Синхронный балансировщик нагрузки для задачи коммивояжера

А.Ю. Горчаков[1]

***Аннотация*—При разработке параллельных методов решения многих численных методов решения прикладных задач, в частности метода ветвей и границ, возникает проблема балансировки нагрузки. Выбор вариантов реализации на текущий момент предложено достаточно много. В первую очередь рассматриваются схемы с выделенным управляющим процессом, который выдает задания процессам «решателям» и собирает от них результаты решения подзадач. Следующий тип балансировки нагрузки, это древовидная схема процессов, соответствующая дереву решения задачи. При этом для всех вариантов балансировки нагрузки возникает вопрос масштабирования. Ответ на этот вопрос может быть получен либо с помощью масштабного тестирования, либо путем моделирования. В работе рассматривается параллельная реализация одного из вариантов метода ветвей и границ для решения асимметричной задачи коммивояжера. Метод разработан на языке программирования python, с использованием пакета mpi4py, обеспечивающего функциональность стандарта интерфейса передачи сообщений (MPI) для языка программирования Python, что позволяет любой программе Python использовать несколько процессоров. Балансировка нагрузки обеспечивается посредством синхронных (блокирующих) коллективных операций. Проведен численный эксперимент на подмножестве тестового набора задач TSPLIB. На основе собранных данных определены основные характеристики балансировщика нагрузки и его узкие места. Кроме этого, проведено статистическое моделирование, для определения качества работы балансировщика нагрузки в случае его масштабирования до 10^7 процессов.**

***Ключевые слова*—асимметричная задача коммивояжера, метод ветвей и границ, балансировка нагрузки, статистическое моделирование.**

I. ВВЕДЕНИЕ

Задача коммивояжера (Traveling Salesman Problem - TSP) является классической задачей комбинаторной оптимизации [1], [2].

Существуют различные вариации постановок задачи коммивояжера, в данной работе мы будем рассматривать следующую задачу: пусть имеется n городов, для каждой пары городов определено расстояние $c_{ij} >= 0, i, j \in \{0, ..., n-1\}$ между ними, причем c_{ij} может быть не равно c_{ji}, коммивояжер выезжая из города 0, объезжает все города, посещая каждый из них ровно один раз и возвращается в город 0. Требуется найти последовательность посещения городов, при которой суммарная длина пути минимальна. Формально можно рассматривать как комбинаторную постановку задачи, так и постановку в виде задачи целочисленного линейного программирования.

Комбинаторная постановка задачи: введем понятие маршрута – $p = \{0, i_1, i_2, ..., i_{n-1}\}$, где $i_1, i_2, ..., i_{n-1}$ – перестановка чисел $1, 2, ...n-1$; длина маршрута равна

$$l(p) = \sum_{k=0}^{n-1} c_{i_k, i_{k+1}}, i_0 = i_n = 0 \quad (1)$$

пусть P – множество всех возможных маршрутов; необходимо найти маршрут $p^* \in P$ такой, что

$$l(p^*) = \min_{p \in P} l(p) \quad (2)$$

Постановка в виде задачи целочисленного линейного программирования: ведем квадратную матрицу X состоящую из булевых переменных;

$$x_{ij} = \begin{cases} 1, & \text{если коммивояжер переезжает из города } i \\ & \text{в город } j \text{ непосредственно} \\ 0, & \text{в ином случае} \end{cases}$$

длина маршрута

$$l(X) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} c_{ij} x_{ij} \to min \quad (3)$$

условия однократного посещения города имеют следующий вид

$$\sum_{i=0}^{n-1} x_{ij} = 1, j = 0, ..., n-1 \quad (4)$$

$$\sum_{j=0}^{n-1} x_{ij} = 1, i = 0, ..., n-1 \quad (5)$$

Задача $(3) - (5)$ без дополнительных условий является задачей о назначениях, ее решением может быть не один замкнутый маршрут, а несколько. Для гарантии наличия одного маршрута вводят специальные условия. Классическим методом решения задачи является метод ветвей и границ [3], [4]. Существует множество вариаций данного метода, различаются алгоритмы отсева задач и ветвления [1], [4], [5]. Для исследований был взят метод с алгоритмом ветвления предложенный в [5], а для отсева задач использовалась одна из реализаций [6] Венгерского алгоритма [7]–[9]. Метод тестировался на асимметричных задачах коммивояжера из известной библиотеки TSPLIB [10]. Для распараллеливания использовался пакет mpi4py [11]. Работа выполнялась с использованием инфраструктуры Центра коллективного пользования «Высокопроизводительные вычисления и большие данные» (ЦКП «Информатика») ФИЦ ИУ РАН (г. Москва). [12].

*Работа выполнена при частичной поддержке поддержке РФФИ, проект 19-01-00666

[1]А.Ю. Горчаков — старший научный сотрудник Федерального исследовательского центра «Информатика и управление» Российской академии наук. andrgor12@gmail.com

II. Описание метода ветвей и границ

Приведем описание параллельной реализации метода:

1) Вход: A матрица расстояний между городами. Возвращаем оптимальный маршрут и его длину.
2) С помощью mpirun запускается K параллельных взаимодействующих процессов. Каждый процесс инициализирует пустой список задач $tasks$.
3) В процессе с номером 0 матрица A считывается из файла, максимально возможным числом инициализируется переменная $RecUB$ и в список задач добавляется задача $(0, A)$, где 0 - нижняя оценка длины маршрута родительской задачи.
4) Каждый процесс в цикле выполняет пункты 5-14.
5) Если список задач пуст, то переходим к пункту 10.
6) Берем задачу из списка $tasks$ (метод .pop()). Если нижняя оценка длины маршрута родительской задачи $LB \geq RecUB$, то переходим к пункту 10.
7) Венгерским методом решаем задачу о назначениях, получая новое значение переменной LB и список маршрутов.
8) Если $LB \geq RecUB$, то переходим к пункту 10.
9) Если список маршрутов содержит всего один маршрут и $RecUB < LB$, то обновляем значение переменной $RecUB = LB$ и запоминаем этот маршрут. В ином случае, мы выбираем маршрут минимальной длины и добавляем к списку задач столько задач $(0, A_i)$, сколько городов имеется в минимальном маршруте. При этом вводятся дополнительные ограничения, для i-ой задачи запрещаются все переходы из i-го города в города этого маршрута.
10) С помощью коллективной операции $allreduce$ с параметром $op = MPI.MIN$ находим минимальное значение переменной $RecUB$ и рассылаем результат всем процессам.
11) С помощью коллективной операции $allreduce$ с параметром $op = MPI.MAXLOC$ находим максимальное количество задач и номер процесса где достигается максимум, рассылаем результат всем процессам.
12) Если максимальное количество задач равно нулю, то переходим к пункту 15 (выходим из цикла - все задачи решены).
13) С помощью коллективной операции $allreduce$ с параметром $op = MPI.MINLOC$ находим минимальное количество задач и номер процесса где достигается минимум, рассылаем результат всем процессам.
14) Если максимальное количество задач больше или равно 2 и минимальное количество задач равно нулю, то процесс с максимальным количеством задач отправляет процессу с минимальным количеством задач половину своих задач. Пересылка производится с помощью команд $send$ и $recv$. Далее переходим к пункту 10. В ином случае возвращаемся к пункту 5.
15) С помощью коллективной операции $reduce$ с параметром $op = MPI.MINLOC$ находим процесс с минимальной переменной и забираем с этого процесса оптимальный маршрут.

Балансировка нагрузки в данном методе осуществляется с помощью коллективных операций. Выделенный управляющий процесс отсутствует. Каждый процесс обладает одинаковой информацией о значении рекорда и количестве задач у процессов (только минимальное и максимальное количество). Объем передаваемой информации при этом минимален. В пункте 10 находится и рассылается одно число - значение переменной $RecUB$. В пунктах 11, 13 два числа - минимальное/максимальное количество задач и номер процесса где оно достигается. Пересылка задач от процесса к процессу производится только в случае необходимости, когда минимальное количество задач у одного из процессов равно нулю.

III. Результаты численного эксперимента

Эксперименты проводились на персональном компьютере с двумя процессорами Intel Xeon E5-2699A, 2.4 GHz, 22 ядра. В качестве тестового набора задач были взяты тесты ftv33, ftv35, ftv38, ftv44, ftv47, ftv55 из библиотеки TSPLIB [10]. Каждый тест был решен с использованием 1, 2, 4, 8, 16, 24, 32, 40 процессов. Измерялись следующие параметры:

1) количество решенных задач (в зависимости от количества процессов количество задач изменяется)
2) время затраченное на решение теста деленное на количество задач - $TimePerTasks$, или же время потраченное на решение одной задачи
3) время затраченное на решение задач и ветвление (пункты 5-9) - $Tcalc$
4) время потраченное на синхронизацию (балансировщик полностью синхронный, а время решения задач отличается от задачи к задаче) и обмен рекордным значением (пункт 10) - $Twait$
5) время проведенное в ожидании получения задач (пункт 10 при условии отсутствия задач) - $Tzero$
6) время потраченное на пересылки задач от процесса к процессу - $Tcomm$.

Зависимость количества задач от количества процессов приведена на Рис.1. На рисунках 1 и 2 задаче ftv33

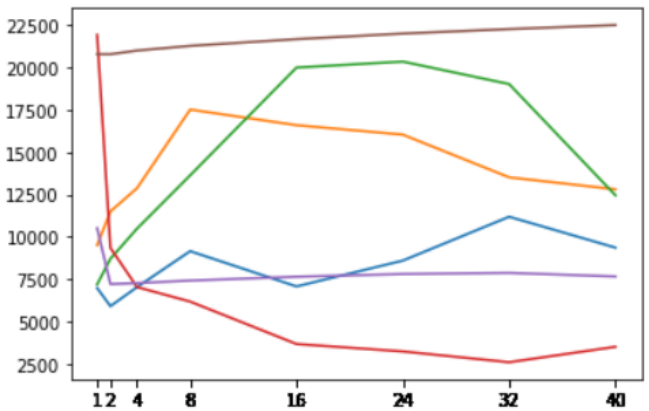

Рис. 1: Зависимость количества задач от количества процессов.

соответвет синяя, ftv35 - желтая, ftv38 - зеленая, ftv44 - красная, ftv47 - фиолетовая, ftv55 - коричневая линия. Как мы можем увидеть, количество задач может сильно изменяется в зависимости от количества процессов. Это ожидаемый эффект для метода поиска в ”глубину”.

Подобный эффект был отмечен ранее в работах [13], [14]. Именно из-за этого эффекта при исследовании параллельных алгоритмов важен параметр $TimePerTasks$ см. Рис.2. Как мы можем увидеть, для всех тестов зави-

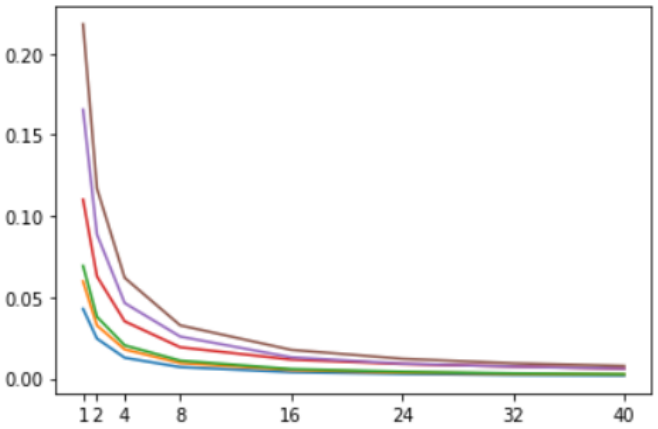

Рис. 2: Зависимость времени решения одной задачи от количества процессов.

симость времени решения одной задачи от количества процессов имеет один и тот же вид без каких-либо изменений или выбросов.

Следующий достаточно важный показатель, который необходимо исследовать, это какую долю в общем времени решения теста занимает та или иная часть алгоритма. На Рис. 3, 4 приведены эти доли. $Tcalc$ отмечен синим, $Tzero$ - оранжевым, $Twait$ - зеленым, $Tcomm$ - красным цветами. Как мы можем увидеть, для 40 потоков наименьшую долю времени от 0.6 до 4.4 процентов занимает $Tzero$ - ожидание процессом получения задачи. Можно сказать, что ситуация простоя процесса по причине отсутствия задач из-за плохой балансировки нагрузки маловероятна. $Tcomm$ - затраты на пересылку задач от процесса к процессу занимают большую долю времени - от 5 до 11 процентов. От 19 до 39 процентов время потраченное на синхронизацию $Twait$. На решение задач остается от 47 до 71 процента времени.

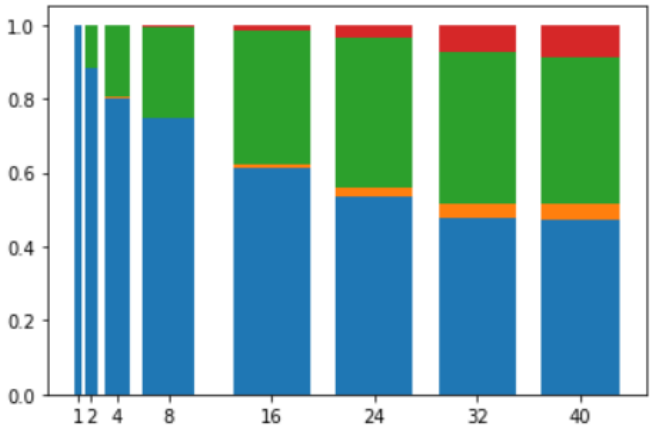

Рис. 3: Доли временных затрат в общем времени решения теста ftv44.

По результатам проведенных экспериментов можно сделать следующие выводы: данный балансировщик на тестовом наборе обеспечивает загрузку процессов задачами, простой без задач минимален; для небольшого

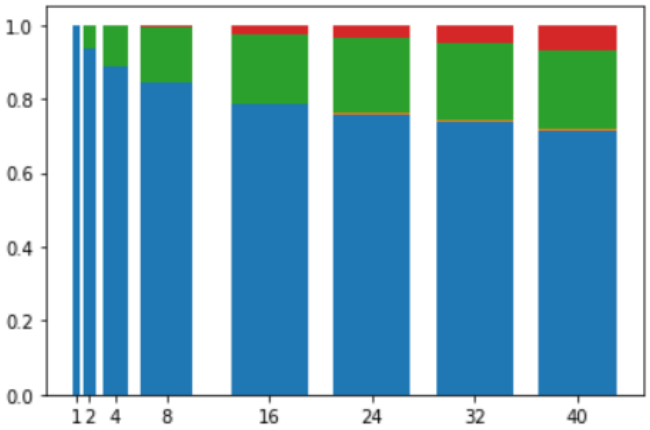

Рис. 4: Доли временных затрат в общем времени решения теста ftv55.

количества процессов (в нашем примере 40 процессов) оптимизация работы балансировщика нецелесообразна, даже если мы за счет асинхронной обработки "обнулим" $Twait$ и $Tcomm$, то более чем в два раза ускорить решение тестов не получиться. Тем не менее, всегда возникает вопрос, а что будет если количество процессов существенно увеличится? Ответить на него можно с помощью иммитационного моделирования (англ. simulation modeling).

IV. Симулятор балансировки нагрузки

Для проведения имитационного моделирования возьмем упрощенную модель:

1) В цикле каждый процесс выполняет пункты 2 и 3
2) решаем одну задачу - $Tcalc$
3) ждем пока все процессы не закончат решение задач - $Twait$.

Чтобы оценить время решения задачи возьмем тест ftv55 и посмотрим на частотное распределение времени решения задач, см. Рис.5

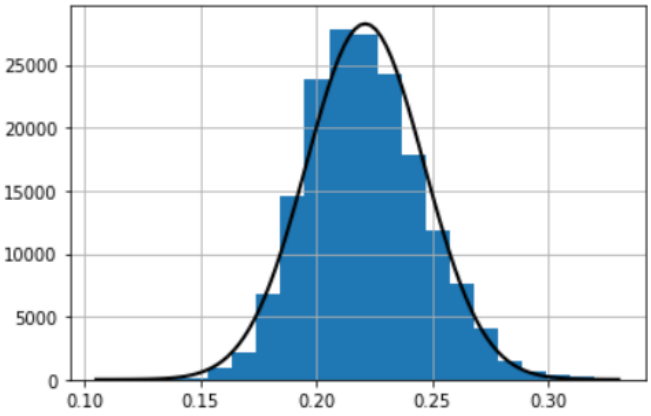

Рис. 5: Частотное распределение времени решения задач.

Как мы видим из рисунка распределение времени решения одной задачи очень близко к нормальному. Поэтому процесс решения задач можно смоделировать генератором случайных чисел: пусть r вектор, состоящий из случайных чисел с нормальным распределением, размерность вектора N равна количеству процессов, тогда

$Tcalc = \sum_i r_i$, а $Twait = N \max_i r_i - Tcalc$. Так как $Tcalc$ и $Twait$ в таком определении так же являются случайными величинами, то для оценки их математических ожиданий с заданной точностью мы проводим последовательно серию экспериментов до тех пор, пока ширина 95% доверительного интервала [15] для обеих величин не будет меньше 5% их выборочного среднего. Результаты моделирования приведены на рис.6. На ри-

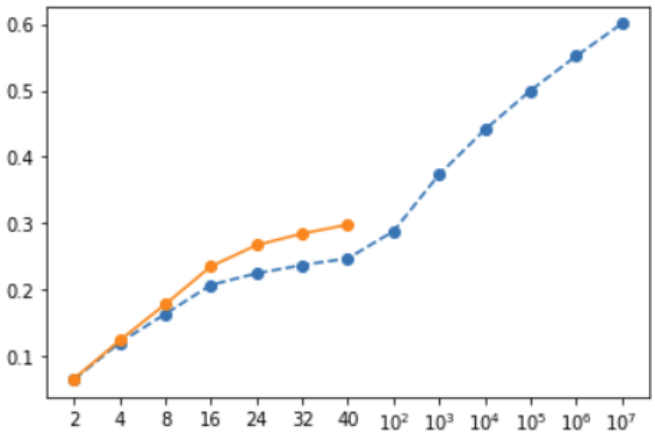

Рис. 6: Доля времени потраченного на синхронизацию в зависимости от количества процессов.

сунке оранжевой линией показаны данные теста ftv55, а синей линией результаты моделирования. Как можно увидеть результаты имитационного моделирования качественно повторяют данные полученные на реальной системе, и для 10^7 процессов доля времени потраченного на синхронизацию не превышает 60 процентов.

V. ЗАКЛЮЧЕНИЕ

Анализируя результаты численных экспериментов и имитационного моделирования можно сделать вывод, что предложенная параллельная реализация метода ветвей и границ для асиметричной задачи коммивояжера, в части балансировки нагрузки, показывает достаточно хорошие результаты.

Список литературы

[1] The traveling salesman problem: a computational study / David L Applegate, Robert E Bixby, Vasek Chvatal, William J Cook. — Princeton university press, 2006.

[2] Gutin Gregory, Punnen Abraham P. The traveling salesman problem and its variations. — Springer Science & Business Media, 2006. — Vol. 12.

[3] Посыпкин Михаил Анатольевич, Сигал Израиль Хаимович. Исследование алгоритмов параллельных вычислений в задачах дискретной оптимизации ранцевого типа // Журнал вычислительной математики и математической физики. — 2005. — Vol. 45, no. 10. — P. 1801–1809.

[4] Сигал ИХ, Иванова АП. Введение в прикладное дискретное программирование. Модели и вычислительные алгоритмы. — 2007.

[5] Bellmore Mandell, Malone John C. Pathology of traveling-salesman subtour-elimination algorithms // Operations Research. — 1971. — Vol. 19, no. 2. — P. 278–307.

[6] Clapper Brian. Munkres implementation for python. — 2019. — URL: https://github.com/bmc/munkres (online; accessed: 2020-07-23).

[7] Kuhn Harold W. The hungarian method for the assignment problem // Naval research logistics quarterly. — 1955. — Vol. 2, no. 1-2. — P. 83–97.

[8] Kuhn Harold W. Variants of the hungarian method for assignment problems // Naval research logistics quarterly. — 1956. — Vol. 3, no. 4. — P. 253–258.

[9] Munkres James. Algorithms for the assignment and transportation problems // Journal of the society for industrial and applied mathematics. — 1957. — Vol. 5, no. 1. — P. 32–38.

[10] Reinelt Gerhard. Tsplib—a traveling salesman problem library // ORSA journal on computing. — 1991. — Vol. 3, no. 4. — P. 376–384.

[11] Parallel distributed computing using python / Lisandro D Dalcin, Rodrigo R Paz, Pablo A Kler, Alejandro Cosimo // Advances in Water Resources. — 2011. — Vol. 34, no. 9. — P. 1124–1139.

[12] Положение о ЦКП ”Информатика”. — 2020. — URL: http://www.frccsc.ru/ckp (online; accessed: 2020-07-23).

[13] Горчаков Андрей Юрьевич, Посыпкин Михаил Анатольевич. Сравнение вариантов многопоточной реализации метода ветвей и границ для многоядерных систем // Современные информационные технологии и ИТ-образование. — 2018. — Vol. 14, no. 1.

[14] Горчаков АЮ, Посыпкин МА, Ямченко ЮВ. экспериментальное исследование трех вариантов реализации метода неравномерных покрытий для многоядерных систем с общей памятью // International Journal of Open Information Technologies. — 2018. — Vol. 6, no. 11.

[15] A Modern Introduction to Probability and Statistics: Understanding why and how / Frederik Michel Dekking, Cornelis Kraaikamp, Hendrik Paul Lopuhaä, Ludolf Erwin Meester. — Springer Science & Business Media, 2005.

Synchronous Load Balancer for Traveling Salesman Problem

Andrei Y. Gorchakov

***Abstract*—When developing parallel methods for solving many numerical methods for solving applied problems, in particular the branch-and-bound method, the problem of load balancing arises. The choice of implementation options at the moment has been proposed quite a lot. First of all, schemes with a dedicated control process are considered, which issues tasks to the processes "solvers"and collects from them the results of solving subtasks. The next type of load balancing is a tree-like diagram of processes corresponding to the solution tree of the problem. At the same time, for all load balancing options, the issue of scaling arises. The answer to this question can be obtained either through large-scale testing or through simulation. This paper considers a parallel implementation of one of the branch-and-bound method variants for solving the asymmetric traveling salesman problem. The method was developed in the python programming language, using the mpi4py package, which provides the functionality of the Message Passing Interface (MPI) standard for the Python programming language, which allows any Python program to use multiple processors. Load balancing is provided through synchronous (blocking) collective operations. A numerical experiment was carried out on a subset of the test set of TSPLIB problems. Based on the collected data, the main characteristics of the load balancer and its bottlenecks are identified. In addition, statistical modeling was carried out to determine the quality of the load balancer in the case of its scaling up to 10^7 processes.**

***Keywords*—asymmetric traveling salesman problem, branch and bound method, load balancing, statistical modeling.**

REFERENCES

[1] The traveling salesman problem: a computational study / David L Applegate, Robert E Bixby, Vasek Chvatal, William J Cook.— Princetonuniversity press, 2006.

[2] Gutin Gregory, Punnen Abraham P. The traveling salesman problemand its variations.— Springer Science Business Media, 2006.—Vol. 12.

[3] Posypkin M. S., Segal I.H. Investigation of parallel computation algorithms in knapsack-type discrete optimization problems // Journal of Computational Mathematics and Mathematical Physics .— 2005.—Vol. 45, no. 10.— P. 1801–1809.

[4] Segal I.H., Ivanova A.P. Introduction to Applied Discrete Programming. Models and computational algorithms .—2007.

[5] Bellmore Mandell, Malone John C. Pathology of traveling-salesman subtour-elimination algorithms // Operations Research.— 1971.—Vol. 19, no. 2.— P. 278–307.

[6] Clapper Brian. Munkres implementation for python.– – 2019.— URL:https://github.com/bmc/munkres(online; accessed: 2020-07-23).

[7] Kuhn Harold W. The hungarian method for the assignment problem //Naval research logistics quarterly.— 1955.— Vol. 2, no. 1-2.— P. 83–97.

[8] Kuhn Harold W. Variants of the hungarian method for assignmentproblems // Naval research logistics quarterly.— 1956.— Vol. 3,no. 4.— P. 253–258.

[9] Munkres James. Algorithms for the assignment and transportationproblems // Journal of the society for industrial and applied math-ematics.— 1957.— Vol. 5, no. 1.— P. 32–38.

[10] Reinelt Gerhard. Tsplib—a traveling salesman problem library //ORSA journal on computing.— 1991.— Vol. 3, no. 4.— P. 376–384.

[11] Parallel distributed computing using python / Lisandro D Dalcin,Rodrigo R Paz, Pablo A Kler, Alejandro Cosimo // Advances in WaterResources.— 2011.— Vol. 34, no. 9.— P. 1124–1139.

[12] Regulations of CKP "Computer Science".— 2020.– – URL:http://www.frccsc.ru/ckp(online; accessed: 2020-07-23).

[13] Gorchakov A.Y., Posypkin M.A. Comparison of options for multithreaded implementation of branch and boundary methods for multicore systems // Modern information technologies and IT education .— 2018.—Vol. 14, no. 1.

[14] Gorchakov A.Y., Posypkin M.A., Yamchenko Y.B. Experimental study of three variants of non-uniform coverage method implementation for multicore systems with shared memory // Interna-tional Journal of Open Information Technologies.— 2018.— Vol. 6,no. 11.

[15] A Modern Introduction to Probability and Statistics: Understandingwhy and how / Frederik Michel Dekking, Cornelis Kraaikamp, Hen-drik Paul Lopuhaä, Ludolf Erwin Meester.— Springer Science Business Media, 2005.