Моделирование и анализ информационных систем. Т. 23, № 4 (2016), с. 401–411
Modeling and Analysis of Information Systems. Vol. 23, No 4 (2016), pp. 401–411

DOI: 10.18255/1818-1015-2016-4-401-411

УДК 519.688: 519.85

Об оптимизации и распараллеливании алгоритма Литтла для решения задачи коммивояжера

Васильчиков В. В.

получена 31 мая 2016

Аннотация.

В данной работе рассматриваются способы ускорения решения NP-полной задачи коммивояжера. Классический алгоритм Литтла, относящийся к категории "методов ветвей и границ" , позволяет ее решать как для ориентированных, так и для неориентированных графов. Однако для неориентированных графов его работу можно ускорить за счет исключения рассмотрения фактически ранее рассмотренных вариантов. В работе предлагаются изменения, которые следует внести в ключевые операции алгоритма для ускорения его работы. Приводятся результаты численного эксперимента, показавшего значительное ускорение решения задачи с использованием усовершенствованного алгоритма. Другой ресурс для ускорения – это разработка параллельного алгоритма. Для задач подобного рода весьма сложно сразу разбить вычисления на достаточное количество сравнимых по трудоемкости подзадач. Параллелизм у них выявляется динамически во время вычислений. Для таких задач разумным представляется использование рекурсивно-параллельной организации вычислений. В нашем случае хорошим выбором оказалась разработанная автором библиотека RPM_ParLib, позволяющая создавать эффективные параллельные программы для вычислений на локальной сети в среде .NET Framework на любом поддерживаемом ею языке программирования. Мы при разработке программы использовали язык C#. Были написаны параллельные программы для реализации как исходного, так и модифицированного алгоритмов, проведено их сравнение. Эксперименты проводились для графов с количеством вершин до 45 с количеством компьютеров в сети до 16. Дополнительно исследовалось ускорение, которого можно достичь за счет распараллеливания базового алгоритма Литтла для ориентированных графов. Результаты этих серий экспериментов также приводятся в работе.

Ключевые слова: задача коммивояжера, алгоритм Литтла, параллельный алгоритм, рекурсия, .NET

Для цитирования: Васильчиков В. В., "Об оптимизации и распараллеливании алгоритма Литтла для решения задачи коммивояжера", *Моделирование и анализ информационных систем*, **23**:4 (2016), 401–411.

Об авторах:
Васильчиков Владимир Васильевич, orcid.org/0000-0001-7882-8906, канд. техн. наук, зав. кафедрой вычислительных и программных систем,
Ярославский государственный университет им. П.Г. Демидова,
ул. Советская, 14, г. Ярославль, 150000 Россия, e-mail: vvv193@mail.ru

Благодарности:
Работа выполнена в рамках инициативной НИР ВИП-004 (номер госрегистрации АААА-А16-116070610022-6).

Введение

Задача коммивояжера является одной из самых известных задач дискретной оптимизации. Она относится к категории NP-полных задач, для которых не найдено алгоритмов точного решения за полиномиальное от размера задачи время [1]. Для нахождения точного решения некоторых задач такого рода достаточно эффективно могут применяться алгоритмы, относящиеся к категории "метод ветвей и границ". Метод впервые был предложен в [2] применительно к задаче целочисленного линейного программирования. Для решения задачи коммивояжера с успехом применяется алгоритм Литтла [3–6], который относится именно к этой категории методов. К сожалению, для него не существует приемлемой оценки трудоемкости, и время его работы очень сильно зависит от исходных данных.

Алгоритм Литтла работает и для ориентированных и для неориентированных графов. Однако в ходе экспериментов на компьютере было отмечено, что для случая симметричной матрицы расстояний (неориентированный граф) алгоритм выполняет повторное рассмотрение фактически уже исследованных ветвей вычислений. В работе предлагается модификация алгоритма, позволяющая устранить это явление и ускорить получение результата. Следует отметить, что и ранее разными авторами предлагались различные модификации базового алгоритма Литтла с целью ускорить его работу, в качестве примера можно назвать, скажем, работы [7, 8], однако наше предложение носит принципиально иной характер.

Другая возможность ускорить решение задачи – это распараллеливание. Поскольку количество вычислений и объем необходимой памяти при решении задачи коммивояжера с увеличением ее размера быстро возрастают, наличие возможности организации вычислений в параллельном режиме становится очень актуальным. Исследования в этом направлении могут быть основаны на различных парадигмах и инструментальных средствах параллельного программирования [9]. Автор же данной работы выбрал в качестве основы парадигму рекурсивно-параллельного (РП) программирования [10] и соответствующие библиотеки поддержки [11, 12].

В [10] изложены основные принципы организации рекурсивно-параллельных вычислений и описаны основные алгоритмы и механизмы поддержки этого стиля программирования. Библиотеки [11, 12] позволяют относительно легко создавать, отлаживать и эксплуатировать РП-приложения в среде .NET Framework. В [13] подробно описаны функциональные возможности упомянутых библиотек, а также процесс разработки и исследования РП-алгоритма решения NP-полной задачи о клике.

1. Задача коммивояжера

Напомним формулировку задачи.

Задано конечное множество $C = \{c_0, c_1, ...c_{n-1}\}$ "городов" и для каждой пары $c_i, c_j \in C$ "расстояние" между ними $d_{ij} = d(c_i, c_j) \in Z^+$ (здесь Z^+ означает положительные целые числа). Матрица расстояний $D = \|d_{ij}\|$ не обязательно должна быть симметричной, также мы не требуем выполнения неравенства треугольника, то есть рассматриваем задачу в самом общем виде.

Коммивояжер, выехав из города c_0, должен объехать все остальные города, побывав в каждом по одному разу, и вернуться в город c_0. Обозначим соответствующий

циклический маршрут $r = \{0, i_1, i_2, ...i_{n-1}, 0\}$, здесь мы без ограничения общности положили $i_0 = i_n = 0$. Очевидно, длина пути определяется как

$$l(r) = \sum_{k=0}^{n-1} d_{i_k j_{k+1}}.$$

Пусть R – множество всех таких маршрутов. Требуется найти цикл $r_0 \in R$, такой что

$$l(r_0) = \min_{r \in R} l(r).$$

Очевидно, что количество всех допустимых маршрутов равно $(n-1)!$ и полный перебор крайне неэффективен.

2. Алгоритм Литтла

Напомним, как организованы вычисления при использовании базовой реализации алгоритма Литтла.

Выберем произвольное положительное значение ∞, заведомо большее любого из d_{ij} . Мы будем присваивать его элементам матрицы D, которые соответствуют дугам, не подлежащим включению в искомый маршрут. Сразу же пропишем это значение во всех элементах главной диагонали, чтобы исключить наличие петель. Также присвоим переменной *Record*, предназначенной для хранения длины кратчайшего на данный момент цикла, заведомо неконкурентоспособное значение. Далее алгоритм носит рекурсивный характер, его основной шаг выглядит следующим образом:

1. Осуществляем редукцию матрицы D по строкам путем вычитания из всех элементов каждой строки минимального в строке значения. Очевидно, что кратчайший маршрут после такой операции так и останется кратчайшим.

2. Аналогичным образом осуществляем приведение матрицы D по столбцам.

3. Сумму вычтенных на предыдущих двух этапах минимальных значений назовем константой редукции. Она имеет смысл нижней оценки длины кратчайшего маршрута. Если бы теперь можно было построить гамильтонов цикл только из ребер нулевой длины, это и был бы оптимальный вариант.

4. Для каждого ребра нулевой длины в модифицированной матрице D получим так называемую "оценку нуля" v_{ij}. Для элемента в позиции (i, j) она равна сумме минимальных элементов в i-й строке и j-м столбце (не считая сам этот нуль).

5. Выбираем нулевой элемент с максимальной оценкой, пусть это элемент, расположенный в i-й строке и j-м столбце. Теперь все допустимые маршруты можно разбить на два класса: содержащие ребро (i, j) и не содержащие. Для последних нижняя оценка длины оптимального маршрута увеличивается на величину v_{ij} .

6. Для продолжения вычислений организуем рекурсивное разделение процесса. Для маршрутов первого класса (с ребром (i, j)) включаем его в искомый маршрут, убедившись, что при этом не образуется цикла, проходящего менее чем через n вершин, далее исключаем из матрицы D i-ю строку и j-й столбец (или заполняем их значениями ∞), затем переходим к пункту 1. Для маршрутов без ребра (i, j) просто полагаем значение d_{ij} равным ∞ и переходим к пункту 1.

Рекурсивный процесс завершается в следующих случаях:

- В маршрут включено $n - 1$ ребро, в этом случае маршрут очевидным образом достраивается, и если его длина меньше, чем значение $Record$, маршрут запоминается как лучший на текущий момент.
- Если полученная на этапе 5 оценка длины маршрута больше или равна $Record$, ветвь считается неперспективной, и вычисления на ней завершаются немедленно.

Отметим, что для эффективной организации работы алгоритма необходимо хорошо продумать способ хранения информации, относящейся к каждой ветви вычислений. Ограниченность объема оперативной памяти сразу вынуждает нас несколько отступить от изложенной выше последовательности действий. На этапе 5 наиболее выгодным было бы выбирать для продолжения вычислений нулевой элемент с глобально максимальной оценкой, так, чтобы вычисления продолжались во всех возможных направлениях. Однако рост количества незаконченных ветвей носит в этом случае экспоненциальный характер, причем для каждой требуется хранить свой вариант редуцированной матрицы, что для программной реализации неприемлемо. Поэтому, когда мы говорим о выборе нулевого элемента с максимальной оценкой, мы имеем в виду максимум *для текущей ветви*.

Следует также отметить необходимость эффективной организации проверки на отсутствие коротких циклов, которую требуется выполнять на каждой итерации. При наличии подходящей структуры данных это можно сделать за $O(n)$ операций.

Насколько автору известно, в настоящее время не существует сколько-нибудь информативных оценок трудоемкости алгоритма Литтла – это все-таки перебор, хотя и хорошо оптимизированный. Скорость решения очень зависит от исходных данных, здесь мы имеем в виду не размер задачи, а матрицу расстояний. Для случайных графов с одинаковым количеством вершин и одинаковым распределением длин ребер, как показывает эксперимент, скорость получения результата может отличаться в десятки и даже сотни раз. На нее влияет и начальное значение $Record$, и даже порядок нумерации вершин, правда не так радикально, как расстояния между вершинами.

3. Модификация алгоритма Литтла для симметричной матрицы расстояний

Как было ранее отмечено, алгоритм работает и для ориентированных, и для неориентированных графов. Последний случай отличается только тем, что матрица рас-

стояний симметрична относительно главной диагонали. Кстати, в ходе экспериментов было отмечено, что для неориентированных графов алгоритм, как правило, работает значительно дольше. Это обусловлено многими причинами. Одна из них заключается в том, что, исключив на каком-то этапе из рассмотрения ребро (a, b), мы в дальнейшем все равно вынуждены рассматривать варианты с включением и исключением ребра (b, a). При продолжении ветви это делается не один раз. В итоге мы выполняем большое количество операций, которых можно было бы избежать, если сразу иметь в виду, что ребро можно пройти в обоих направлениях, а значит, исключив ребро (a, b), мы могли бы сразу же исключить и ребро (b, a).

Однако алгоритм для этого потребуется значительно изменить. Во-первых, редукцию нужно осуществлять так, чтобы текущая матрица расстояний постоянно оставалась симметричной. Во-вторых, требуется изменить способ вычисления "оценки нуля", чтобы учесть возможность прохождения ребра в разных направлениях. Наконец, модификацию матрицы при включении ребра в маршрут также следует производить иначе.

Ниже перечислены предлагаемые модификации. Их смысл легче понять, если представить, что мы строим не один цикл, а два: в прямом и обратном направлении.

- *Редукция.* Сразу после редукции i-й строки проделываем эту же операцию с i-м столбцом. Фактически достаточно просто скопировать значения элементов строки в элементы столбца, причем в константе редукции следует вычтенное значение учесть и для строки, и для столбца. Кроме того, если вершина еще не включена в строящийся цикл, в константе редукции она учитывается с весом 2, а если один раз уже включена, то с весом 1. Элементы, равные ∞, не редуцируются.

- *"Оценка нуля".* Для оценки нуля в позиции (i, j) возьмем сумму оценок для i-й и j-й вершин. Оценка i-й вершины вычисляется следующим образом. Если вершина i встречается в уже построенном участке цикла один раз, то оценкой является минимальный элемент в строке (или столбце) i, исключая сам нуль в позиции (i, j). Если вершина еще не встречалась, то оценка равна сумме *двух самых маленьких элементов* в строке. Два раза или более вершина при выполнении данной операции, разумеется, встретиться не может.

- *Изменение матрицы при включении или исключении ребра.* Если мы на данной ветви исключили ребро (i, j), то в матрице на позициях (i, j) и (j, i) записываем значение ∞. При добавлении ребра (i, j) выполняем следующие преобразования матрицы расстояний. Если вершина i включается в строящийся цикл повторно (в нее и вошли, и вышли), всем элементам i-й строки и i-го столбца присваивается значение ∞. Аналогично поступаем с вершиной j.

Для оценки качества предлагаемого варианта алгоритма был проведен численный эксперимент. Сразу отметим, что приводимые ниже оценки имеют чисто качественный характер. Во-первых, как было отмечено выше, количество итераций для получения результата непредсказуемо. Во-вторых, предложенная модификация радикально изменяет порядок рассмотрения вариантов построения цикла в базовом и оптимизированном алгоритме, что делает не вполне корректным их сравнение даже на одних и тех же исходных данных.

Однако несколько десятков проведенных экспериментов позволяют утверждать, что модифицированный алгоритм работает лучше почти всегда. Слово "почти" означает только то, что ни в одном из экспериментов он не работал дольше базового алгоритма, хотя непредсказуемость порядка выбора ребер для построения цикла не позволяет утверждать, что так будет всегда. При этом на достаточно больших примерах (35, 40 и более вершин) скорость работы модифицированного алгоритма зачастую была выше в десять и более раз.

Для сравнения скорости работы двух алгоритмов было следующим образом сгенерировано несколько десятков неориентированных графов. Вершины случайным образом равномерно размещались в квадрате размером 3000∗3000. Расстояния округлялись до ближайшего целого. Количество вершин бралось в пределах от 20 до 40. При количестве вершин, равном 45, базовый алгоритм обычно работал слишком долго, поэтому мы ограничились только этими значениями.

На этом этапе для сбора статистики использовался компьютер на базе двухъядерного процессора AMD с тактовой частотой 1.90 GHz и 6 GB оперативной памяти, работающий под управлением 64-разрядной ОС Windows 7. Ниже приводится таблица 1, содержащая результаты экспериментов на 10 графах, выбранных случайным образом.

№ п/п	Кол-во вершин	Базовый алгоритм		Оптимизированный алгоритм		Полученное ускорение	
		Итераций	Время (мс)	Итераций	Время (мс)	Итерации	Время
1.	35	*107 230*	13 418	*1 602*	262	***66.94***	***51.21***
2.	35	*847 722*	115 643	*14 727*	2 199	***57.56***	***52.59***
3.	35	*45 487 366*	6 488 281	*7 528 783*	899 248	***6.04***	***7.22***
4.	40	*31 832 080*	6 694 203	*242 609*	4 1706	***131.21***	***160.51***
5.	40	*11 242 224*	2 093 984	*1 362 560*	161 358	***8.25***	***12.98***
6.	40	*25 721 496*	4 736 154	*1 601 237*	256 790	***16.06***	***18.44***
7.	40	*1 144 205*	190 076	*492 012*	63 070	***2.33***	***3.01***
8.	40	*2 864 366*	480 548	*1 316 074*	189 683	***2.18***	***2.53***
9.	40	*673 835*	129 462	*9 320*	1 597	***72.30***	***81.07***
10.	40	*3 670 605*	649 980	*255 015*	34 266	***14.39***	***18.97***

Таблица 1. Сравнение скорости работы базового и оптимизированного алгоритмов

Несмотря на существенный разброс результатов сравнения двух алгоритмов, обусловленный вышеупомянутыми причинами, по-видимому, есть основания отдать предпочтение оптимизированному варианту.

4. Параллельный вариант алгоритма Литтла

Для таких трудоемких задач, как задача коммивояжера, совершенно естественным выглядит желание ускорить получение решения за счет распараллеливания алгоритма. Например, в [9] описан опыт распараллеливания алгоритма на суперкомпьютере MVS-15000 BM с применением такого подхода, как динамическое назначение подзадач на обработку. Задача коммивояжера решалась для графов с количеством вершин от 21 до 40, при этом на 12 процессорах достигалось ускорение до 6-7 раз.

Следует отметить, что способ разбиения данной задачи на ветви, допускающие параллельную обработку отнюдь не очевиден, а от него в основном и зависит достигнутое ускорение. За основу мы взяли рекурсивно-параллельный подход к построению алгоритма [10], язык программирования C#, а в качестве основного инструмента организации параллельных вычислений использовали библиотеку RPM_ParLib [11–13].

Такой подход естественным образом вытекает из рекурсивной организации алгоритма Литтла. На каждой ветви после выбора очередного ребра (a, b) ("нулевой элемент" с максимальной оценкой) вычисления могут быть разбиты на подзадачу, содержащую это ребро, и подзадачу, его не содержащую. Первую процессорный модуль (ПМ) оставляет себе, вторую оформляет как потенциально мигрирующий процесс, который при необходимости может быть выполнен на любом модуле. Из соображений эффективности такое рекурсивное дробление нецелесообразно продолжать до предела, поэтому мы заложили ограничение на уровень вложенности рекурсии и на количество уже включенных ребер. При превышении этих значений подзадача решается с использованием последовательного алгоритма. В наших экспериментах оба ограничения были равны 16, этого было достаточно, чтобы механизм динамической балансировки нагрузки, реализованный в библиотеке RPM_ParLib, мог достаточно равномерно распределить работу.

При передаче подзадачи на другой модуль достаточно передавать только накопленные для нее текущую оценку и векторы редукции, поскольку исходная матрица расстояний в самом начале работы рассылалась на все процессорные модули. Как только на одной из веток улучшался текущий рекорд, полученное значение немедленно рассылалось на все модули и использовалось ими в процессе вычислений для отсечения неперспективных ветвей.

		Вар.1	Вар.2	Вар.3	Вар.4	Вар.5	Вар.6
Последов. алгоритм	Итераций	*655 505*	*7 563 800*	*16 195 164*	*5 517 076*	*1 078 332*	*14 133 682*
	Время (с.)	*76*	*690*	*1343*	*527*	*127*	*1176*
1 ПМ	Время (с.)	80	722	1391	535	143	1282
	Ускорение	***0.94***	***0.96***	***0.97***	***0.98***	***0.89***	***0.92***
2 ПМ	Время (с.)	32	265	603	84	49	405
	Ускорение	***2.34***	***2.60***	***2.23***	***6.24***	***2.62***	***2.90***
4 ПМ	Время (с.)	23	183	232	46	37	261
	Ускорение	***3.35***	***3.77***	***5.78***	***11.46***	***3.48***	***4.50***
6 ПМ	Время (с.)	22	70	145	35	35	208
	Ускорение	***3.49***	***9.85***	***9.28***	***15.16***	***3.61***	***5.66***
8 ПМ	Время (с.)	20	54	136	30	34	203
	Ускорение	***3.84***	***12.81***	***9.90***	***17.63***	***3.73***	***5.78***
12 ПМ	Время (с.)	19	38	124	30	32	148
	Ускорение	***3.96***	***18.26***	***10.86***	***17.45***	***4.03***	***7.94***
16 ПМ	Время (с.)	16	33	93	27	28	121
	Ускорение	***4.66***	***20.90***	***14.39***	***19.80***	***4.59***	***9.73***

Таблица 2. Ускорение параллельного оптимизированного алгоритма на графах с 45 вершинами

В процессе тестирования использовались компьютеры на базе четырехъядерного процессора Intel Core i3 с тактовой частотой 3.07 GHz и 4 GB оперативной памяти, работающие под управлением 64-разрядной ОС Windows 7. Пропускная способность сети равнялась 100 Mb/s.

В таблице 2 приведены результаты вычислений, позволяющие оценить ускорение, полученное при распараллеливании оптимизированного алгоритма на 6 случайных графах с 45 вершинами. Способ их генерации описан выше. Отметим, что это ускорение для оптимизированного алгоритма, в котором, как отмечалось выше, уже исключены лишние вычисления. Строка "ускорение" для одного ПМ содержит отношение времени работы параллельного алгоритма на одном модуле ко времени работы алгоритма последовательного. Это своего рода накладные расходы, и, как видно из таблицы, они не слишком высоки. В остальных случаях ускорение считалось также относительно времени работы именно последовательного варианта.

Отметим, что ускорение нередко превышало количество исполнителей. Для рекурсивно-параллельной организации алгоритмов "ветвей и границ" это вполне естественно, поскольку при поиске на решения "с разных сторон" больше шансов быстро найти хорошее решение и за счет этого отсечь большее количество неперспективных ветвей. Такое явление наблюдалось и при решении задачи о клике [13].

		Базовый алгоритм Литтла			Оптимизированный алгоритм Литтла		
		Вар.1	Вар.2	Вар.3	Вар.1	Вар.2	Вар.3
Посл. алг.	Итераций	*31 832 080*	*11 242 224*	*25 721 496*	*242 609*	*1 362 560*	*1 601 237*
	Время (с.)	3331	1119	2528	25	79	143
1 ПМ	Время (с.)	3423	1150	2617	29	85	156
	Ускорение	***0.97***	***0.97***	***0.97***	***0.87***	***0.93***	***0.92***
	Отн. баз.				***115.30***	***13.16***	***16.23***
2 ПМ	Время (с.)	1091	284	489	21	23	54
	Ускорение	***3.05***	***3.95***	***5.17***	***1.22***	***3.41***	***2.67***
	Отн. баз.				***161.94***	***48.20***	***47.11***
4 ПМ	Время (с.)	312	122	366	17	17	39
	Ускорение	***10.68***	***9.16***	***6.90***	***1.44***	***4.55***	***3.70***
	Отн. баз.				***191.07***	***64.41***	***65.22***
8 ПМ	Время (с.)	189	76	199	16	15	31
	Ускорение	***17.66***	***14.82***	***12.67***	***1.54***	***5.30***	***4.59***
	Отн. баз.				***203.87***	***74.92***	***81.00***
12 ПМ	Время (с.)	119	54	153	14	16	25
	Ускорение	***27.93***	***20.75***	***16.49***	***1.77***	***5.00***	***5.66***
	Отн. баз.				***234.07***	***70.73***	***99.93***
16 ПМ	Время (с.)	98	51	141	15	12	24
	Ускорение	***34.01***	***22.04***	***17.92***	***1.66***	***6.43***	***6.00***
	Отн. баз.				***219.20***	***90.91***	***105.93***

Таблица 3. Ускорение параллельной версии для базового и оптимизированного алгоритмов на графах с 40 вершинами

Для того чтобы оценить ускорение одновременно и за счет оптимизации, и за счет распараллеливания, для нескольких примеров были проведены эксперимен-

ты одновременно как для базового, так и для оптимизированного алгоритма. Мы ограничились здесь графами с 40 вершинами, поскольку, как было выше отмечено, при большем числе вершин базовый алгоритм обычно работает недопустимо долго. Из числа сгенерированных случайных графов мы выбрали три с наибольшим количеством итераций, потребовавшихся для решения задачи.

Из таблицы 3 видно, что параллельный вариант оптимизированного алгоритма относительно своего же последовательного варианта обеспечивает меньший выигрыш, чем базовый параллельный относительно базового последовательного, что, если честно, неудивительно. Однако он все равно работает намного быстрее базового варианта. И с ростом размера задачи этот разрыв, по-видимому, будет только расти, достаточно сравнить эти результаты с результатами для графов с 45 вершинами. Можно отметить, что описанный подход к распараллеливанию алгоритма для базового варианта работает также очень хорошо.

5. Результаты использования РП-алгоритма для ориентированных графов

Все упомянутые выше результаты относились к неориентированным графам, для них, собственно, и разрабатывался оптимизированный алгоритм. Однако нас интересовали также и возможности рекурсивного распараллеливания алгоритма Литтла для ориентированных графов, разумеется, уже без описанной выше оптимизации.

Для эксперимента мы генерировали случайные графы, длина дуг в которых была равномерно распределена в промежутке от 1 до 5000. Разумеется, применительно к исходной формулировке задачи такие примеры выглядят очень уж неестественно, однако нас в первую очередь интересовало качество получающегося параллельного алгоритма и возможности библиотеки RPM_ParLib.

		Вар.1	Вар.2	Вар.3	Вар.4	Вар.5	Вар.6
Последов. алгоритм	Итераций	*2714275*	*1774096*	*1586139*	*2006961*	*2723686*	*1394583*
	Время (с.)	801	553	485	590	853	414
1 ПМ	Время (с.)	817	593	489	594	878	425
	Ускорение	***0.98***	***0.93***	***0.99***	***0.99***	***0.97***	***0.97***
2 ПМ	Время (с.)	240	149	182	233	228	118
	Ускорение	***3.34***	***3.72***	***2.67***	***2.53***	***3.75***	***3.51***
4 ПМ	Время (с.)	156	106	164	130	137	85
	Ускорение	***5.15***	***5.22***	***2.96***	***4.53***	***6.23***	***4.90***
8 ПМ	Время (с.)	145	71	124	94	88	78
	Ускорение	***5.53***	***7.83***	***3.91***	***6.30***	***9.75***	***5.29***
12 ПМ	Время (с.)	106	67	108	63	69	75
	Ускорение	***7.52***	***8.26***	***4.51***	***9.43***	***12.42***	***5.50***
16 ПМ	Время (с.)	109	65	99	53	59	75
	Ускорение	***7.34***	***8.56***	***4.90***	***11.18***	***14.49***	***5.54***

Таблица 4. Результаты вычислений для ориентированных графов с 70 вершинами

Поскольку для графов, построенных таким образом, отсечение неперспективных направлений поиска кратчайшего цикла происходит намного быстрее, за приемлемое время мы можем решать задачи большего размера. В таблице 4 приведены результаты некоторых вычислений для графов с 70 вершинами. Для этого эксперимента мы сгенерировали 15 вариантов исходных данных и выбрали из них 6 самых трудоемких.

Рассмотрение полученных результатов приводит нас к выводу о том, что и в этом случае алгоритм Литтла обладает хорошим потенциалом для организации вычислений в рекурсивно-параллельном стиле. При этом, как правило, на более трудоемких задачах удается достичь большего ускорения.

Список литературы / References

[1] Гэри М. Джонсон Д., *Вычислительные машины и труднорешаемые задачи*, Мир, Москва, 1982; [Garey M. R., Johnson D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co, San Francisco, 1979.]

[2] Land A. H. Doig A. G., "An Autmatic Method of Solving Discrete Programming Problems", *Econometrica*, **28** (1960), 497–520.

[3] John D. C. Little, Katta G. Murty, Dura W. Sweeney, Caroline Karel, "An Algorithm for the Traveling Salesman Problem", *Operations Research*, **11**:6 (1963), 972–989.

[4] Кофман А., *Введение в прикладную комбинаторику*, Наука, Москва, 1975; [Kaufmann A., *Intraduction a la combinatorique en vue des applications*, Dumond, Paris, 1968.]

[5] Рейнгольд Э. Нивергельт Ю. Део Н., *Комбинаторные алгоритмы. Теория и практика*, Мир, Москва, 1980; [Reingold E. Nievergelt Ju. Deo N., *Combinatorial Algorithms: Theory and Practice*, Prentice Hall College, 1977.]

[6] Кормен Т. Лейзерсон Ч. Ривест Р. Штайн К., *Алгоритмы: построение и анализ*, Вильямс, Москва, 2006; [Cormen T. Leiserson Ch. Rivest R. Stein C., *Introduction to Algorithms*, The MIT Press, 2001.]

[7] Петрунин С. В., "Использование метода последовательной сепарации для решения задачи коммивояжёра", *Научный вестник МГТУ ГА*, 2009, № 146, 105–108; [Petrounine S.V., "Employment of permanent separation metrhod for solution of traveling salesman problems", *Scientifik bulletin of the MSTU CA*, 2009, № 146, 105–108, (in Russian).]

[8] Костюк Ю. Л., "Эффективная реализация алгоритма решения задачи коммивояжёра методом ветвей и границ", *Прикладная дискретная математика*, 2013, № 2, 78–90; [Kostyuk Yu. L., "Effective implementation of algorithm for solving the travelling salesman problem by branch-and-bound method", *Prikl. Diskr. Mat.*, 2013, № 2, 78–90, (in Russian).]

[9] Сигал И. Х., Бабинская Я. Л., Посыпкин М. А., "Параллельная реализация метода ветвей и границ в задаче коммивояжера на базе библиотеки BNB-Solver", *Труды ИСА РАН*, **25** (2006), 26–36; [Sigal I. Kh., Babinskaya Ya. L., Pocypkin M. A., "Parallelnaya realizaciya metoda vetvey i granic v zadache kommivoyazhera na baze biblioteki BNB-Solver", *Trudy ISA RAN*, **25** (2006), 26–36, (in Russian).]

[10] Васильчиков В. В., *Средства параллельного программирования для вычислительных систем с динамической балансировкой загрузки*, ЯрГУ, Ярославль, 2001; [Vasilchikov V. V., *Sredstva parallelnogo programmirovaniya dlya vychislitelnykh sistem s dinamicheskoy balansirovkoy zagruzki*, YarGU, Yaroslavl, 2001, (in Russian).]

[11] Васильчиков В. В., *Коммуникационный модуль для организации полносвязного соединения компьютеров в локальной сети с использованием .NET Framework*, Свидетельство о государственной регистрации программы для ЭВМ № 2013619925,

2013; [Vasilchikov V. V., *Kommunikatsionnyy modul dlya organizatsii polnosvyaznogo soedineniya kompyuterov v lokalnoy seti s ispolzovaniem .NET Framework*, Svidetelstvo o gosudarstvennoy registratsii programmy dlya EVM № 2013619925, 2013, (in Russian).]

[12] Васильчиков В. В., *Библиотека поддержки рекурсивно-параллельного программирования для .NET Framework*, Свидетельство о государственной регистрации программы для ЭВМ № 2013619926, 2013; [Vasilchikov V. V., Svidetelstvo o gosudarstvennoy registratsii programmy dlya EVM № 2013619926, 2013, (in Russian).]

[13] Васильчиков В. В., "О поддержке рекурсивно-параллельного программирования в .NET Framework", *Модел. и анализ информ. систем*, **21**:2 (2014), 15–25; [Vasilchikov V. V., "On the Recursive-Parallel Programming for the .NET Framework", *Modeling and Analysis of Information Systems*, **21**:2 (2014), 15–25, (in Russian).]

Vasilchikov V. V., "On the Optimization and Parallelizing Little Algorithm for Solving the Traveling Salesman Problem", *Modeling and Analysis of Information Systems*, **23**:4 (2016), 401–411.

DOI: 10.18255/1818-1015-2016-4-401-411

Abstract. The paper describes some ways to accelerate solving the NP-complete Traveling Salesman Problem. The classic Little algorithm belonging to the category of "branch and bound methods" can solve it both for directed and undirected graphs. However, for undirected graphs its operation can be accelerated by eliminating the consideration of branches examined earlier. The paper proposes changes to be made in the key operations of the algorithm to speed up its execution. It also describes the results of an experiment that demonstrated a significant acceleration of solving the problem by using an advanced algorithm. Another way to speed up the work is to parallelize the algorithm. For problems of this kind it is difficult to break the task into a sufficient number of subtasks having comparable complexity. Their parallelism arises dynamically during the execution. For such problems, it seems reasonable to use parallel-recursive algorithms. In our case the use of the library RPM_ParLib developed by the author was a good choice. It allows us to develop effective applications for parallel computing on a local network using any .NET-compatible programming language. We used C# to develop the programs. Parallel applications were developed as for basic and modified algorithms, the comparing of their speed was made. Experiments were performed for the graphs with the number of vertexes up to 45 and with the number of network computers up to 16. We also investigated the acceleration that can be achieved by parallelizing the basic Little algorithm for directed graphs. The results of these experiments are also presented in the paper.

Keywords: Traveling Salesman Problem, Little algorithm, parallel algorithm, recursion, .NET

On the authors:
Vasilchikov Vladimir Vasilyevich, orcid.org/0000-0001-7882-8906, PhD,
P.G. Demidov Yaroslavl State University,
Sovetskaya str., 14, Yaroslavl, 150000, Russia, e-mail: vvv193@mail.ru

Acknowledgments:
This work was supported by initiative program VIP-004 (state registration number AAAA-A16-116070610022-6).