УДК 519.16, 519.17 DOI: 10.17587/it.24.698-704

М. В. Ульянов, д-р техн. наук, проф., вед. науч. сотр., проф., muljanov@mail.ru,
Институт проблем управления им. В. А. Трапезникова РАН,
Московский государственный университет им. М. В. Ломоносова,
М. И. Фомичев, магистрант, michan94@yandex.ru,
Национальный исследовательский университет "Высшая школа экономики"

Подходы к организации поискового дерева решений в методе ветвей и границ для асимметричной задачи коммивояжера[1]

Повышение временной эффективности программных реализаций метода ветвей и границ для асимметричной задачи коммивояжера может быть достигнуто как за счет выбора наиболее приемлемой структуры данных, обеспечивающей эффективные по времени операции с листьями поискового дерева решений, так и за счет использования дополнительной памяти для хранения усеченных матриц в листьях поискового дерева решений. Дополнительно могут быть предложены и различные подходы к хранению и обработке матриц, соответствующих листьям поискового дерева. Такое исследование должно опираться на особенности операций с деревом и матрицами, порожденные спецификой метода ветвей и границ. Описанию различных подходов к организации, хранению и доступу к элементам поискового дерева решений в совокупности со способами хранения матриц в листьях такого дерева и посвящена настоящая статья.

Ключевые слова: *метод ветвей и границ, асимметричная задача коммивояжера, поисковое дерево решений, структуры данных, структуры хранения усеченных матриц*

Введение

Достаточно широкий спектр задач, порожденных практическими постановками в области информационных технологий, логистики и бизнес-информатики сводится к классической задаче коммивояжера (TSP). При этом некоторые постановки порождают асимметричную задачу коммивояжера (ATSP), например, стоимости перелетов из одного города в другой и обратно, как правило, не совпадают. При решении ATSP перед исследователями стоит задача выбора между эвристическими и точными методами решения, связанная как с размерностью исходной задачи, так и с ограничениями по допустимому времени решения. Обилие эвристических методов не означает отказа от возможности получения точных решений ATSP для небольших значений размерности. Далее объектом исследования будет точный алгоритм решения ATSP, реализующий классический метод ветвей и границ [1].

Экспериментальные данные свидетельствуют об экспоненциальном росте сложности в среднем в зависимости от размерности задачи и, следовательно, среднего времени получения точного решения при использовании метода ветвей и границ для индивидуальной ATSP [2, 3]. Тем не менее при реализации на современных компьютерах метод позволяет получить точное решение для задач, содержащих не более 60...70 вершин, за приемлемое время, при этом очевидный интерес представляет задача повышения его временной эффективности.

Для некоторых типов задач и решающих их алгоритмов повышение временной эффективности может быть получено за счет использования дополнительного доступного объема памяти. Такая ситуация возникает в задачах, допускающих замену повторных вычислений на хранение ранее полученных промежуточных результатов. Этот подход может быть использован и для алгоритмов, реализующих метод ветвей и границ для задачи коммивояжера.

Ресурсная эффективность различных реализаций метода ветвей и границ для асимметричной задачи коммивояжера зависит в том числе и от способов организации и работы с поисковым деревом решений, порождаемым этим методом. Классическая дилемма "время—память" реализуется здесь либо в виде варианта хранения усеченных матриц в вершинах дерева, что приводит к сокращению трудоемкости при дополнительных емкостных затратах, либо как перевычисление матрицы для текущей верши-

[1]Работа выполнена при поддержке Гранта РФФИ 18-07-00656.

ны, что ведет к увеличению трудоемкости при экономии памяти.

Повышение временной эффективности может быть, очевидно, достигнуто за счет хранения матриц в листьях поискового дерева решений. Такой подход требует детального исследования в целях выбора структуры данных, обеспечивающей эффективные по времени операции с листьями поискового дерева решений. Вместе с тем могут быть предложены и различные подходы к хранению и обработке матриц, соответствующих листьям поискового дерева. Для формулировки рекомендаций по выбору структуры данных и способа хранения матриц необходимо изучение особенностей операций с деревом и матрицами, порожденных спецификой метода ветвей и границ. Таким образом, предметом исследования данной статьи являются структуры данных, предназначенные для хранения поискового дерева решений, и способы хранения матриц в листьях этого дерева. Полученные при реализации этих предложений экспериментальные данные позволят дать оценки размерности задач, решаемых за приемлемое время при соответствующих затратах дополнительной памяти, и сформулировать рекомендации при выборе метода ветвей и границ как точного метода решения ATSP в практических задачах информационных технологий, логистики и бизнес-информатики.

Применение метода ветвей и границ к задаче коммивояжера

Идея применения метода ветвей и границ к ATSP, предложенная в работе [1], предполагает разделение множества допустимых решений на подмножества в целях дальнейшего сокращения перебора, это разделение называется процедурой ветвления. Под допустимым решением понимается некоторый гамильтонов цикл в полном графе, называемый туром, стоимость которого определяется суммой весов входящих в него дуг. С каждым таким подмножеством должна быть связана оценка, обеспечивающая отсечение тех подмножеств, которые заведомо не содержат оптимального решения; процедура ее построения называется процедурой вычисления границ. Тем самым метод приводит к исследованию древовидной модели пространства решений. В ATSP таким исходным множеством является множество всех туров коммивояжера, на котором минимизируется целевой функционал, задающий стоимость тура. Для построения алгоритма необходимо разработать две основные процедуры — процедуру ветвления и процедуру вычисления границ.

Авторы в работе [1] предлагают следующую процедуру ветвления. Построение поискового дерева решений начинается с корня, который будет соответствовать множеству "всех возможных туров коммивояжера", т. е. корень дерева представляет множество всех $(n-1)!$ возможных туров в задаче с n городами. Ветви, выходящие из текущей вершины, определяются выбором дуги (для ATSP). Идея авторов алгоритма состоит в том, чтобы разделить текущее множество туров на два множества: одно — которое, весьма вероятно, содержит оптимальный тур, и другое — которое, вероятно, этого тура не содержит. Для этого предлагается специальный алгоритм выбора дуги, которая, вполне вероятно, входит в оптимальный тур. После выбора дуги текущее множество туров разделяется на два подмножества, в одно из которых входят все туры из текущего множества, содержащие выбранную дугу, т. е. проходящие через нее, а в другое — туры, не содержащие эту дугу. Заметим, что идея авторов алгоритма очень перспективна — если так организовать процесс ветвления, что на каждом шаге выбирается "правильное" ребро, то весь процесс будет завершен за n шагов.

С каждой вершиной поискового дерева связывается нижняя граница стоимости любого тура из подмножества, связанного с этой вершиной. Очевидно, что задача состоит в получении как можно более точных нижних границ. Причина этого следующая. Предположим, что уже получен конкретный тур T со стоимостью $C(T)$. Если нижняя граница, связанная с подмножеством туров, представленных некоторой другой вершиной поискового дерева, больше, чем $C(T)$, то до конца процесса поиска не нужно рассматривать эту и все следующие за ней вершины. В реализации это приводит к усечению поискового дерева решений путем отбрасывания всех листьев поискового дерева, имеющих стоимость большую, чем $C(T)$. Подробное изложение других этапов метода можно найти, например, в работах [4, 5].

Схема алгоритма метода ветвей и границ для задачи коммивояжера

В целях дальнейшего исследования приведем схему алгоритма метода ветвей и границ

для ATSP. Пусть X — текущая вершина поискового дерева, а (k, l) — дуга, по которой происходит ветвление. Обозначим Y и $\overline{Y}$ вершины дерева, порожденные из X. Вершина Y обозначает подмножество туров из X, проходящих через дугу (k, l), а подмножество $\overline{Y}$ — подмножество туров, не проходящих через дугу (k, l). Вычисленные нижние границы для подмножеств Y и $\overline{Y}$ обозначим $w(Y)$ и $w(\overline{Y})$ соответственно. Пусть z_0 — самый дешевый тур, известный алгоритму в данный момент. Мы предполагаем, что в вершинах поискового дерева решений в том или ином виде хранятся соответствующие усеченные матрицы.

Приведенная ниже схема алгоритма соответствует изложению метода в работе [4].

```
A(C, n) Схема алгоритма МВГ для задачи коммивояжера
     (n — размерность, C — матрица стоимостей)
     1. Инициализация.
     2. Приведение матрицы стоимостей C — вычисление нижней границы всех туров w(R).
     3. Установка корня поискового дерева решений X = R и w(X).
     While (w(X) < z0)
          begin
               4. Выбор ребра ветвления(k,l).
               5. Процесс ветвления. Создание вершины Y и вычисление w(Y).
               6. Процесс ветвления. Создание вершины Y и вычисление w(Y).
               If (размер матрицы стоимостей в вершине Y = 2)
                    then
                         begin
                              7. Наилучший тур в вершине Y, вычисление w(Y)
                              If (w(Y) < z0)
                                   then
                                        begin
                                             z0 = w(Y)(запоминаем текущий лучший тур)
                                        end
                         end
               8. Выбор следующей вершины поискового дерева решений и установка X
               9. Чтение матрицы, соответствующей выбранной вершине X,
                    из структуры хранения поискового дерева решений.
          end (while w(X) < z0)
Оптимальное (точное) решение со стоимостью z0 найдено
End.
```

Структуры данных для обслуживания поискового дерева решений

Сформулируем требования, предъявляемые к структуре данных, связанные с обеспечением временной эффективности. Для этого обратим внимание на некоторые этапы приведенной выше схемы и некоторые экспериментальные результаты:

- экспериментальные результаты работ [2, 3] свидетельствуют о том, что общее число порожденных вершин растет экспоненциально с размерностью задачи и для ATSP с 50 вершинами в худшем случае может достигать сотен миллионов;
- при выборе на этапе 8 следующей текущей вершины выбирается лист текущего поискового дерева решений, имеющий минимальную границу. За все время работы алгоритма число обращений к этапу 8 линейно зависит от общего числа порожденных вершин. Содержательно это означает, что структура данных, поддерживающая хранение дерева, должна обеспечить быстрый поиск листа дерева, имеющего минимальную границу;
- кроме того, на этапе 8, помимо обеспечения быстрого поиска листа дерева решений, также необходимо удалить этот лист из структуры для обслуживания поискового дерева решений, так как на этапах 5 и 6 из этой вершины будет обеспечено ветвление, и, как результат, данный лист становится внутренним (и не активным) узлом поискового дерева решений;

- поскольку на этапе 5 необходимо добавить вершину $\bar{Y}$ в поисковое дерево решений (при условии, что $w(\bar{Y}) < z0$), структура данных должна поддерживать операцию вставки элемента;
- в связи с тем что нижние оценки для разных вершин поискового дерева решений могут быть равны между собой, структура данных должна иметь возможность хранить несколько одинаковых элементов (несколько одинаковых ключей);
- так как на этапе 7 может быть найден тур, который является на данный момент работы алгоритма наилучшим из известных (или даже оптимальным), то необходимо удалить из структуры данных все листья дерева решений, которые имеют оценки, большие или равные стоимости найденного тура.

Учитывая вышеизложенные особенности структуры данных, можно сделать вывод, что необходимая структура данных должна представлять такой абстрактный тип данных, как очередь с приоритетом [6].

Согласно работам [6, 7] следующие структуры данных следует рассмотреть как реализацию очереди с приоритетом:

▲ упорядоченный список;
▲ двоичная (бинарная) куча;
▲ самобалансирующиеся двоичные (бинарные) деревья поиска (например, красно-черные деревья и АВЛ-деревья).

Эти структуры данных поддерживают все необходимые операции и обладают нужными свойствами. Кроме того, сложность операции по получению минимального элемента может быть сведена к $O(1)$, если кешировать минимальный элемент. Более того, эти структуры данных имеют линейную оценку требуемой памяти $O(n)$, где n — число хранимых элементов. Однако, на этом сходство этих структур данных заканчивается. Сложность операции по удалению минимального элемента для упорядоченного списка — $O(n)$, в то время как сложность аналогичной операции для самобалансирующегося двоичного дерева и двоичной кучи — $O(\log n)$. Вместе с тем сложность операции удаления элементов, чей ключ больше заданного числа, — $O(n)$, в то время как для самобалансирующегося двоичного дерева и двоичной кучи — $O(n \log n)$. Что касается операции вставки, то в худших случаях стоимость операции вставки для упорядоченного списка и двоичной кучи — $O(n)$, а для самобалансирующегося двоичного дерева — $O(\log n)$.

Вышеизложенные оценки сложности представлены в табл. 1.

Таблица 1

Сложность операций над структурами данных в худшем случае

Операция	Упорядоченный список	Двоичная куча	Самобалансирующееся двоичное дерево
Удаление наименьшего элемента	$O(n)$	$O(\log n)$	$O(\log n)$
Получение наименьшего элемента	$O(1)$	$O(1)$	$O(1)$
Вставка элемента	$O(n)$	$O(n)$	$O(\log n)$
Удаление элементов, чьи ключи больше, чем заданное значение	$O(n)$	$O(n \log n)$	$O(n \log n)$

Приведенные оценки сложности операций в худших случаях для рассмотренных структур данных не позволяют сделать однозначные выводы об их эффективности. Аргументированный выбор может быть сделан только на основе тщательного экспериментального исследования.

Способы хранения матриц в вершинах поискового дерева

Варианты решения в данном случае относятся к этапу 9 общей схемы. В этот момент новая текущая вершина поискового дерева X уже выбрана, и задача данного этапа — получить матрицу стоимостей, соответствующую вершине X.

Основная особенность состоит в том, что метод предполагает усечение матрицы при построении вершины, соответствующей подмножеству Y, путем исключения строки и столбца, соответствующих выбранной дуге ветвления. При этом могут быть предложены различные подходы к хранению матриц, а именно:

- подход, предполагающий прямое хранение матрицы (без усечения), при этом элементы исключенной строки и исключенного столбца помечаются специальным значением. Это приводит к увеличению (по оценке в два раза) требуемого объема памяти при возможном росте временной эффективности;

- подход, предполагающий хранение усеченных матриц, при котором необходимо вводить два специальных массива переиндексации (для строк и столбцов), элементы которых содержат "настоящие", т. е. не усеченные индексы для строк и столбцов усеченной матрицы. При этом мы сокращаем объем требуемой памяти за счет дополнительных операций, связанных с переиндексацией;
- комбинированный подход, предполагающий усеченное хранение и восстановление полной матрицы по усеченному описанию, с которой далее проводятся вычисления на этапах 4, 5 и 6 общей схемы. Этот подход, оставляя преимущество усеченного хранения (в смысле объема требуемой дополнительной памяти), позволяет сократить временные затраты при условии, что трудоемкость восстановления и обратной свертки полной матрицы окупится отсутствием двойной индексации при обработке. Очевидно, что это произойдет в случае, если объем вычислений на этапах 4, 5 и 6 общей схемы существенно превалирует над дополнительными затратами на восстановление и свертку.

Достаточно трудно сказать, какой из этих подходов является более эффективным с точки зрения времени выполнения, поскольку выбор ребра ветвления предполагает рассмотрение всех нулевых элементов приведенной матрицы [1, 4], число которых варьируется от n до $2n - 1$ в зависимости от особенностей данных. Вместе с тем временная эффективность переиндексации связана с особенностями компилятора и внутренними алгоритмами управления кеш-памятью компьютера, что не может быть аналитически точно описано.

Таким образом, решение о выборе способа хранения может быть принято только на основе анализа экспериментальных данных о временной эффективности.

Предварительные экспериментальные результаты

Для проведения экспериментального исследования был сформирован пул несимметричных матриц стоимостей. Размерность матриц от 20 до 43 с шагом один. Число различных матриц для каждой размерности: 100 000. Элементы матриц (стоимости перехода между городами) генерировались с помощью стандартного равномерного генератора псевдослучайных чисел в интервале (0; 1) с точностью до шестого знака после запятой.

Все эксперименты проводили на персональном компьютере со следующими характеристиками:
- процессор Intel i7 3770K 3800 МГц;
- оперативная память Kingston KHX1600C9D3P1 16 ГБ;
- материнская плата GIGABYTE GA-Z77X-D3H.

В качестве операционной системы использовался дистрибутив Linux CentOS 7.0 с версией ядра 3.10.0-3247.4.4.e17. Для минимизации шумов операционной системы были отключены ненужные для исследования фоновые процессы (например, фаервол), а также у дистрибутива отсутствуют компоненты графического пользовательского интерфейса (управление осуществляется посредством командной строки). Более того, был отключен своппинг, чтобы скорость работы HDD не сказывалась на времени работы реализации.

Реализация всех алгоритмов осуществлялась на языке C++ с использованием библиотеки стандартных шаблонов (STL). Версия компилятора: GNU GCC 4.8.5 20150623. Замер времени работы программных реализаций алгоритмов осуществлялся посредством chrono. Для замера используемой памяти использовались счетчики. Наличие погрешности у чисел с плавающей точкой (согласно стандарту IEEE 754-2008) может привести к измененному процессу ветвления. Как показывает практика, хотя процесс ветвления и изменяется, и ошибка может накапливаться, это не влияет на поиск оптимального тура.

В табл. 2 (с шагом два по размерности) и на рис. 1, 2 (см. вторую сторону обложки) показаны результаты экспериментов для метода ветвей и границ с использованием двоичной кучи (ARS_{heap}), упорядоченного списка (ARS_{ord_vector}) и самобалансирующегося двоичного дерева (в качестве представителя данного класса структур данных используется красно-черное дерево [6]) (ARS_{RBT}), кроме того, для сравнения (только на рис. 1) представлены результаты работы реализации метода ветвей и границ без хранения списка вершин, при этом поиск вершины дерева решений с минимальной границей осуществляется посредством обхода дерева решений ($ARS_{traverse\_tree}$).

Предварительные экспериментальные результаты показали, что за исключением двоичной кучи, использование которой дает худший

Таблица 2

Среднее затраченное время (в мкс) для ARS_{heap}, ARS_{ord_vector}, ARS_{RBT} и $ARS_{traverse\_tree}$

n	$\bar{t}_{ARS_{heap}}(n)$	$\bar{t}_{ARS_{ord\_vector}}(n)$	$\bar{t}_{ARS_{RBT}}(n)$	$\bar{t}_{ARS_{traverse\_tree}}(n)$
20	2228	2174	2187	2297
22	4134	4021	4004	4255
24	7243	6976	7000	7440
26	12 258	11 893	11 957	12 827
28	20 917	20 121	20 155	22 440
30	35 197	33 841	33 828	39 710
32	58 333	56 012	56 005	70 502
34	96 740	93 028	92 799	130 210
36	159 891	154 136	153 307	255 872
38	261 227	252 851	250 524	515 642
40	424 082	413 123	406 551	1 120 809
42	680 455	665 432	651 617	—
43	872 727	857 094	835 382	—

Таблица 3

Среднее затраченное время (в мкс) для ARS и $ARS_{matrices\_in\_leaves}$

n	$\bar{t}_{ARS}(n)$	$\bar{t}_{ARS_{matrices\_in\_leaves}}(n)$
20	2228	904
21	2993	1174
22	4134	1504
23	5533	1958
24	7243	2549
25	9534	3240
26	12 258	4053
27	16 266	5159
28	20 917	6432
29	27 167	8060
30	35 197	10 163
31	45 411	12 731
32	58 333	15 818
33	74 381	19 559
34	96 740	24 805
35	123 770	30 974
36	159 891	39 161
37	204 587	48 997
38	261 227	61 323
39	333 328	76 778
40	424 082	96 085
41	537 207	119 288
42	680 455	148 712
43	872 727	187 109

результат для всей рассматриваемой выборки, выбор между остальными структурами данных не оказывает существенного влияния на время работы программной реализации алгоритма. При этом упорядоченный список работает лучше самобалансирующегося двоичного дерева до $n = 29$, а начиная с $n = 30$ — самобалансирующееся дерево показывает лучший результат. Таким образом, можно рекомендовать использовать до $n = 29$ двоичную кучу, а начиная с $n = 30$ — самобалансирующиеся деревья. Это свидетельствует о том, что потребность в операции вставки элемента в очередь с приоритетом растет с размерностью задачи, а сложность этой операции у красно-черного дерева в худшем случае минимальная по сравнению с другими рассматриваемыми структурами данных.

В рамках данного предварительного экспериментального исследования, относительно хранения матриц в вершинах поискового дерева решений, мы показываем сравнение времени работы метода ветвей и границ без использования дополнительной памяти ($\bar{t}_{ARS}$) и с хранением усеченных матриц в листьях дерева решений ($\bar{t}_{ARS_{matrices\_in\_leaves}}$). Результаты предварительного экспериментального исследования представлены в табл. 3.

Экспериментальные данные показали, что хранение усеченных матриц стоимостей в листьях поискового дерева решений существенно уменьшает время расчета. Уже начиная с 43 городов, прирост в производительности по сравнению с классической реализацией составляет более чем 2 раза. Вместе с тем объем потребляемой памяти растет экспоненциально с ростом числа городов.

Заключение

В статье исследованы структуры данных, обеспечивающие эффективные по времени операции с листьями поискового дерева решений при решении асимметричной задачи коммивояжера методом ветвей и границ. Предложены различные подходы к хранению и обработке матриц, соответствующих листьям поискового дерева. Приведены предварительные экспериментальные результаты о временной эффективности соответствующих программных реализаций. Полученные результаты позволили предварительно рекомендовать в качестве решения упорядоченный список до $n = 29$ и самобалансирующееся двоичное дерево для бóльших размерностей для обслуживания поискового дерева решений. Что касается хранения матриц в вершинах поискового дерева решений, то предварительные результаты показали, что

хранение усеченных матриц может существенно сократить время точного решения задачи коммивояжера. На основе полученных результатов можно предварительно прогнозировать размерности задач ATSP, допускающие приемлемое время получения точного решения.

Возможные дальнейшие исследования будут направлены на проведение масштабных экспериментальных исследований в целях вероятностного прогнозирования временной эффективности точного решения индивидуальных асимметричных задач коммивояжера.

Список литературы

1. **Little J. D. C., Murty K. G., Sweeney D. W., Karel C.** An algorithm for the traveling salesman problem // Operations Research. 1963. Vol. 11. P. 972—989.

2. **Ульянов М. В., Фомичев М. И., Головешкин В. А., Жукова Г. Н.** Оценка параметров распределения логарифма сложности задачи коммивояжера // Современные информационные технологии и ИТ-образование. 2017. Т. 13, № 1. С. 19—24.

3. **Головешкин В. А., Жукова Г. Н., Ульянов М. В., Фомичев М. И.** Использование квантильных коэффициентов асимметрии и эксцесса для оценки сложности решения задачи коммивояжера // International Journal of Open Information Technologies. 2016. Т. 4, № 12. С. 7—12.

4. **Гудман С., Хидетниеми С.** Введение в разработку и анализ алгоритмов. М.: Мир, 1981. 368 с.

5. **Сигал И. Х., Иванова А. П.** Введение в прикладное дискретное программирование: модели и вычислительные алгоритмы. М.: ФИЗМАТЛИТ, 2007. 304 с.

6. **Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.** Алгоритмы: построение и анализ: Пер. с англ. М.: Издательский дом "Вильямс", 2005. 1296 с.

7. **Вирт Н.** Алгоритмы и структуры данных. Новая версия для Оберона / Пер. с англ. Ткачев Ф. В. М.: ДМК Пресс, 2010. 272 с.

M. V. Ulyanov, Leading Researcher, V. A. Trapeznikov Institute of Control Sciences of the Russian Academy of Sciences, M. V. Lomonosov Moscow State University
M. I. Fomichev, Graduate Student, National Research University Higher School of Economics

Approaches to Design Search Decision Tree in the Branch and Bound Method for the Asymmetric Traveling Salesman Problem

Run time of the Branch and Bound method for solving asymmetric Traveling Salesman Problem can be reduced by using the most appropriate data structure providing time-efficient operations with leaves of the search decision tree and additional memory for storing truncated matrices in leaves of search decision tree. In additional, different approaches for storing and handling matrices, which correspond leaves of search decision tree, can be presented. Such research should be based on the features of operations with search decision tree and matrices, generated by the specificity of the Branch and Bound method. Different approaches of storing and accessing elements of search decision tree and ways to store matrices in leaves of such tree are presented at this paper.

Keywords: *branch and bound method, asymmetric Traveling Salesman Problem, search decision tree, data structure, truncated matrices structure*

DOI: 10.17587/it.24.698-704

References

1. **Little J. D. C., Murty K. G., Sweeney D. W., Karel C.** An algorithm for the traveling salesman problem, *Operations Research*, 1963, vol. 11, pp. 972—989.

2. **Ulyanov M. V., Fomichev M. I., Goloveshkin V. A., Zhukova G. N.** Ocenka parametrov raspredeleniya logarifma slozhnosti zadachi kommivoyazhora (Parameters estimation of log of complexity for the Traveling Salesman Problem), *Sovremennie informacionnie texnologii i IT-obrazovanie*, 2017, vol. 13, no. 1. pp. 19—24 (in Russian).

3. **Goloveshkin V. A., Zhukova G. N., Ulyanov M. V., Fomichev M. I.** Ispolzovanie kvantilnix koefficientov asimmetrii i ekscessa dlya ocenki slozhnosti resheniya zadachi kommivoyazhora (The use of quantile coefficients of asymmetry and kurtosis for estimating the complexity of solving the Traveling Salesman Problem), *International Journal of Open Information Technologies*, 2016, vol. 4, no. 12, pp. 7—12.

4. **Goodman S., Hidetniemi S.** *Vedenie v razrabotku i analiz algoritmov (*Development and analysis of algorithms), Moscow, Mir, 1981, pp. 368 (in Russian).

5. **Sigal I. H., Ivanova A. P.** *Vedenie v prikladnoe discretnoe programmirovanie: modeli i vichislitelnie algoritmi (*Introduction to Applied Discrete Programming: Models and Computational Algorithms), Moscow, FIZMATLIT, 2007, pp. 304 (in Russian).

6. **Cormen T. H., Leiserson C. E., Rivest R. L., Stein C.** Introduction to Algorithm, Cambridge, MA, MIT Press, 2009.

7. **Wirth N.** Algorithms + Data Structures = Programs, Upper Saddle River, NJ, Prentice Hall, 1976.