MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 27, NO. 1, 2020
journal homepage: www.mais-journal.ru

ALGORITHMS

Branch and Bound Algorithm for the Traveling Salesman Problem is not a Direct Type Algorithm

A. N. Maksimenko[1]

DOI: 10.18255/1818-1015-2020-1-72-85

[1]P. G. Demidov Yaroslavl State University, 14 Sovetskaya, Yaroslavl 150003, Russia.

MSC2020: 90C57
Research article
Full text in Russian

Received December 3, 2019
After revision January 5, 2020
Accepted February 28, 2020

In this paper, we consider the notion of a direct type algorithm introduced by V. A. Bondarenko in 1983. A direct type algorithm is a linear decision tree with some special properties. The concept of a direct type algorithm is determined using the graph of solutions of a combinatorial optimization problem. The vertices of this graph are all feasible solutions of a problem. Two solutions are called adjacent if there are input data for which these and only these solutions are optimal. A key feature of direct type algorithms is that their complexity is bounded from below by the clique number of the solutions graph. In 2015-2018, there were five papers published, the main results of which are estimates of the clique numbers of polyhedron graphs associated with various combinatorial optimization problems. The main motivation in these works is the thesis that the class of direct type algorithms is wide and includes many classical combinatorial algorithms, including the branch and bound algorithm for the traveling salesman problem, proposed by J. D. C. Little, K. G. Murty, D. W. Sweeney, C. Karel in 1963. We show that this algorithm is not a direct type algorithm. Earlier, in 2014, the author of this paper showed that the Hungarian algorithm for the assignment problem is not a direct type algorithm. Thus, the class of direct type algorithms is not so wide as previously assumed.

Keywords: branch and bound; traveling salesman problem; linear decision tree; clique number; direct type algorithm

INFORMATION ABOUT THE AUTHORS

Aleksandr N. Maksimenko | orcid.org/0000-0002-0887-1500. E-mail: maximenko.a.n@gmail.com
PhD.

For citation: A. N. Maksimenko, “Branch and Bound Algorithm for the Traveling Salesman Problem is not a Direct Type Algorithm”, *Modeling and analysis of information systems*, vol. 27, no. 1, pp. 72-85, 2020.

© Maksimenko A. N., 2020
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 27, № 1, 2020
сайт журнала: www.mais-journal.ru

ALGORITHMS

Алгоритм ветвей и границ для задачи коммивояжера не является алгоритмом прямого типа

А. Н. Максименко[1]

DOI: 10.18255/1818-1015-2020-1-72-85

[1]Ярославский государственный университет им. П. Г. Демидова, ул. Советская, 14, Ярославль, 150003, Россия.

УДК 519.16
Научная статья
Полный текст на русском языке

Получена 3 декабря 2019 г.
После доработки 5 января 2020 г.
Принята к публикации 28 февраля 2020 г.

В настоящей работе рассматривается понятие линейного разделяющего алгоритма прямого типа, введенное В. А. Бондаренко в 1983 г. Понятие алгоритма прямого типа определяется с помощью графа решений задачи комбинаторной оптимизации. Вершинами этого графа служат все допустимые решения задачи. Два решения называются смежными, если существуют входные данные, для которых эти решения и только они являются оптимальными. Ключевой особенностью алгоритмов прямого типа является то, что их трудоемкость оценивается снизу кликовым числом графа решений. В 2015–2018 гг. было опубликовано пять работ, основными результатами которых являются оценки кликовых чисел графов многогранников, ассоциированных с различными задачами комбинаторной оптимизации. В качестве основной мотивации в этих работах приводится тезис о том, что класс алгоритмов прямого типа является широким и включает в себя многие классические комбинаторные алгоритмы, в том числе алгоритм ветвей и границ для задачи коммивояжера, предложенный J. D. C. Little, K. G. Murty, D. W. Sweeney, C. Karel в 1963 г. Мы покажем, что этот алгоритм не является алгоритмом прямого типа. Ранее, в 2014 г., автором настоящей работы было показано, что венгерский алгоритм для задачи о назначениях не является алгоритмом прямого типа. Таким образом, класс алгоритмов прямого типа не является настолько широким, как предполагалось ранее.

Ключевые слова: метод ветвей и границ; задача коммивояжера; линейное разделяющее дерево; кликовое число; алгоритм прямого типа

ИНФОРМАЦИЯ ОБ АВТОРАХ

Александр Николаевич Максименко	orcid.org/0000-0002-0887-1500. E-mail: maximenko.a.n@gmail.com канд. физ.-мат. наук, доцент.

Для цитирования: A. N. Maksimenko, “Branch and Bound Algorithm for the Traveling Salesman Problem is not a Direct Type Algorithm”, *Modeling and analysis of information systems*, vol. 27, no. 1, pp. 72-85, 2020.

© Максименко А. Н., 2020
Эта статья открытого доступа под лицензией CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Введение

В 2015–2018 гг. было опубликовано несколько работ [1—5], основными результатами которых являются оценки кликовых чисел графов многогранников, ассоциированных с различными задачами комбинаторной оптимизации. Основной мотивацией для таких оценок является следующий тезис: “It is known that this value characterizes the time complexity in a broad class of algorithms based on linear comparisons”[1] [5]. А именно, речь идет о классе алгоритмов прямого типа, впервые введенном в [6]. В качестве подтверждения этого тезиса в [2, 3] говорится о том, что этот класс включает алгоритмы сортировки, жадный алгоритм, динамическое программирование и метод ветвей и границ[2]. Доказательства того, что эти алгоритмы (а также алгоритм Эдмондса для задачи о паросочетаниях) являются алгоритмами прямого типа, впервые были опубликованы в диссертации [7] (см. также монографию [8]). В 2014 г. в [9] было показано, что алгоритм Куна—Манкреса для задачи о назначениях (а вместе с ним и алгоритм Эдмондса) не принадлежит к этому классу. Там же был описан часто используемый на практике способ модификации алгоритмов, выводящий их из класса алгоритмов прямого типа. Ниже мы докажем, что классический алгоритм ветвей и границ для задачи коммивояжера [10, 11] тоже не принадлежит к этому классу. Тем самым будет показано, что теорема 2.6.3 из диссертации [7] (теорема 3.6.6 из монографии [8]) не может быть доказана в оригинальной постановке. Это позволяет сделать вывод о том, что класс алгоритмов прямого типа не является столь широким, как предполагалось ранее.

Текст статьи организован следующим образом. В разделе 1 приводится псевдокод классического алгоритма ветвей и границ для задачи коммивояжера. В разделе 2 вводятся основные понятия концепции алгоритмов прямого типа и два ключевых определения: алгоритма прямого типа и алгоритма «прямого типа». В разделе 3 показано, что классический алгоритм ветвей и границ для задачи коммивояжера не является алгоритмом прямого типа, а в разделе 4 — что он не является алгоритмом «прямого типа».

1. Алгоритм ветвей и границ для задачи коммивояжера

Рассмотрим полный орграф $G = (V, A)$ с множеством вершин $V = [n] = \{1, 2, \dots, n\}$ и дуг $A = \{(i, j) \mid i, j \in V,\ i \neq j\}$. Каждой дуге $(i, j) \in A$ поставлено в соответствие число $c_{ij} \in \mathbb{Z}$, называемое *длиной дуги. Длиной подмножества* $H \subseteq A$ будем называть суммарную длину входящих в него дуг: $\mathrm{len}(H) = \sum_{(i,j)\in H} c_{ij}$. Задача коммивояжера состоит в том, чтобы найти $H^* \subseteq A$, являющееся гамильтоновым контуром в G и имеющее минимальную длину $\mathrm{len}(H^*)$.

Для удобства дальнейшего обсуждения поместим числа c_{ij} в матрицу $C = (c_{ij})$. Диагональным элементам c_{ii} припишем максимально возможные длины, $c_{ii} := \infty$, чтобы исключить их влияние на работу алгоритма, и будем предполагать, что $\infty - b = \infty$ для любого числа $b \in \mathbb{Z}$. Через $\mathrm{I}(M)$ будем обозначать множество индексов строк матрицы M, а через $\mathrm{J}(M)$ обозначим множество индексов столбцов матрицы M. В начале работы алгоритма $\mathrm{I}(C) = \mathrm{J}(C) = V$. Через $M(S, T)$ обозначим подматрицу матрицы M, лежащую на пересечении строк $S \subseteq \mathrm{I}(M)$ и столбцов $T \subseteq \mathrm{J}(M)$.

Сам алгоритм подробно описан в [11, раздел 4.1.6] и [10]. Мы приводим лишь его псевдокод — алгоритм 1. Отдельно, в алгоритме 2 описан процесс редуцирования строк и столбцов матрицы, а в алгоритме 3 — способ выбора такого нулевого элемента матрицы, при замене которого на бесконечность сумма редукций матрицы максимальна.

[1] «Известно, что эта величина характеризует сложность по времени в широком классе алгоритмов, основанных на линейных сравнениях»

[2] Но ссылки на источник с соответствующими доказательствами не приводятся.

Алгоритм 1. Метод ветвей и границ для задачи коммивояжера

Глобальные: гамильтонов контур Hopt с минимальной длиной; его длина lopt. До начала работы алгоритма lopt := ∞.

Вход : матрица длин M; множество дуг Arcs, обязательных для включения в контур; текущая сумма всех редукций sum. В самом начале работы алгоритма M := C, Arcs := $\emptyset$, sum := 0.

```
Procedure BranchBound(M, Arcs, sum)
    /* Редуцируем матрицу M                                          */
    Reduction(M, sum)
    if sum ≥ lopt then
        завершить текущий экземпляр процедуры
    /* Выбираем оптимальный нулевой элемент матрицы M                */
    (i, j) := ChooseArc(M)
    /* Разбираем случаи, когда контур содержит дугу (i, j)           */
    if |I| = 3 then
        /* Находим единственный гамильтонов контур                   */
        H := HamiltonCycle(Arcs ∪ {(i, j)})
        if len(H) < lopt then
            Hopt := H
            lopt := len(H)
    else
        /* Вычеркиваем i-ю строку и j-й столбец                      */
        Mnew := M(I(M) \ {i}, J(M) \ {j})
        /* Находим запрещенную дугу                                  */
        (l, k) := ForbiddenArc(Arcs,(i,j))
        Mnew[l,k] := ∞
        BranchBound(Mnew, Arcs ∪ {(i, j)}, sum)
    /* Разбираем случаи, когда контур не содержит дугу (i, j)        */
    M[i,j] := ∞
    BranchBound(M, Arcs, sum)

Function HamiltonCycle(Arcs)
    Найти гамильтонов контур, содержащий все дуги из Arcs.

Function ForbiddenArc(Arcs,(i, j))
    Найти пару вершин l и k, являющихся концом и началом наибольшего (по включению)
      пути в Arcs, содержащего (i, j).
```

Алгоритм 2. Редуцирование строк и столбцов матрицы

Вход : матрица M; текущая сумма всех редукций sum.
Выход : редуцированная матрица M; измененная sum.

```
Procedure Reduction(M, sum)
    /* Редуцируем строки матрицы M                                  */
    for i ∈ I(M) do
        m := ∞
        /* Находим m = m(i) = min_{j∈J(M)} M[i,j]                   */
        for j ∈ J(M) do
            if m > M[i,j] then m := M[i,j]
        sum := sum + m
        for j ∈ J(M) do M[i,j] := M[i,j] − m
    /* Редуцируем столбцы матрицы M                                 */
    for j ∈ J(M) do
        m := ∞
        for i ∈ I(M) do
            if m > M[i,j] then m := M[i,j]
        sum := sum + m
        for i ∈ I(M) do M[i,j] := M[i,j] − m
```

Алгоритм 3. Выбор дуги

Вход : матрица M.
Выход : дуга (i^*, j^*), при запрещении которой нижняя оценка длины гамильтонова контура максимальна.

```
Function ChooseArc(M)
    w := −1
    for i ∈ I(M) do
        for j ∈ J(M) do
            if M[i,j] = 0 then
                m := ∞
                /* Находим m = min_t M[i,t]                         */
                for t ∈ J(M) \ {j} do
                    if m > M[i,t] then m := M[i,t]
                k := ∞
                /* Находим k = min_t M[t,j]                         */
                for t ∈ I(M) \ {i} do
                    if k > M[t,j] then k := M[t,j]
                /* Сравниваем m + k с текущим рекордом w            */
                if m + k > w then
                    w := m + k
                    (i*, j*) := (i, j)
```

2. Алгоритмы прямого типа

При изложении основ теории алгоритмов прямого типа мы будем придерживаться [7] (см. также [8]).

С целью унификации изложения матрица длин дуг C далее будет называться *вектором*[3] *входных данных* или просто *входом*. Решение задачи коммивояжера, т.е. гамильтонов контур $H \subseteq A$, будет представляться в виде 0/1-вектора $\boldsymbol{x} = (x_{ij})$, имеющего ту же размерность, что и C. Координаты этого вектора $x_{ij} = 1$, при $(i, j) \in H$, и $x_{ij} = 0$ иначе. Через X обозначаем множество всех 0/1-векторов $\boldsymbol{x}$, соответствующих гамильтоновым контурам в рассматриваемом орграфе G. Таким образом, при фиксированном входе C задача коммивояжера состоит в поиске решения $\boldsymbol{x}^* \in X$ такого, что $\langle \boldsymbol{x}^*, C\rangle \leq \langle \boldsymbol{x}, C\rangle \, \forall \boldsymbol{x} \in X$. Далее будем называть такое решение $\boldsymbol{x}^*$ *оптимальным относительно входа* C. Следуя [7, определение 1.1.2], совокупность всех таких оптимизационных задач, образованную фиксированным множеством допустимых решений X (в случае задачи коммивояжера, X однозначно определяется числом вершин орграфа G) и всевозможными входными векторами C, будем называть *задачей* X. Два допустимых решения $\boldsymbol{x}, \boldsymbol{y} \in X$ задачи X называются *смежными*, если найдется вектор C такой, что они, и только они, являются оптимальными относительно C. Подмножество $Y \subseteq X$ называется *кликой*, если любая пара $\boldsymbol{x}, \boldsymbol{y} \in Y$ смежна.

Выпуклая оболочка $\operatorname{conv}(X)$ называется *многогранником задачи* X. Так как X в задаче коммивояжера является подмножеством вершин единичного куба, то X совпадает с множеством вершин многогранника $\operatorname{conv}(X)$. В этой терминологии два решения $\boldsymbol{x}, \boldsymbol{y} \in X$ смежны тогда и только тогда, когда смежны соответствующие вершины многогранника $\operatorname{conv}(X)$ [7]. Известно [12], что все вершины многогранника коммивояжера попарно смежны при $n < 6$, где n — число вершин орграфа G, в котором требуется найти оптимальный гамильтонов контур.

Алгоритмы прямого типа относятся к классу линейных разделяющих алгоритмов, которые удобно представлять в виде линейных разделяющих деревьев.

Определение 1 ([7, определение 1.3.1])**.** Линейным разделяющим деревом *задачи* $X \subset \mathbb{Z}^m$ *называется ориентированное дерево, обладающее следующими свойствами:*

а) в каждый узел, за исключением одного, называемого корнем, входит ровно одна дуга; дуг, входящих в корень, нет;

б) для каждого узла либо имеется две выходящих из него дуги, либо таких дуг нет вообще; в первом случае узел называется внутренним, во втором — внешним, или листом;

в) каждому внутреннему узлу соответствует некоторый вектор $B \in \mathbb{Z}^m$*;*

г) каждому листу соответствует некоторый элемент из X *(нескольким листьям может соответствовать один и тот же элемент множества* X*);*

д) каждой дуге d *соответствует число* $\operatorname{sgn} d$*, равное* 1 *либо* -1*; две дуги, выходящие из одного узла, имеют различные значения;*

е) для каждой цепи $W = B_1 d_1 B_2 d_2 \ldots B_k d_k \boldsymbol{x}$*, соединяющей корень и лист (в обозначении цепи перечислены соответствующие ее узлам векторы* B_i*; дуга* d_i *выходит из узла* B_i*,* $i \in [k]$*), и для любого входа* C *из неравенств* $\langle B_i, C\rangle \operatorname{sgn} d_i \geq 0$*,* $i \in [k]$*, следует, что решение* $\boldsymbol{x}$ *является оптимальным относительно* C*.*

Таким образом, в рамках теории линейных разделяющих алгоритмов внимание уделяется только тем операциям, где выполняется проверка условий вида $\langle B, C\rangle \geq 0$, где C — вектор входных данных. Так, например, в строке 5 алгоритма 2 на самом первом шаге цикла проверяется неравенство $\infty > C_{11}$; на втором шаге проверяется условие $C_{11} > C_{12}$, и т. д. А в функциях `HamiltonCycle` и `ForbiddenArc` алгоритма 1, с точки зрения линейных разделяющих алгоритмов, не происходит

[3] Элементы матрицы всегда можно выписать в строку или столбец.

ничего интересного, так как не выполняются никакие сравнения с элементами вектора входных данных.

Процесс работы линейного разделяющего алгоритма для фиксированного вектора входных данных C представляет собой некоторую цепь $B_1 d_1 B_2 d_2 \dots B_m d_m \boldsymbol{x}$, соединяющую корень B_1 и некоторый лист $\boldsymbol{x}$ соответствующего линейного разделяющего дерева. Листом в нашем случае является гамильтонов контур (точнее, его характеристический вектор), являющийся оптимальным относительно C.

Пусть B — некоторый внутренний узел в линейном разделяющем дереве рассматриваемого алгоритма, а X — множество всех допустимых решений (множество меток всех листьев). Обозначим через X_B, $X_B \subseteq X$, множество меток всех листьев этого дерева, которым предшествует узел B, а через X_B^+ и X_B^- обозначим подмножества множества X_B, соответствующие двум выходящим из B дугам. Очевидно, $X_B = X_B^+ \cup X_B^-$. Обозначим через $R_B^- = X_B^+ \setminus X_B^-$ множество меток, отбрасываемых при переходе по «отрицательной» дуге. По аналогии определим множество меток $R_B^+ = X_B^- \setminus X_B^+$, отбрасываемых при переходе по «положительной» дуге.

Определение 2 ([7, определение 1.4.2]). *Линейное разделяющее дерево называется деревом* прямого типа*, если для любого внутреннего узла B и для любой клики $Y \subseteq X$ выполняется неравенство*

$$\min\{|R_B^+ \cap Y|, |R_B^- \cap Y|\} \le 1. \tag{1}$$

Непосредственно из определения следует, что высота дерева прямого типа (то есть число сравнений, используемых алгоритмом в худшем случае) для задачи X не может быть меньше, чем $\omega(X) - 1$, где $\omega(X)$ — кликовое число множества X [7, теорема 1.4.3].

Если же мы хотим доказать, что некий алгоритм не является алгоритмом прямого типа, достаточно указать клику Y, состоящую из четырех решений, и узел B такие, что $|R_B^+ \cap Y| = |R_B^- \cap Y| = 2$.

Для каждого $\boldsymbol{x} \in X$ определим *конус исходных данных*

$$K(\boldsymbol{x}) = \{C \mid \langle \boldsymbol{x}, C\rangle \le \langle \boldsymbol{y}, C\rangle,\ \forall \boldsymbol{y} \in X\}.$$

Т. е. $K(\boldsymbol{x})$ состоит из всех векторов C таких, что $\boldsymbol{x}$ оптимален относительно C.

Определение 3 ([7, определение 1.4.4]). *Линейное разделяющее дерево называется деревом* «прямого типа»*, если каждая цепь $B_1 d_1 B_2 d_2 \dots B_k d_k \boldsymbol{x}$, соединяющая корень и лист, удовлетворяет условиям:*

(*) для любого $\boldsymbol{y} \in X$, смежного с $\boldsymbol{x}$, найдется такой номер $i \in [k]$, что условия $\langle B_i, C\rangle \operatorname{sgn} d_i > 0$ и $C \in K(\boldsymbol{y})$ несовместны;

(**) для любого $i \in [k]$ из несовместности условий

$$\langle B_i, C\rangle \operatorname{sgn} d_i > 0 \qquad \text{и} \qquad C \in K(\boldsymbol{y})$$

для $\boldsymbol{y}$, смежного с $\boldsymbol{x}$, и из телесности конуса

$$K(\boldsymbol{x}) \cap \{C \mid \langle B_i, C\rangle \operatorname{sgn} d_i \le 0\}$$

следует, что ветвь, начинающаяся в узле B_i с дугой $-d_i$, имеет хотя бы один лист, помеченный $\boldsymbol{x}$.

Деревья «прямого типа» с деревьями прямого типа объединяет тот факт, что их высота тоже ограничена снизу величиной $\omega(X) - 1$ [7, теорема 1.4.5].

Чтобы доказать, что алгоритм 1 не является алгоритмом «прямого типа», мы ограничимся проверкой условия (*) из этого определения. А именно, мы укажем вполне конкретный входной вектор C^*, который однозначно определит некоторую цепь $B_1 d_1 B_2 d_2 \dots B_k d_k \boldsymbol{x}$. Далее будет выбран $\boldsymbol{y} \in X$, смежный с $\boldsymbol{x}$, для которого условия $\langle B_i, C\rangle \operatorname{sgn} d_i > 0$ и $C \in K(\boldsymbol{y})$ совместны при любом $i \in [k]$. Обратим особое внимание на то, что нам нужно будет проверить совместность условий $\langle B_i, C\rangle \operatorname{sgn} d_i > 0$ и $C \in K(\boldsymbol{y})$ отдельно для каждого $i \in [k]$, вне зависимости от результатов других сравнений. То есть для каждого $i \in [k]$ достаточно указать C_i такой, что $\langle B_i, C_i\rangle \operatorname{sgn} d_i > 0$ и $C_i \in K(\boldsymbol{y})$.

3. Алгоритм 1 не является прямым

Рассмотрим задачу коммивояжера в полном орграфе на 5 вершинах. Множество допустимых решений X такой задачи состоит из двадцати четырех 0/1-векторов, соответствующих гамильтоновым контурам в этом орграфе. Все 24 решения попарно смежны [12].

Предположим, что элементы матрицы длин дуг $C \in \mathbb{Z}^{5\times 5}$ удовлетворяют следующим условиям:

$$\begin{aligned} &c_{12} \le c_{13}, \quad c_{12} \le c_{14}, \quad c_{12} \le c_{15}, \\ &c_{21} \le c_{23}, \quad c_{21} \le c_{24}, \quad c_{21} \le c_{25}, \\ &c_{31} > c_{32}, \quad c_{32} > c_{34}, \quad c_{34} > c_{35}. \end{aligned} \tag{2}$$

В самом начале работы рассматриваемого алгоритма выполняется процедура редуцирования этой матрицы (алгоритм 2). Мы ограничимся рассмотрением этапа редуцирования строк. В результате последовательных сравнений в первой строке выбирается наименьший элемент (в данном случае c_{12}) и вычитается из всех её элементов. Далее выбирается минимальный элемент во второй строке, им оказывается c_{21}, и минимальный элемент в третьей строке — c_{35}. После этого алгоритм переходит к проверке неравенства

$$c_{41} > c_{42} \tag{3}$$

(сравнение $\infty > c_{41}$ присутствует в алгоритме исключительно для краткости описания и не несет никакой информации). Соответствующий узел линейного разделяющего дерева алгоритма обозначим B. Ясно, что алгоритм попадает в этот узел дерева, если, и только если для входного вектора C выполняются условия (2).

Рассмотрим характеристические вектора четырех гамильтоновых контуров:

$$\boldsymbol{x} = \begin{pmatrix} & 0 & 0 & 1 & 0 \\ 0 & & 0 & 0 & 1 \\ 1 & 0 & & 0 & 0 \\ 0 & 1 & 0 & & 0 \\ 0 & 0 & 1 & 0 & \end{pmatrix}, \quad \boldsymbol{y} = \begin{pmatrix} & 0 & 0 & 0 & 1 \\ 0 & & 1 & 0 & 0 \\ 1 & 0 & & 0 & 0 \\ 0 & 1 & 0 & & 0 \\ 0 & 0 & 0 & 1 & \end{pmatrix},$$

$$\boldsymbol{z} = \begin{pmatrix} & 0 & 1 & 0 & 0 \\ 0 & & 0 & 0 & 1 \\ 0 & 1 & & 0 & 0 \\ 1 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 1 & \end{pmatrix}, \quad \boldsymbol{w} = \begin{pmatrix} & 0 & 0 & 0 & 1 \\ 0 & & 0 & 1 & 0 \\ 0 & 1 & & 0 & 0 \\ 1 & 0 & 0 & & 0 \\ 0 & 0 & 1 & 0 & \end{pmatrix}.$$

Нетрудно проверить, что входные векторы

$$C_x = \begin{pmatrix} & 0 & 6 & 1 & 6 \\ 0 & & 6 & 6 & 1 \\ 3 & 2 & & 1 & 0 \\ 6 & 0 & 6 & & 6 \\ 6 & 6 & 0 & 6 & \end{pmatrix}, \quad C_y = \begin{pmatrix} & 0 & 6 & 6 & 1 \\ 0 & & 1 & 6 & 6 \\ 3 & 2 & & 1 & 0 \\ 6 & 0 & 6 & & 6 \\ 6 & 6 & 6 & 0 & \end{pmatrix},$$

$$C_z = \begin{pmatrix} & 0 & 1 & 6 & 6 \\ 0 & & 6 & 6 & 1 \\ 6 & 3 & & 1 & 0 \\ 0 & 6 & 6 & & 6 \\ 6 & 6 & 6 & 0 & \end{pmatrix}, \quad C_w = \begin{pmatrix} & 0 & 6 & 6 & 1 \\ 0 & & 6 & 1 & 6 \\ 6 & 3 & & 1 & 0 \\ 0 & 6 & 6 & & 6 \\ 6 & 6 & 0 & 6 & \end{pmatrix}$$

удовлетворяют условиям (2), а для каждого $\boldsymbol{t} \in \{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \boldsymbol{w}\}$ и для любого $\boldsymbol{s} \in X \setminus \{\boldsymbol{t}\}$ выполняется неравенство $\langle \boldsymbol{t}, C_t \rangle = 5 < \langle \boldsymbol{s}, C_t \rangle$. Следовательно, все четыре вектора входят в множество меток X_B всех листьев дерева алгоритма, которым предшествует узел B.

Покажем, что $\boldsymbol{z}$ и $\boldsymbol{w}$ входят в множество меток R_B^+, отбрасываемых при выполнении неравенства (3), а $\boldsymbol{x}$ и $\boldsymbol{y}$ входят в множество меток R_B^-, отбрасываемых при невыполнении неравенства (3).

Предположим, что для входной матрицы C выполнены условия (2) и неравенство (3). Тогда $\langle \boldsymbol{z}, C \rangle > \langle \boldsymbol{z}', C \rangle$ для

$$\boldsymbol{z}' = \begin{pmatrix} & 0 & 1 & 0 & 0 \\ 1 & & 0 & 0 & 0 \\ 0 & 0 & & 0 & 1 \\ 0 & 1 & 0 & & 0 \\ 0 & 0 & 0 & 1 & \end{pmatrix}.$$

Аналогично, $\langle \boldsymbol{w}, C \rangle > \langle \boldsymbol{w}', C \rangle$ для

$$\boldsymbol{w}' = \begin{pmatrix} & 0 & 0 & 0 & 1 \\ 1 & & 0 & 0 & 0 \\ 0 & 0 & & 1 & 0 \\ 0 & 1 & 0 & & 0 \\ 0 & 0 & 1 & 0 & \end{pmatrix}.$$

Таким образом, $\boldsymbol{z}, \boldsymbol{w} \in R_B^+$.

Предположим, что для C выполнены условия (2), но не выполнено неравенство (3). Тогда $\langle \boldsymbol{x}, C \rangle > \langle \boldsymbol{x}', C \rangle$ для

$$\boldsymbol{x}' = \begin{pmatrix} & 1 & 0 & 0 & 0 \\ 0 & & 0 & 0 & 1 \\ 0 & 0 & & 1 & 0 \\ 1 & 0 & 0 & & 0 \\ 0 & 0 & 1 & 0 & \end{pmatrix},$$

и $\langle \boldsymbol{y}, C \rangle > \langle \boldsymbol{y}', C \rangle$ для

$$\boldsymbol{y}' = \begin{pmatrix} & 1 & 0 & 0 & 0 \\ 0 & & 1 & 0 & 0 \\ 0 & 0 & & 0 & 1 \\ 1 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 1 & \end{pmatrix}.$$

Следовательно, $\boldsymbol{z}, \boldsymbol{w} \in R_B^+$.

Таким образом, условие (1) для данного узла B не выполнено, и алгоритм 1 не является алгоритмом прямого типа.

4. Алгоритм 1 не является «прямым»

При анализе алгоритма 1, как линейного разделяющего дерева, нам будут встречаться только неравенства следующего вида:

$$\langle B^+, C \rangle - \langle B^-, C \rangle > 0, \tag{4}$$

где $C \in \mathbb{Z}^{n^2}$ — вектор входных данных,

$$B^+, B^- \in \{0, 1\}^{n^2}, \quad \langle B^+, B^- \rangle = 0 \quad \text{и} \quad \langle B^+, \mathbf{1} \rangle = \langle B^-, \mathbf{1} \rangle > 0, \tag{5}$$

$\mathbf{1}$ — вектор из единиц. Иными словами, условие (5) означает, что множества единичных координат для B^+ и B^- равномощны и не пересекаются. Для каждого такого неравенства и для некоторого

допустимого решения $\boldsymbol{y} \in X \subset \{0,1\}^{n^2}$ нам нужно будет проверить, что существует $C \in K(\boldsymbol{y})$, для которого это неравенство выполнено. Такой анализ существенно упрощается, если воспользоваться следующим критерием.

Лемма 1. *Пусть $\boldsymbol{y} \in \{0,1\}^{n^2}$ — характеристический вектор некоторого гамильтонова контура в полном орграфе $G = ([n], A)$. Если выполняются условия (5) и $\langle B^+, \boldsymbol{y}\rangle \le 2$, то неравенство (4) и условие $C \in K(\boldsymbol{y})$ совместны.*

Доказательство. Пусть

$$S = \{(i,j) \in [n]^2 \mid y_{ij} = 1 \text{ и } B^+_{ij} = 0\}.$$

Из условия $\langle B^+, \boldsymbol{y}\rangle \le 2$ следует, что $|S| \ge n-2$. Положим

$$C := \mathbf{4} - B^-$$

и, после этого, $C_{ij} := 0$ для $(i,j) \in S$.
Тогда $\langle B^+, C\rangle = \langle B^+, \mathbf{4} - B^-\rangle = \langle B^+, \mathbf{4}\rangle$ и $\langle B^-, C\rangle \le \langle B^-, \mathbf{4} - B^-\rangle = \langle B^+, \mathbf{4}\rangle - \langle B^-, B^-\rangle$ (так как B^+ и B^- удовлетворяют условиям (5)). Следовательно, неравенство (4) для такого C будет выполнено.

Покажем теперь, что $\langle \boldsymbol{y}, C\rangle < \langle \boldsymbol{x}, C\rangle$ для любого $\boldsymbol{x} \in X \setminus \boldsymbol{y}$.

Очевидно, $\langle \boldsymbol{y}, C\rangle = (n - |S|)4 \le 8$.

Пусть $\boldsymbol{x} \in X$. Заметим, что если $\langle \boldsymbol{y}, \boldsymbol{x}\rangle \ge n-2$, то $\boldsymbol{x} = \boldsymbol{y}$, так как любой гамильтонов контур в орграфе на n вершинах однозначно определяется по любым своим $n-2$ дугам. Следовательно, $\langle \boldsymbol{x}, C\rangle \ge 3 \cdot 3 = 9$ для любого $\boldsymbol{x} \in X \setminus \boldsymbol{y}$. □

В частности, условия леммы выполнены, если в B^+ не более двух единиц.

Итак, положим $n = 4$ и рассмотрим следующий вектор входных данных (вместо бесконечности будем подставлять пробел):

$$C^* := \begin{pmatrix} & 0 & 2 & 1 \\ 2 & & 0 & 2 \\ 1 & 2 & & 0 \\ 0 & 1 & 2 & \end{pmatrix}. \tag{6}$$

Ясно, что единственным оптимальным решением будет вектор

$$\boldsymbol{x} := \begin{pmatrix} & 1 & 0 & 0 \\ 0 & & 1 & 0 \\ 0 & 0 & & 1 \\ 1 & 0 & 0 & \end{pmatrix}$$

и соответствующий ему контур $\{(1,2),(2,3),(3,4),(4,1)\}$. Нетрудно проверяется, что множество всех допустимых решений X состоит из 6 попарно смежных векторов. Положим

$$\boldsymbol{y} := \begin{pmatrix} & 0 & 0 & 1 \\ 0 & & 1 & 0 \\ 1 & 0 & & 0 \\ 0 & 1 & 0 & \end{pmatrix}.$$

Обратим внимание, что $\boldsymbol{y}$ является вторым (после $\boldsymbol{x}$) по оптимальности относительно C^*. Именно это обстоятельство во многом упрощает дальнейшую проверку соответствующих сравнений.

В целом схема работы алгоритма при заданном входе C^* изображена на рис. 1.

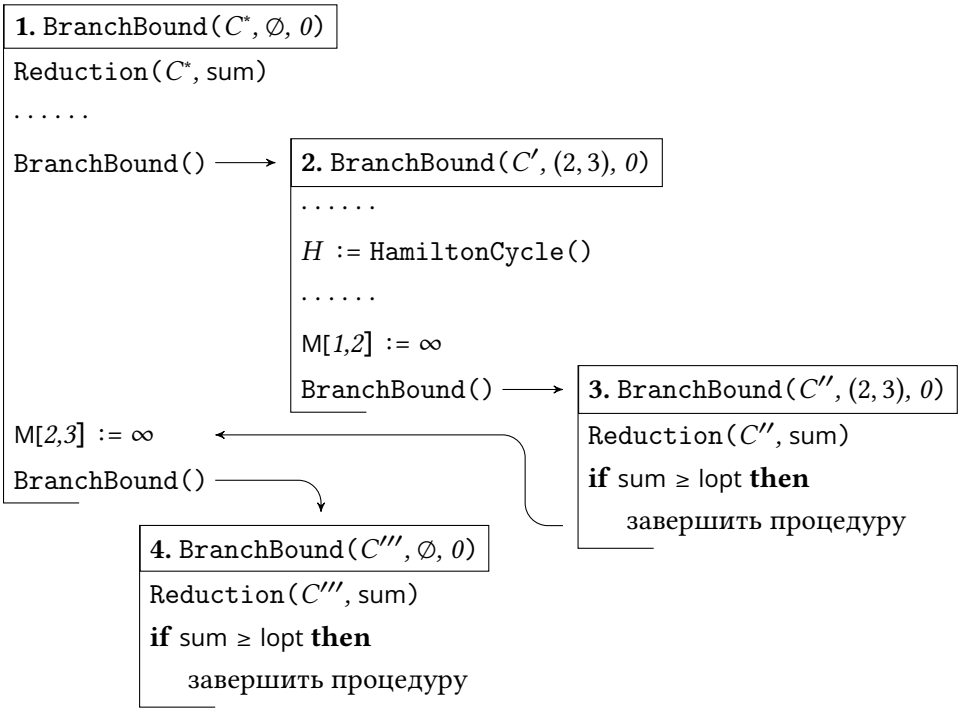

Fig. 1. General scheme of work of the algorithm 1 for the input given by the formula (6)

Рис. 1. Общая схема работы алгоритма 1 для входа, задаваемого формулой (6)

Рассмотрим, прежде всего, какие неравенства проверяются при первом входе в процедуру BranchBound с входом C^*. При редуцировании первой строки матрицы C^* (строка 5 алгоритма 2) проверяются (и выполняются) неравенства $\infty > C_{12}$, $C_{13} > C_{12}$ и $C_{14} > C_{12}$. Далее мы не будем рассматривать неравенства, в которых сумма (либо разность) элементов исходной матрицы сравнивается с бесконечностью, так как они всегда выполняются и совместны с любым допустимым решением. Заметим, что только что перечисленные неравенства удовлетворяют условиям леммы 1, так как $\langle B^+, \mathbf{1}\rangle = 1$. А значит, они совместны с условием $C \in K(\boldsymbol{y})$.

После редуцирования первой строки в её ячейках $\mathsf{M}[1, j]$, $j \in [4]$, содержатся разности $C_{1j} - C_{12}$, а переменная sum принимает значение C_{12}.

При редуцировании второй строки проверяются неравенства $C_{21} > C_{23}$ и $C_{24} > C_{23}$. Согласно лемме 1, они совместны с условием $C \in K(\boldsymbol{y})$.

После редуцирования второй строки в её ячейках $\mathsf{M}[2, j]$, $j \in [4]$, содержатся разности $C_{2j} - C_{23}$, а переменная sum принимает значение $C_{12} + C_{23}$.

При редуцировании последних двух строк ситуация полностью аналогична. После завершения редуцирования строк

$$\mathsf{sum} = C_{12} + C_{23} + C_{34} + C_{41},$$

$$\mathsf{M} = \begin{pmatrix} & 0 & C_{13} - C_{12} & C_{14} - C_{12} \\ C_{21} - C_{23} & & 0 & C_{24} - C_{23} \\ C_{31} - C_{34} & C_{32} - C_{34} & & 0 \\ 0 & C_{42} - C_{41} & C_{43} - C_{41} & \end{pmatrix}.$$

Далее, при редуцировании первого столбца проверяются неравенства $\mathsf{M}[2, 1] > \mathsf{M}[3, 1]$ и $\mathsf{M}[3, 1] > \mathsf{M}[4, 1]$. Нам известно, что $\mathsf{M}[2, 1] = C_{21} - C_{23}$, $\mathsf{M}[3, 1] = C_{31} - C_{34}$, $\mathsf{M}[4, 1] = C_{41} - C_{41} = 0$. Следовательно, проверяются неравенства $C_{21} - C_{23} > C_{31} - C_{34}$ и $C_{31} - C_{34} > 0$. Каждое из них удовлетворяет условиям леммы 1.

При редуцировании оставшихся трех столбцов ситуация повторяется. Значение sum при редуцировании столбцов не меняется, так как каждый столбец уже содержит нули.

После этого в алгоритме 1 выполняется проверка условия sum ≥ lopt. Но lopt = ∞. Поэтому алгоритм переходит к вычислению функции `ChooseArc`.

Первым нулевым элементом является $\mathsf{M}[1,2]$. После этого в строке 8 алгоритма 3 выполняются сравнения $\infty > \mathsf{M}[1,3]$ и $\mathsf{M}[1,3] > \mathsf{M}[1,4]$. При этом, после предыдущего этапа редукции, имеем $\mathsf{M}[1,3] = C_{13} - C_{12}$ и $\mathsf{M}[1,4] = C_{14} - C_{12}$. Очевидно, неравенство $C_{13} - C_{12} > C_{14} - C_{12}$ удовлетворяет условиям леммы 1. На этом шаге выполняется присвоение $m := C_{14} - C_{12}$. Далее, в строке 11 алгоритма 3 выполняются сравнения $\infty > \mathsf{M}[3,2]$ и $\mathsf{M}[3,2] > \mathsf{M}[4,2]$. При этом $\mathsf{M}[3,2] = C_{32} - C_{34}$ и $\mathsf{M}[4,2] = C_{42} - C_{41}$. Условия леммы 1 снова выполнены. На этом шаге выполняется присвоение $k := C_{42} - C_{41}$. Далее выполняется сравнение $m + k > -1$ или, что то же самое, $C_{14} - C_{12} + C_{42} - C_{41} > -1$. Очевидно, это неравенство совместимо с условием $C \in K(\boldsymbol{y})$. В переменную w заносится значение выражения $C_{14} - C_{12} + C_{42} - C_{41}$.

Второй нулевой элемент — $\mathsf{M}[2,3]$. Действуя по аналогии, перечислим только нетривиальные сравнения. Неравенство $\mathsf{M}[2,1] \le \mathsf{M}[2,4]$ или $C_{21} - C_{23} \le C_{24} - C_{23}$, очевидно, совместимо с условием $C \in K(\boldsymbol{y})$. Неравенство $\mathsf{M}[1,3] \le \mathsf{M}[4,3]$ тоже совместимо. Далее, в строке 12 проверяется неравенство $m + k > w$ или, с учетом предыдущих действий,

$$C_{21} - C_{23} + C_{13} - C_{12} > C_{14} - C_{12} + C_{42} - C_{41}.$$

Очевидно, оно удовлетворяет условиям леммы 1. После этого шага

$$w = C_{21} - C_{23} + C_{13} - C_{12}.$$

Третий нулевой элемент — $\mathsf{M}[3,4]$. Неравенство $\mathsf{M}[3,1] < \mathsf{M}[3,2]$ или $C_{31} - C_{34} < C_{32} - C_{34}$, очевидно, совместимо с условием $C \in K(\boldsymbol{y})$. Неравенство $\mathsf{M}[1,4] < \mathsf{M}[2,4]$ тоже совместимо. Условие $m + k < w$ имеет вид

$$C_{14} - C_{12} + C_{31} - C_{34} < C_{21} - C_{23} + C_{13} - C_{12}$$

и тоже совместимо с условием $C \in K(\boldsymbol{y})$.

Четвертый нулевой элемент — $\mathsf{M}[4,1]$. Легко проверить, что $\mathsf{M}[4,2] < \mathsf{M}[4,3]$ и $\mathsf{M}[3,1] < \mathsf{M}[2,1]$ совместимы с условием $C \in K(\boldsymbol{y})$. Условие $m + k < w$ имеет вид

$$C_{31} - C_{34} + C_{42} - C_{41} < C_{21} - C_{23} + C_{13} - C_{12}$$

и тоже совместимо.

В данный момент мы все еще находимся в первом экземпляре процедуры `BranchBound`. После описанного выше выполнения функции `ChooseArc` выбирается дуга $(i,j) = (2,3)$ (сумма $m + k$ для нее оказалась наибольшей), из матрицы M вычеркиваются 2-я строка и 3-й столбец, а дуга $(3,2)$ становится запрещенной. На вход второго экземпляра процедуры `BranchBound` подается матрица

$$C' := \begin{pmatrix} & 0 & 1 \\ & & \\ 1 & & 0 \\ 0 & 1 & \end{pmatrix}$$

(пустая строка и пустой столбец оставлены для удобства чтения). Ясно, что при её редуцировании ничего нового не происходит, так как каждая строка и каждый столбец содержат нули. При вызове функции `ChooseArc` в строке 12 выполняются следующие сравнения типа $m + k > w$.

$$C_{14} - C_{12} + C_{42} - C_{41} > -1.$$

Очевидно, это неравенство совместимо с условием $C \in K(\boldsymbol{y})$. Далее, выполняется неравенство

$$C_{31} - C_{34} + C_{14} - C_{12} \le C_{14} - C_{12} + C_{42} - C_{41},$$

которое удовлетворяет условиям леммы 1. Следующее сравнение

$$C_{31} - C_{34} + C_{42} - C_{41} \le C_{14} - C_{12} + C_{42} - C_{41}$$

тоже совместимо с $C \in K(\boldsymbol{y})$.

Итак, после вызова функции `ChooseArc` во втором экземпляре `BranchBound`, выбирается дуга $(1, 2)$. Гамильтонов цикл с дугами $(2, 3)$ и $(1, 2)$ определяется однозначно. Выполняется присвоение

$$\mathsf{lopt} := C_{12} + C_{23} + C_{34} + C_{41}.$$

После этого алгоритм переходит к рассмотрению случаев, когда контур содержит дугу $(2, 3)$, но не содержит $(1, 2)$. Запускается третий экземпляр `BranchBound` с матрицей

$$C'' := \begin{pmatrix} & & & 1 \\ & & & \\ 1 & & & 0 \\ 0 & 1 & & \end{pmatrix}.$$

При редуцировании две единицы заменяются нулями. Никакие «отбрасывающие» сравнения не выполняются. Значение переменной sum увеличивается на $\mathsf{M}[1, 4] = C_{14} - C_{12}$ и на $\mathsf{M}[4, 2] = C_{42} - C_{41}$. Текущий экземпляр процедуры завершается в строке 3 после проверки неравенства $\mathsf{sum} \ge \mathsf{lopt}$:

$$(C_{14} - C_{12}) + (C_{42} - C_{41}) > 0.$$

Заметим, что допустимое решение $\boldsymbol{y}$ полностью отбраковывается алгоритмом именно на этом шаге (с учетом ранее проверенного неравенства $C_{31} > C_{34}$). Тем не менее, это неравенство удовлетворяет условиям леммы 1 и, следовательно, совместно с условием $C \in K(\boldsymbol{y})$.

Вместе с третьим экземпляром процедуры `BranchBound` завершается и второй её экземпляр. Алгоритм переходит к выполнению предпоследней строки в первом экземпляре. В этом экземпляре

$$\mathsf{sum} = C_{12} + C_{23} + C_{34} + C_{41}.$$

Для разбора случаев, когда контур не содержит дугу $(2, 3)$, вызывается четвертый экземпляр процедуры с матрицей

$$C''' := \begin{pmatrix} & 0 & 2 & 1 \\ 2 & & & 2 \\ 1 & 2 & & 0 \\ 0 & 1 & 2 & \end{pmatrix}.$$

При редуцировании второй строки выполняется сравнение $\mathsf{M}[2, 1] \le \mathsf{M}[2, 4]$. При редуцировании третьего столбца — $\mathsf{M}[1, 3] \le \mathsf{M}[4, 3]$. Очевидно, ни то ни другое не отбрасывают целиком конус $K(\boldsymbol{y})$. Значение sum увеличивается на $(C_{21} - C_{23}) + (C_{13} - C_{12})$.

И, наконец, сравнение $\mathsf{sum} \ge \mathsf{lopt}$ завершает этот четвертый экземпляр процедуры и вообще весь алгоритм. Это сравнение имеет вид

$$(C_{21} - C_{23}) + (C_{13} - C_{12}) \ge 0$$

и тоже совместимо с условием $C \in K(\boldsymbol{y})$.

Итак, условие (*) из определения 3 не выполнено для этого алгоритма.

References

[1] V. Bondarenko, A. Nikolaev, and D. Shovgenov, "1-skeletons of the spanning tree problems with additional constraints", *Automatic Control and Computer Sciences*, vol. 51, no. 7, pp. 682–688, 2017.

[2] V. Bondarenko and A. Nikolaev, "On graphs of the cone decompositions for the min-cut and max-cut problems", *International Journal of Mathematics and Mathematical Sciences*, vol. 2016, 2016.

[3] V. Bondarenko and A. Nikolaev, "Some properties of the skeleton of the pyramidal tours polytope", *Electronic Notes in Discrete Mathematics*, vol. 61, pp. 131–137, 2017.

[4] V. A. Bondarenko, A. V. Nikolaev, and D. Shovgenov, "Polyhedral characteristics of balanced and unbalanced bipartite subgraph problems", *Automatic Control and Computer Sciences*, vol. 51, no. 7, pp. 576–585, 2017.

[5] V. Bondarenko and A. Nikolaev, "On the skeleton of the polytope of pyramidal tours", *Journal of Applied and Industrial Mathematics*, vol. 12, no. 1, pp. 9–18, 2018.

[6] V. Bondarenko, "Nonpolynomial lowerbound of the traveling salesman problem complexity in one class of algorithms", *Automation and Remote Control*, vol. 44, no. 9, pp. 1137–1142, 1983.

[7] V. Bondarenko, "Geometricheskie metody sistemnogo analiza v kombinatornoy optimizatsii", *diss. … dokt. fiz.-mat. nauk, Yaroslavl*, 1993.

[8] V. Bondarenko and A. Maksimenko, *Geometricheskie konstruktsii i slozhnost v kombinatornoy optimizatsii*. Moskva: URSS, 2008, 182 pp.

[9] A. Maksimenko, "Kharakteristiki slozhnosti: klikovoe chislo grafa mnogogrannika i chislo pryamougolnogo pokrytiya", *Modelirovanie i analiz informatsionnykh sistem*, vol. 21, no. 5, pp. 116–130, 2014.

[10] J. Little, K. Murty, D. Sweeney, and C. Karel, "An algorithm for the traveling salesman problem", *Operations research*, vol. 11, no. 6, pp. 972–989, 1963.

[11] E. Reingold, J. Nievergelt, and N. Deo, *Combinatorial algorithms: theory and practice*. Pearson College Div, 1977, 433 pp.

[12] M. Padberg and M. Rao, "The travelling salesman problem and a class of polyhedra of diameter two", *Mathematical Programming*, vol. 7, no. 1, pp. 32–45, 1974.