving the Problem of Reconstructing the Distance Matrix Between DNA Chains," in *Proc. SITITO 2018: Modern Information Technology and IT Education*, vol. 1201, 2020, pp. 211–223; doi: 10.1007/978-3-030-46895-8_17
17. B. F. Melnikov, E. V. Davydova, A. V. Nichiporchuk, and M. A. Trenina, "Clustering of situations in the algorithms for solving the traveling salesman problem and its application in some applied tasks. Part II. List metrics and some related optimization problems," *University proceedings. Volga region*, vol. 4, no. 48, pp. 62–77, 2018 (in Russian).
18. B. F. Melnikov and A. N. Radionov, "O vybore strategii v nedeterminirovannykh antagonisticheskikh igrakh" [On the choice of strategy in nondeterministic antagonistic games], *Izvestiya RAN. Programmirovanie*, no. 5, pp. 55–62, 1998 (in Russian).
19. B. Melnikov and N. Romanov, "Eshche raz ob evristikakh dlya zadachi kommivoyazhera" [Once again about heuristics for the traveling salesman problem], *Teoreticheskie problemy informatiki i ee prilozhenii*, vol. 4, pp. 81–89, 2001 (in Russian).
20. B. F. Melnikov and S. V. Pivneva, "Prinyatie reshenii v prikladnykh zadachakh s primeneniem dinamicheski podobnykh funktsii riska" [Decision making in applied problems using dynamically similar risk functions], *Vestnik transporta Povolzhya*, vol. 3, no. 23, pp. 28a–33, 2010 (in Russian).
21. N. Wirth, *Algorithms + Data Structures*, Moscow: Mir, 1989 (in Russian).
22. Kh. Sigal and A. P. Ivanova, *Introduction to Applied Discrete Programming: Models and Computational Algorithms*, Moscow: Fizmatlit, 2007 (in Russian).
23. B. Melnikov and E. Melnikova, "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*, no. 2, pp. 16–20, 2007 (in Russian).

Received 16-01-2021, the final version — 18-02-2021.

Boris Melnikov, PhD, Professor, Shenzhen MSU — BIT University, Shenzhen, China, ✉ bf-melnikov@yandex.ru

Elena Melnikova, PhD, Associated Professor, Russian State Social University, Moscow, Russia, ✉ ya.e.melnikova@yandex.ru