ISSN 2079-3316 **ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ** **т. 11**, № 4(47), с. 3–16

ББК 22.181.19:22.181.2
ГРНТИ 28.25.23
УДК 004.832.25:519.712.45

В. В. Бурховецкий

Оптимизация и распараллеливание упрощенного алгоритма Балаша-Кристофидеса для задачи коммивояжера

Аннотация. В работе описывается точный параллельный алгоритм для задачи коммивояжера, основанный на упрощенном алгоритме Балаша-Кристофидеса, его оптимизация и увеличение эффективности распараллеливания. За счет нового метода передачи заданий между параллельными потоками алгоритм способен решать задачи с 3000 вершинами (со случайными весами дуг), в среднем, за минуту, а задачи с 10000 вершинами — за 50 минут. Возможность решать задачи с более чем 3000 вершинами появилась благодаря проведенной автором оптимизации расхода памяти.

Ключевые слова и фразы: метод ветвей и границ, параллельные вычисления, задача коммивояжера, обход дерева, оптимизация расхода памяти.

Введение

Задача коммивояжера является NP-трудной [1], т. е. на данный момент не существует точных алгоритмов полиномиальной сложности для ее решения. Этой задаче и алгоритмам для нее посвящено много статей, например [2], [3], [4]. Алгоритм Балаша-Кристофидеса [5] является одним из лучших точных методов ее решения. Он использует венгерский алгоритм [6] для решения вспомогательной задачи о назначениях. Похожий прием, который эвристически решает вспомогательную задачу, двойственную к задаче о назначениях, рассмотрен в [7]. В 1979 г. алгоритм Балаша-Кристофидеса выполнялся на компьютере CDC-7600 на случайных графах размерности 325 с целыми случайными весами дуг в диапазоне от 0 до 1000 менее чем за 2 минуты [5], что в то время являлось впечатляющим результатом.

Особенностью компьютеров того времени было то, что вычислительные операции были значительно медленнее операций над памятью. Сейчас же ситуация противоположная и одной из наиболее эффективных оптимизаций является уменьшение количества обращений к памяти [8]. Такие оптимизации дают огромный выигрыш многим алгоритмам и из других областей, например [9], [10].

Несколько применений задачи коммивояжера описано в [2]. К ним стоит добавить более современные задачи — развозка товаров покупателям интернет-магазина и сборка генома [11].

В данной статье представлен новый метод передачи заданий между параллельными потоками (см. раздел 5), за счет которого была увеличена эффективность распараллеливания упрощенного алгоритма Балаша-Кристофидеса, описанного в разделе 2, а также оптимизация расхода памяти, благодаря которой появилась возможность решать задачи с более чем 3000 вершинами (см. раздел 4). В разделе 6 представлено сравнение получившегося алгоритма с Concorde[1] — комплексом программного обеспечения, лидирующем по скорости решения симметричных задач коммивояжера (асимметричные задачи могут быть преобразованы в симметричные алгоритмом Йонкера-Волгенанта [12]).

Предыдущая версия данного алгоритма была представлена на конференции SECR'17 [13].

1. Постановка задачи коммивояжера

Приведем некоторые определения теории графов, необходимые для постановки задачи коммивояжера:

Определение 1. ***Гамильтонов цикл*** — *простой цикл, содержащий все вершины графа.*

Определение 2. ***Минимальный гамильтонов цикл*** — *гамильтонов цикл, сумма стоимостей дуг которого минимальна на данном графе.*

Теперь можно сформулировать задачу коммивояжера:

[1] http://www.math.uwaterloo.ca/tsp/concorde.html

Определение 3. ***Задача коммивояжера*** — *задача нахождения минимального гамильтонова цикла в полном взвешенном ориентированном графе, стоимости дуг которого неотрицательны.*

2. Упрощенный алгоритм Балаша-Кристофидеса

Алгоритм Балаша-Кристофидеса [5] — точный алгоритм решения задачи коммивояжера, основанный на методе ветвей и границ («точный» означает, что он всегда находит оптимальное решение). В данной работе используется описанная ниже упрощенная версия алгоритма (из нее исключены некоторые оптимизации).

Сформулируем несколько утверждений, на которых основаны алгоритм Балаша-Кристофидеса и используемый в данной работе алгоритм:

Лемма 2.1. *Если из всех элементов произвольной строки или столбца матрицы весов вычесть одну и ту же константу C_0, то стоимость минимального гамильтонова цикла уменьшится на C_0, а сам цикл (т. е. порядок обхода вершин) при этом не изменится* [5, 14].

Лемма 2.2. *Если трактовать матрицу весов дуг $n \times n$ для задачи коммивояжера как двудольный граф с $2n$ вершинами и решить на нем задачу о назначении, то ее решение на исходном графе будет выглядеть как множество непересекающихся простых циклов. Эти циклы будут покрывать все вершины графа. Сумма весов дуг всех циклов будет меньше или равна стоимости минимального гамильтонова цикла. Если же при решении задачи о назначении получился один простой цикл, то он — минимальный гамильтонов цикл на данном графе* [5].

Теперь приведем алгоритм:

(1) На вход поступает граф, представленный в виде матрицы весов дуг. В качестве начального значения рекорда берем любой гамильтонов цикл (например, $1 \to 2 \to \cdots \to n \to 1$).

(2) Матрица весов дуг трактуется как матрица задачи о назначениях, которая решается с помощью модифицированного венгерского алгоритма. Двойственные переменные $u_i,\ i = 1, \ldots, n$, и $v_j,\ j = 1, \ldots, n$, полученные в ходе решения, вычитаются из соответствующих строк и столбцов (из i-й строки вычитается u_i, а из j-го столбца — v_j). По лемме 2.1 это не меняет минимального

гамильтонова цикла на графе. Стоимость оптимального решения (сумма всех u_i и v_j) используется в качестве нижней оценки на текущем узле в соответствии с леммой 2.2.

(3) Если решение задачи о назначении — один простой цикл, то он является оптимальным на данной ветви дерева вариантов. Если его стоимость меньше рекорда, то меняем рекорд.

(4) Если же решение задачи о назначении на исходном графе выглядит как множество непересекающихся простых циклов, то из них выбирается цикл с минимальным количеством дуг k. По нему происходит ветвление: берем любую вершину данного цикла и рассматриваем следующие варианты:

(a) Гамильтонов цикл на данной ветви не пойдет по дугам выбранного цикла. Для этого удаляем дугу, выходящую из выбранной вершины.

(b) Гамильтонов цикл пойдет по $s = 1, 2, \ldots, k-1$ дугам выбранного цикла. Для этого фиксируем s дуг выбранного цикла (как происходит фиксация см. ниже) и удаляем $s+1$-ю дугу цикла.

Если какая-либо дуга уже была зафиксирована, то она не удаляется, а потомок, в котором требуется ее удалить, отбрасывается. Все полученные потомки помещаются в очередь в порядке возрастания их нижних оценок.

(5) Далее из очереди извлекается узел дерева вариантов с наименьшей нижней оценкой и алгоритм переходит на шаг 2. Если очередь пуста, то алгоритм завершается.

Модификация венгерского алгоритма заключается в том, что решение задачи о назначении в родительском узле (за исключением удаленных дуг) служит начальным решением для венгерского (итерационного) алгоритма [6] в потомках.

Под фиксированием дуги подразумевается наложение требования, чтобы все гамильтоновы циклы на данной ветви проходили через неё. Для этого удаляются все дуги, входящие в конечную вершину данной дуги и выходящие из начальной вершины данной дуги (т. е. удаляются все элементы матрицы из столбца и строки, в которой находится фиксируемая дуга, кроме самой этой дуги).

Описанный выше алгоритм является основной частью алгоритма [5]. Он и понимается в данной статье под словами «упрощенный алгоритм Балаша-Кристофидеса». Весь алгоритм Балаша-Кристофидеса [5] включает еще дополнительные эвристические оценки величины

минимального гамильтонова цикла в узлах дерева вариантов при методе ветвей и границ.

Корректность разработанной программной реализации последовательного и параллельного алгоритмов проверялась на асимметричных матрицах из TSPLIB[2], для которых известны оптимальные решения. Также, hd 0a случайно сгенерированных матрицах, описанных в разделе 3, проверялось, что все представленные реализации получили одинаковые результаты.

3. Условия проведения численных экспериментов

Все представленные в данной работе результаты были получены в следующих условиях:

- Процессор: Intel Core i5-6600 CPU @ 3.30GHz
  - 4 ядра
  - без hyperthreading
  - L3 кэш: 6 МБ (общий)
  - L2 кэш: 256 КБ (у каждого ядра свой)
  - L1 кэш: 32 КБ для инструкций, 32 КБ для данных (у каждого ядра свой)
- Операционная система: Debian 9 (Linux)
- Язык программирования: C++
- Компилятор: GCC версии 6.3.0 с флагами «-Ofast» и «-march= native»
- Библиотека для распараллеливания: ParallelTree

Примечание: с момента проведения измерений времени работы, представленных на SECR'17 [**13**], поменялся код ядра linux и микрокод процессоров Intel, в связи с обнаруженными в нем уязвимостями «spectre» и «meltdown». Поэтому все измерения проведены заново и отличаются от представленных в [**13**].

Все измерения времени работы и используемой памяти проводились на матрицах со случайными целыми четырехбайтными числами диапазона от 0 до 1000000. Их суммы (например, рекорд) хранятся в восьмибайтных числах. Всего было сгенерировано по 5 матриц каждой из размерностей.

[2] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

4. Уменьшение расхода памяти

Основной проблемой более ранней реализации алгоритма, представленной в [13], был чрезмерный расход памяти, который возникал ввиду того, что на каждом узле дерева вариантов создавалась новая матрица. Из-за этого раннее не получалось решать задачи с матрицами весов размерности выше 3000. Расход памяти удалось значительно уменьшить за счет повторного использования строк матриц. Для этого матрицы весов были представлены в виде структуры, которая содержит:

(1) Массив указателей на строки матрицы.

(2) Каждая строка содержит счетчик указателей, ссылающихся на нее. Как только счетчик становится равным 0, строка удаляется.

(3) Массив булевых переменных для запоминания того, какие столбцы были удалены. Т. е. при необходимости удалить столбец соответствующий ему флаг в этом массиве устанавливается в true, а сам столбец не удаляется.

Эта структура поддерживает две операции:

(1) Копирование с удалением одного элемента (i, j). Создается копия массива указателей и копия строки i, соответствующему указателю присваивается адрес копии (остальные строки не копируются). Элемент j удаляется из копии строки i.

(2) Копирование с удалением всех элементов строки i и всех элементов столбца j за исключением элемента на их пересечении (i, j). Создается копия массива указателей, копия массива удаленных столбцов и новая строка, которая содержит только элемент (i, j). Указателю на строку i присваивается адрес новой строки (остальные указатели не меняются). Элементу j массива удаленных столбцов присваивается значение true.

Этих операций достаточно для обеспечения работы упрощенного алгоритма Балаша-Кристофидеса. Ввиду того, что на одну и ту же строку ссылаются много структур в нескольких потоках, менять содержимое строк нельзя. Чтобы изменить содержимое строки, необходимо ее скопировать и внести изменения в копию сразу после копирования, пока на нее ссылается только один указатель. Т. к. различные потоки получают доступ к одним и тем же строкам только на чтение, то нет необходимости синхронизировать доступ к ним (с помощью мютексов, критических секций и т. п.).

Отметим, что во всем алгоритме Балаша-Кристофидеса данная структура не дала бы существенного уменьшения расхода памяти, т. к. в нем производится больше операций над матрицей, которые за раз меняют элементы почти во всех строках [5].

Использование этой структуры позволило уменьшить расход памяти в сотни раз, как видно в таблице 1. Ввиду значительного

Таблица 1. Расход памяти параллельным алгоритмом, МБ

Размер матриц	Максимально используемая память до внедрения структуры	после
1000	1 668	25
1500	5 235	50
2000	11 351	82
2500	9 575	97
3000	22 394	149
5000	$> 2^{15}$	361
7000	$> 2^{15}$	1 431
10000	$> 2^{15}$	1 530

уменьшения расхода памяти удалось применить алгоритм к задачам намного больших размерностей.

5. Увеличение эффективности распараллеливания

Еще одной проблемой реализации, представленной в [13], был не очень эффективный способ передачи заданий от потока к потоку. За передачу заданий отвечает созданная автором библиотека для распараллеливания обхода дерева «ParallelTree», которая, в свою очередь, использовала OpenMP. «ParallelTree» позволяет распараллеливать обход дерева без написания параллельного кода, за исключением синхронизации доступа к рекорду. Передача заданий осуществлялась через глобальный контейнер, который мог содержать максимум один узел дерева вариантов и доступ к которому осуществлялся по двум мютексам. Если поток освобождался, то он забирал узел из глобального контейнера, либо ждал пока в контейнер кто-то положит узел. Занятые потоки на каждой итерации проверяли, является ли контейнер пустым, и если да, то клали в него один из своих узлов дерева вариантов.

Однако потоки освобождаются достаточно редко (количество передач от потока к потоку составляет менее 10% от общего числа узлов дерева вариантов), поэтому синхронизированная проверка глобального контейнера на пустоту на каждой итерации является излишней.

Сначала, для решения данной проблемы, в библиотеке «ParallelTree» OpenMP был заменен на pthread. OpenMP — высокоуровневый и не дает программисту достаточно инструментов для контроля над ходом выполнения параллельной программы (например, в OpenMP нет условных переменных и нет read-write lock). Более низкоуровневый pthread предоставляет нужные для решения проблемы инструменты, хотя написание и отладка параллельных программ на нем и занимает больше времени. Время работы алгоритма после перехода на pthread и до изменения метода передачи заданий представлено в таблице 2 (это время было измерено для того, чтобы убедиться, что ускорение получено за счет нового метода передачи заданий, а не за счет перехода на pthread).

Далее:

(*1*) В глобальный контейнер была добавлена возможность содержать сколько угодно узлов, а не только один.

(*2*) Глобальный контейнер теперь использует один мютекс и одну условную переменную, а не два мютекса.

(*3*) Каждому потоку было присвоено «состояние» — однобайтная переменная state (у каждого потока своя), которая принимает одно из двух возможных значений: Continue (нужно продолжать работу, ничего класть в глобальный контейнер не нужно) и Push (нужно положить один из своих узлов в глобальный контейнер). Доступ к переменным state осуществляется без синхронизации (без мютексов, без атомарных операций и т. п.). Поскольку переменные state занимают один байт, то на процессорах Intel они не могут оказаться в некорректном состоянии в результате гонки данных. Однако случай, когда state находится в некорректном состоянии, все равно предусмотрен — любое значение, не равное Continue, считается равным Push.

(*4*) Когда поток освобождается, он проверяет глобальный контейнер. Если тот не пуст, то поток забирает из него один узел, а иначе поток изменяет state всех остальных потоков на Push (без синхронизации) и засыпает, ожидая пробуждения по условной переменной.

(5) Занятые потоки на каждой итерации проверяют свои переменные state. Если state заданного потока равно Continue, то поток продолжает работу, а иначе кладет в глобальный контейнер один из своих узлов, изменяет свое состояние на Continue (без синхронизации) и пробуждает один из ожидающих потоков. Т. е. в состоянии Continue, в котором потоки находятся более 90% времени, доступа к глобальному контейнеру не осуществляется вообще.

(6) Доступ к рекорду ранее осуществлялся по самописному read-write lock (который был реализован на двух мютексах), т. к. в OpenMP нет read-write lock, а теперь доступ осуществляется по read-write lock из pthread.

(7) Доступ к счетчикам указателей на строки матриц, описанным в разделе 3, теперь осуществляется с помощью атомарных операций с более слабой моделью памяти, чем ранее, т. к. OpenMP не дает возможности напрямую задавать атомарную модель памяти.

Время работы программы с новым методом передачи узлов представлено в таблице 2.

Таблица 2. Сравнение различных реализаций алгоритма по времени вычислений и ускорению

Метод передачи заданий	Размер матриц 3000	5000	7000	10000
Последовательный	01м 38с	12м 51с	28м 51с	1ч 14м 06с
Старый (OpenMP)	57с (×**1.72**)	09м 45с (×**1.32**)	17м 05с (×**1.69**)	53м 59с (×**1.37**)
Старый (pthread)	56с (×**1.73**)	08м 58с (×**1.43**)	16м 30с (×**1.75**)	1ч 02м 35с (×**1.18**)
Новый (pthread)	53с (×**1.85**)	05м 54с (×**2.17**)	17м 02с (×**1.69**)	47м 32с (×**1.56**)

6. Сравнение с Concorde

Concorde[3] — комплекс программного обеспечения с открытым исходным кодом для решения симметричной задачи коммивояжера,

[3] http://www.math.uwaterloo.ca/tsp/concorde.html

который считается лидирующем по скорости решения симметречных задач [15], [16], [17].

Concorde решает только симметричные задачи коммивояжера, причем он не может решать задачи с числом вершин больше 600 (с весами дуг от 0 до 1000000) ввиду целочисленного переполнения. Поэтому были использованы асимметричные задачи с меньшим числом вершин, которые преобразовывались в симметричные задачи с удвоенным числом вершин с помощью алгоритма Йонкера-Волгенанта [12]. Сравнение данного алгоритма с Concorde представлено в таблице 3.

Таблица 3. Сравнение нового параллельного метода с Concorde

Метод передачи заданий	Размер матриц				
	100	200	300	400	500
Concorde	0.912	1.704	2.046	14.432	10.866
Новый (pthread)	0.004	0.018	0.066	0.343	0.363

Заключение

Новый метод передачи заданий повысил эффективность распараллеливания, но не позволил получить ускорение, близкое к четырехкратному, на 4 ядрах. Вероятно, ввиду большого размера матриц и совместного использования памяти несколькими потоками, возникает большое количество кэш-промахов, которые значительно уменьшают эффективность распараллеливания.

В дальнейшем автор планирует уменьшить число кэш-промахов, исследовать возможность дальнейшего увеличения эффективности распараллеливания за счет уменьшения числа синхронизированных операций, и применить данный алгоритм к задаче сборки генома путем сведения ее к задаче коммивояжера.

Список литературы

[1] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, USA, 1979, ISBN 0716710447. ↑3

[2] И. И. Меламед, С. И. Сергеев, И. Х. Сигал. «Задача коммивояжера.

Вопросы теории», *Автоматика и телемеханика*, 1989, №9, с. 3–33. ↑3,4

[3] И. И. Меламед, С. И. Сергеев, И. Х. Сигал. «Задача коммивояжера. Точные методы», *Автоматика и телемеханика*, 1989, №10, с. 3–29. ↑3

[4] И. И. Меламед, С. И. Сергеев, И. Х. Сигал. «Задача коммивояжера. Приближенные алгоритмы», *Автоматика и телемеханика*, 1989, №11, с. 3–26. ↑3

[5] E. Balas, N. Christofides. "A restricted Lagrangean approach to the traveling salesman problem", *Mathematical Programming*, **21**:1 (1981), pp. 19–46. ↑3,5,6,9

[6] G. B. Dantzig. "Optimal assignment and other distribution problems", *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, USA, 1998, ISBN 0691059136, pp. 316–334. ↑3,6

[7] Ю. Л. Костюк. «Эффективная реализация алгоритма решения задачи коммивояжера методом ветвей и границ», *Прикладная дискретная математика*, **20**:2 (2013), с. 78–90. ↑3

[8] National Research Council. *Getting Up to Speed: The Future of Supercomputing*, eds. S. L. Graham, M. Snir, C. A. Patterson, The National Academies Press, Washington, DC, 2005, ISBN 978-0-309-09502-0, 289 pp. ↑4

[9] V. Levchenko, A. Perepelkina, A. Zakirov. "DiamondTorre algorithm for high-performance wave modeling", *Computation*, **4**:3 (2016), pp. 29. ↑4

[10] S. G. Ammaev, L. R. Gervich, B. Y. Steinberg. "Combining parallelization with overlaps and optimization of cache memory usage", *Parallel Computing Technologies*, PaCT 2017, Lecture Notes in Computer Science, vol. **10421**, ed. V. Malyshkin, Springer International Publishing, 2017, pp. 257–264. ↑4

[11] S. El-Metwally, O. M. Ouda, M. Helmy, *Next Generation Sequencing Technologies and Challenges in Sequence Assembly*, SpringerBriefs in Systems Biology, vol. **7**, Springer, 2014, ISBN 978-1-4939-0715-1, pp. 17–19, 84–85. ↑4

[12] R. Jonker, T. Volgenant. "Transforming asymmetric into symmetric traveling salesman problems", *Operations Research Letters*, **2**:4 (1983), pp. 161–163. ↑4,12

[13] V. Burkhovetskiy, B. Steinberg. "An exact parallel algorithm for traveling salesman problem", *Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia*, CEE-SECR'17, ACM, New York, NY, USA, 2017 (in Russian), 5 pp. ↑4,7,8,9

[14] M. Bellmore, G. L. Nemhauser. "The traveling salesman problem: a survey", *Operations Research*, **16**:3 (1968), pp. 538–558. ↑5

[15] Y. Kaempfer, L. Wolf. *Learning the multiple Traveling Salesmen Problem with permutation invariant pooling networks*, 2018. arXiv:1803.09621 ↑12

[16] A. Fischer, F. Fischer, G. Jäger, J. Keilwagen, P. Molitor, I. Grosse. “Exact algorithms and heuristics for the Quadratic Traveling Salesman Problem with an application in bioinformatics”, *Discrete Applied Mathematics*, **166** (2014), pp. 97–114. ↑12

[17] T. Fischer, T. Stützle, H. Hoos, P. Merz. “An analysis of the hardness of TSP instances for two high performance algorithms”, MIC2005: The Sixth Metaheuristics International Conference (Vienna, Austria, August 22–26, 2005), 2005, pp. 361–367. ↑12

Поступила в редакцию	11.04.2020
Переработана	06.08.2020
Опубликована	09.10.2020

Рекомендовал к публикации *проф. Н Н. Непейвода*

Пример ссылки на эту публикацию:

В. В. Бурховецкий. «Оптимизация и распараллеливание упрощенного алгоритма Балаша-Кристофидеса для задачи коммивояжера». *Программные системы: теория и приложения*, 2020, **11**:4(47), с. 3–16.

10.25209/2079-3316-2020-11-4-3-16

http://psta.psiras.ru/read/psta2020_4_3-16.pdf

Об авторе:

Виктор Витальевич Бурховецкий

Автор статей про эффективные алгоритмы для задачи коммивояжера и про распараллеливание алгоритмов на основе метода ветвей и границ. Аспирант Института математики, механики и компьютерных наук имени И. И. Воровича Южного федерального университета. Научный руководитель — Штейнберг Борис Яковлевич.

0000-0002-1292-0280

e-mail: buvictor95@inbox.ru

CSCSTI 28.25.23
UDC 004.832.25:519.712.45

Victor V. Burkhovetskiy. *Optimization and parallelization of simplified Balas' and Christofides' algorithm for the traveling salesman problem.*

Abstract. The paper describes an exact parallel algorithm for the traveling salesman problem based on simplified Balas' and Christofides' algorithm, its optimization, and improvements in its parallel efficiency. Due to the new method of passing tasks between parallel threads, the algorithm solves, on average, instances with 3000 nodes (with random edge weights) in 1 minute, and instances with 10000 nodes in 50 minutes. The algorithm can solve instances with more than 3000 nodes due to the memory usage optimization introduced in this paper.

Key words and phrases: branch-and-bound, parallel computing, traveling salesman problem, tree traversal, memory optimization.

2020 *Mathematics Subject Classification*: 97K30; 97N60, 97N80

References

[1] M. R. Garey, D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, USA, 1979, ISBN 0716710447. ↑3

[2] I. I. Melamed, S. I. Sergeyev, I. Kh. Sigal. "The traveling salesman problem. Issues in theory", *Automation and Remote Control*, **50**:9 (1989), pp. 1147–1173. ↑3,4

[3] I. I. Melamed, S. I. Sergeyev, I. Kh. Sigal. "The traveling salesman's problem. Exact methods", *Autom. Remote Control*, **50**:10 (1989), pp. 1303–1324. ↑3

[4] I. I. Melamed, S. I. Sergeyev, I. Kh. Sigal. "The traveling salesman problem. Approximate algorithms", *Autom. Remote Control*, **50**:11 (1989), pp. 1459–1479. ↑3

[5] E. Balas, N. Christofides. "A restricted Lagrangean approach to the traveling salesman problem", *Mathematical Programming*, **21**:1 (1981), pp. 19–46. ↑3,5,6,9

[6] G. B. Dantzig. "Optimal assignment and other distribution problems", *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, USA, 1998, ISBN 0691059136, pp. 316–334. ↑3,6

[7] Yu. L. Kostyuk. "Effective implementation of algorithm for solving the travelling salesman problem by branch-and-bound method", *Prikladnaya diskretnaya matematika*, **20**:2 (2013), pp. 78–90 (in Russian). ↑3

[8] National Research Council. *Getting Up to Speed: The Future of Supercomputing*, eds. S. L. Graham, M. Snir, C. A. Patterson, The National Academies Press, Washington, DC, 2005, ISBN 978-0-309-09502-0, 289 pp. ↑4

[9] V. Levchenko, A. Perepelkina, A. Zakirov. "DiamondTorre algorithm for high-performance wave modeling", *Computation*, **4**:3 (2016), pp. 29. ↑4

[10] S. G. Ammaev, L. R. Gervich, B. Y. Steinberg. "Combining parallelization with overlaps and optimization of cache memory usage", *Parallel Computing Technologies*, PaCT 2017, Lecture Notes in Computer Science, vol. **10421**, ed. V. Malyshkin, Springer International Publishing, 2017, pp. 257–264. ↑4

[11] S. El-Metwally, O. M. Ouda, M. Helmy, *Next Generation Sequencing Technologies and Challenges in Sequence Assembly*, SpringerBriefs in Systems Biology, vol. **7**, Springer, 2014, ISBN 978-1-4939-0715-1, pp. 17–19, 84–85. ↑4

[12] R. Jonker, T. Volgenant. "Transforming asymmetric into symmetric traveling salesman problems", *Operations Research Letters*, **2**:4 (1983), pp. 161–163. ↑4,12

[13] V. Burkhovetskiy, B. Steinberg. "An exact parallel algorithm for traveling salesman problem", *Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia*, CEE-SECR'17, ACM, New York, NY, USA, 2017 (in Russian), 5 pp. ↑4,7,8,9

[14] M. Bellmore, G. L. Nemhauser. "The traveling salesman problem: a survey", *Operations Research*, **16**:3 (1968), pp. 538–558.↑5

[15] Y. Kaempfer, L. Wolf. *Learning the multiple Traveling Salesmen Problem with permutation invariant pooling networks*, 2018. arXiv:1803.09621↑12

[16] A. Fischer, F. Fischer, G. Jäger, J. Keilwagen, P. Molitor, I. Grosse. "Exact algorithms and heuristics for the Quadratic Traveling Salesman Problem with an application in bioinformatics", *Discrete Applied Mathematics*, **166** (2014), pp. 97–114. ↑12

[17] T. Fischer, T. Stützle, H. Hoos, P. Merz. "An analysis of the hardness of TSP instances for two high performance algorithms", MIC2005: The Sixth Metaheuristics International Conference (Vienna, Austria, August 22–26, 2005), 2005, pp. 361–367. ↑12

Sample citation of this publication:

Victor V. Burkhovetskiy. "Optimization and parallelization of simplified Balas' and Christofides' algorithm for the traveling salesman problem". *Program Systems: Theory and Applications*, 2020, **11**:4(47), pp. 3–16. (*In Russian*).

10.25209/2079-3316-2020-11-4-3-16

http://psta.psiras.ru/read/psta2020_4_3-16.pdf