MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 29, NO. 1, 2022
journal homepage: www.mais-journal.ru

SOFTWARE

Recursive-Parallel Algorithm for Solving the Graph-Subgraph Isomorphism Problem

V. V. Vasilchikov[1]

DOI: 10.18255/1818-1015-2022-1-30-43

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

MSC2020: 68W10
Research article
Full text in Russian

Received November 29, 2021
After revision February 28, 2022
Accepted March 9, 2022

The paper proposes a parallel algorithm for solving the Graph-Subgraph Isomorphism Problem and makes an experimental study of its efficiency. The problem is one of the most famous NP-complete problems. Its solution may be required when solving many practical problems associated with the study of complex structures. We solve the problem in a formulation that requires finding all existing isomorphic substitutions or proving their absence. In view of the high complexity of the problem, it is natural to want to speed up its solution by parallelizing the algorithm.
We used the RPM_ParLib library, developed by the author, as the main tool to program the algorithm. This library allows us to develop effective applications for parallel computing on a local network under the control of the runtime environment .NET Framework. Thanks to this library, applications have the ability to generate parallel branches of computation directly during program execution and dynamically redistribute work between computing modules. Any language with support of the .NET Framework can be used as a programming language in conjunction with this library. For the numerical experiment, an open database from the Internet was used, which was specially developed to study algorithms for searching for isomorphic substitutions. Also, the author has developed a special application in C# for generating additional sets of initial data with specified characteristics. The aim of the experiment was to study the speedup achieved due to the recursively parallel organization of computations. The paper provides a detailed description of the proposed algorithm, as well as the results obtained during the experiment.

Keywords: graph-subgraph isomorphism problem; parallel algorithm; recursion; .NET

INFORMATION ABOUT THE AUTHORS

Vladimir V. Vasilchikov correspondence author	orcid.org/0000-0001-7882-8906. E-mail: vvv193@mail.ru PhD.

Funding: This work was supported by initiative program VIP-016.

For citation: V. V. Vasilchikov, “Recursive-Parallel Algorithm for Solving the Graph-Subgraph Isomorphism Problem”, *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 30-43, 2022.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 29, № 1, 2022
сайт журнала: www.mais-journal.ru

SOFTWARE

Рекурсивно-параллельный алгоритм решения задачи об изоморфизме граф-подграф

В. В. Васильчиков[1]

DOI: 10.18255/1818-1015-2022-1-30-43

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

УДК 519.688: 519.85
Научная статья
Полный текст на русском языке

Получена 29 ноября 2021 г.
После доработки 28 февраля 2022 г.
Принята к публикации 9 марта 2022 г.

В работе предложен параллельный алгоритм решения задачи об изоморфизме граф-подграф и произведено экспериментальное исследование его эффективности. Задача является одной из самых известных NP-полных задач. Ее решение может потребоваться при решении многих практических задач, связанных с исследованием сложных структур. Мы решаем ее в постановке, в которой требуется найти все возможные изоморфизмы или доказать отсутствие таковых. Ввиду высокой трудоемкости задачи естественным является желание ускорить ее решение за счет распараллеливания алгоритма.

Для организации параллельных вычислений автором использовалась библиотека RPM_ParLib, которая позволяет создавать параллельные приложения, работающие в локальной вычислительной сети под управлением среды исполнения .NET Framework. Библиотека поддерживает рекурсивно-параллельный стиль программирования и обеспечивает эффективное распределение работы и динамическую балансировку загрузки вычислительных модулей в процессе исполнения программы. Она может быть использована для приложений, написанных на любом языке программирования, поддерживаемом .NET Framework. Для проведения численного эксперимента использовалась открытая база данных из сети Интернет, специально разработанная для исследования алгоритмов поиска изоморфизмов. Также автором было разработано специальное приложение на языке C# для генерации собственных дополнительных наборов исходных данных с заданными характеристиками. Целью эксперимента было исследование ускорения, достигаемого за счет рекурсивно-параллельной организации вычислений. В работе приводится подробное описание предлагаемого алгоритма, а также полученных в ходе эксперимента результатов.

Ключевые слова: изоморфизм граф-подграф; параллельный алгоритм; рекурсия; .NET

ИНФОРМАЦИЯ ОБ АВТОРАХ

Владимир Васильевич Васильчиков
автор для корреспонденции

orcid.org/0000-0001-7882-8906. E-mail: vvv193@mail.ru
канд. техн. наук, зав. кафедрой вычислительных и программных систем.

Финансирование: Работа выполнена при поддержке инициативного проекта VIP-016.

Для цитирования: V. V. Vasilchikov, “Recursive-Parallel Algorithm for Solving the Graph-Subgraph Isomorphism Problem”, *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 30-43, 2022.

Введение

Задача об изоморфизме граф-подграф является одной из классических NP-полных задач дискретной оптимизации [1]. Потребность в решении этой задачи возникает в самых разных предметных областях, где требуется установление идентичности структур тех или иных сложных систем. В качестве примеров можно назвать транспортные, энергетические системы, системы связи, электронные схемы, задачи сравнения молекулярных структур, системы распознавания образов, а также задачи математической химии, исследование социальных сетей и многие другие.

Для решения данной задачи в разное время было разработано и исследовано множество алгоритмов. В числе самых известных можно назвать алгоритм Ульмана [2], алгоритм VF, первая версия которого была предложена Корделлой, Фогиа, Самсоне и Венто в 1997 году [3] и решала задачу быстрее, а также его усовершенствованный вариант VF2 [4]. Позднее были предложены различные модификации этих алгоритмов, например, VF3 [5], а также алгоритмы, построенные на других подходах: предварительной оценке совместимости вершин графов [6, 7], бит-векторные алгоритмы [8].

Ввиду высокой трудоемкости данной задачи возникает естественный интерес к разработке и исследованию параллельных алгоритмов ее решения. Существует ряд публикаций, посвященных именно таким алгоритмам, основанным на самых разных подходах к распараллеливанию. Например, в [9] был предложен алгоритм решения задачи средствами MPI. В [10] произведено описание алгоритма VF3P – параллельной версии VF3 и исследована его эффективность. VF3P в основном базируется на динамическом назначении задач на обработку с использованием глобального и, возможно, локальных стеков для хранения подзадач, что на наш взгляд, может ограничить возможности масштабирования.

При подготовке данной работы автор использовал специальные программные инструменты для организации параллельных вычислений в соответствии концепцией рекурсивно-параллельного (РП) программирования. В [11] изложены основные принципы организации таких вычислений, алгоритмы и механизмы поддержки рекурсивно-параллельного стиля программирования. Ранее разработанные автором библиотеки [12, 13] позволяют относительно легко создавать, отлаживать и эксплуатировать РП-приложения в среде .NET Framework. Функциональные возможности упомянутых библиотек подробно описаны в [14]. Библиотеки успешно применялись при разработке и исследовании параллельных алгоритмов решения задачи о клике [14], задачи коммивояжера [15], задачи о рюкзаке [16], а также задачи об изоморфизме неориентированных графов [17]. При этом стратегия разделения задачи на параллельно решаемые подзадачи в целях наилучшего использования вычислительных ресурсов каждый раз выстраивалась по-новому.

Постановка задачи

Напомним формулировку задачи в постановке из [1]. Пусть есть два графа, заданных своими множествами вершин и дуг: $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$. Существует ли подмножество $V \subseteq V_1, E \subseteq E_1$ и взаимно-однозначная функция $f(V_2 \longrightarrow V)$ такие, что $|V| = |V_2|, |E| = |E_2|, \{u, v\} \in E_2 \iff \{f(u), f(v)\} \in E$. Упомянутая функция, очевидно, задает некоторую изоморфную подстановку. Мы будем решать задачу в несколько более широкой и более востребованной формулировке, которая требует найти все такие подстановки.

Последовательный алгоритм решения задачи

Как мы отметили выше, разными авторами было предложено много различных алгоритмов решения задачи, основанных на принципиально разных подходах. Нашей целью является исследование возможностей распараллеливания такого широко используемого алгоритма, как VF2.

Сначала опишем упомянутый последовательный алгоритм, поскольку именно он является основой для дальнейшего распараллеливания решения задачи. В основном наш вариант алгорит-

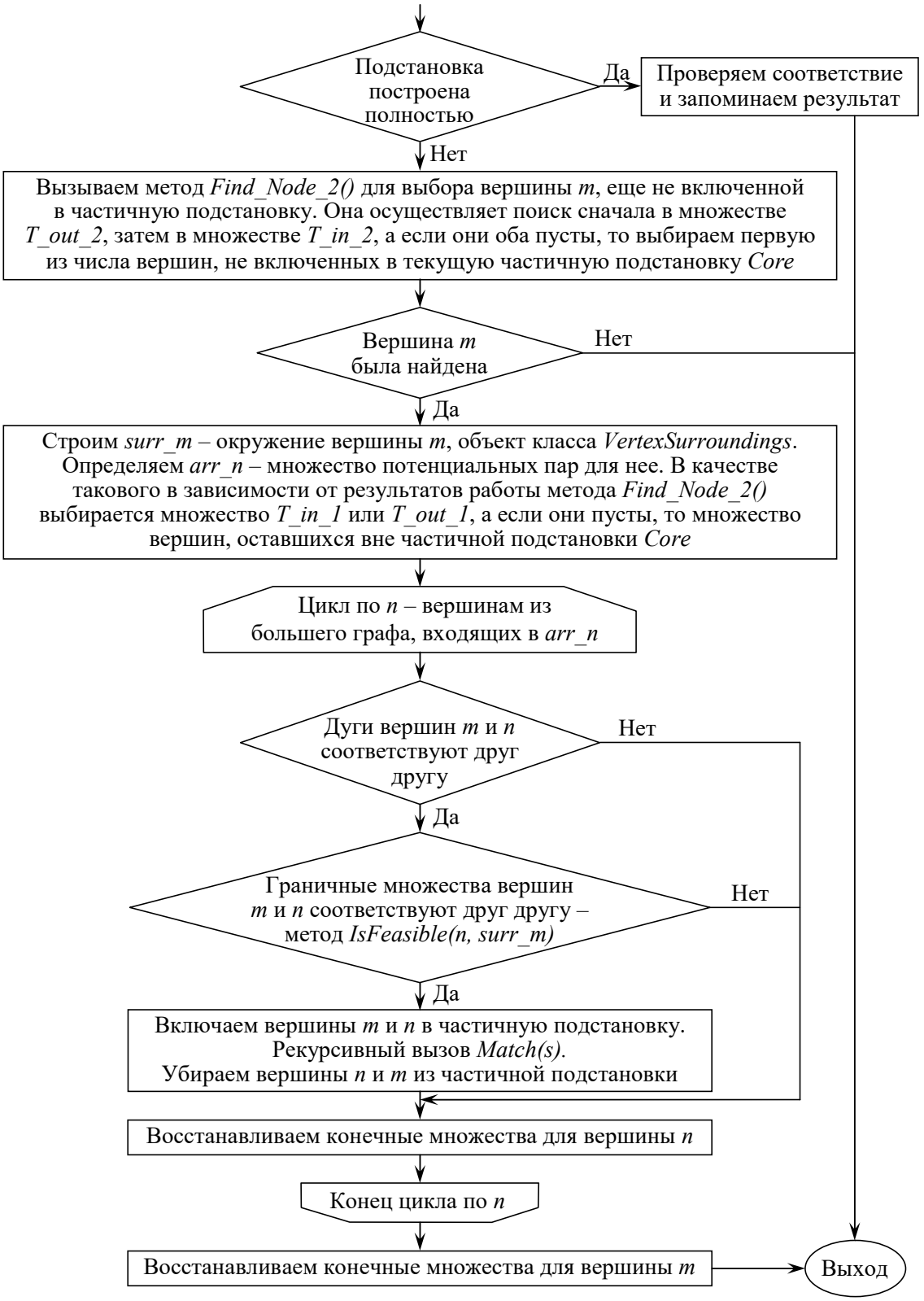

Fig. 1. Block diagram of serial variant of the method *Match(s)*

Рис. 1. Блок-схема последовательного варианта метода *Match(s)*

ма будет соответствовать его описанию, изложенному в [4]. Блок-схема алгоритма представлена на Рис. 1.

Пусть $N = |V1|$ – количество вершин в первом (большем) графе, а $M = |V2|$ – во втором. Основной структурой данных является строящаяся частичная подстановка $Core$, в которой хранятся соответствующие друг другу вершины $n \in V_1$ и $m \in V_2$. Когда ее размер достигает M, это означает, что очередная искомая подстановка построена полностью.

Строительством подстановки занимается рекурсивный метод $Match(s)$, где s – текущее состояние. На каждом уровне рекурсии он пытается найти подходящую пару (n, m) и добавить ее в $Core$. Для этого он сначала вызывает метод $Find\_Node\_2()$ для выбора m, его описание мы приведем чуть ниже. Далее он собирает информацию об окружении вершины m, сохраняет ее в специальном объекте (у нас он назван $surr\_m$), а затем последовательно перебирает вершины-кандидаты на роль n, оценивая их пригодность через вызов метода $IsFeasible(n, surr\_m)$.

Упомянутые методы используют понятие так называемых терминальных множеств, а именно: $T_1^{in}(s)$ – множество вершин, еще не включенных в $Core$, дуги из которых ведут в вершины первого графа, включенные строящуюся частичную подстановку. Аналогичным образом определяется $T_1^{out}(s)$ – множество вершин для исходящих дуг, а также множества $T_2^{in}(s)$ и $T_2^{out}(s)$ для второго графа. Отметим здесь, что в процессе сбора информации об окружении вершины m для того, чтобы метод $IsFeasible(n, surr\_m)$ быстрее отсекал недопустимые варианты, собирается информация о количестве исходящих и входящих дуг для вершин, включенных в концевые множества. В нашем варианте алгоритма мы дополнительно учитываем и количество двунаправленных дуг.

Метод $Find\_Node\_2()$ для выбора m осуществляет сначала поиск вершины в множестве $T_2^{out}(s)$, затем при отсутствии таковой в множестве $T_2^{in}(s)$ и, наконец, если ничего найти не удалось, среди оставшихся вершин, не включенных в $Core$. Метод $IsFeasible(n, surr\_m)$ считает вершину n неподходящим кандидатом на включение в $Core$, если хотя бы одна количественная характеристика о связях с вершинами из $Core$ у нее меньше, чем у вершины m. К таковым характеристикам мы отнесли:

- количество дуг из $Core$ в вершину;
- количество дуг из вершины в $Core$;
- количество двунаправленных дуг между $Core$ и вершиной;
- количество входящих, выходящих и двунаправленных дуг между вершиной и соответствующим множеством T^{in};
- количество входящих, выходящих и двунаправленных дуг между вершиной и соответствующим множеством T^{out}.

Мы также экспериментировали с дополнительными проверками на совместимость, учитывающими количество всех дуг без учета того, куда они ведут. Эти проверки никакого выигрыша не дали, поэтому мы от них отказались.

Если вершина n не была отвергнута, мы включаем пару (n, m) в $Core$, состояние s тем самым меняется, и рекурсивно вызываем $Match(s)$. После этого исключаем пару (n, m) из $Core$, восстанавливаем конечные множества в состояние до добавления n и продолжаем цикл по перебору вершин-кандидатов из первого графа.

Распараллеливание алгоритма

Описанный выше алгоритм представляет собой обход дерева вариантов совмещения вершин с постоянной проверкой возможности получения на очередной ветви корректной подстановки и отсечением неудачных вариантов. Ветви порождаются в процессе вычислений, и их трудоемкость не представляется возможным оценить заранее. Концепция рекурсивного распараллеливания [11] была разработана специально для таких задач и позволяет относительно легко справиться с возникающими при выстраивании параллельной стратегии проблемами.

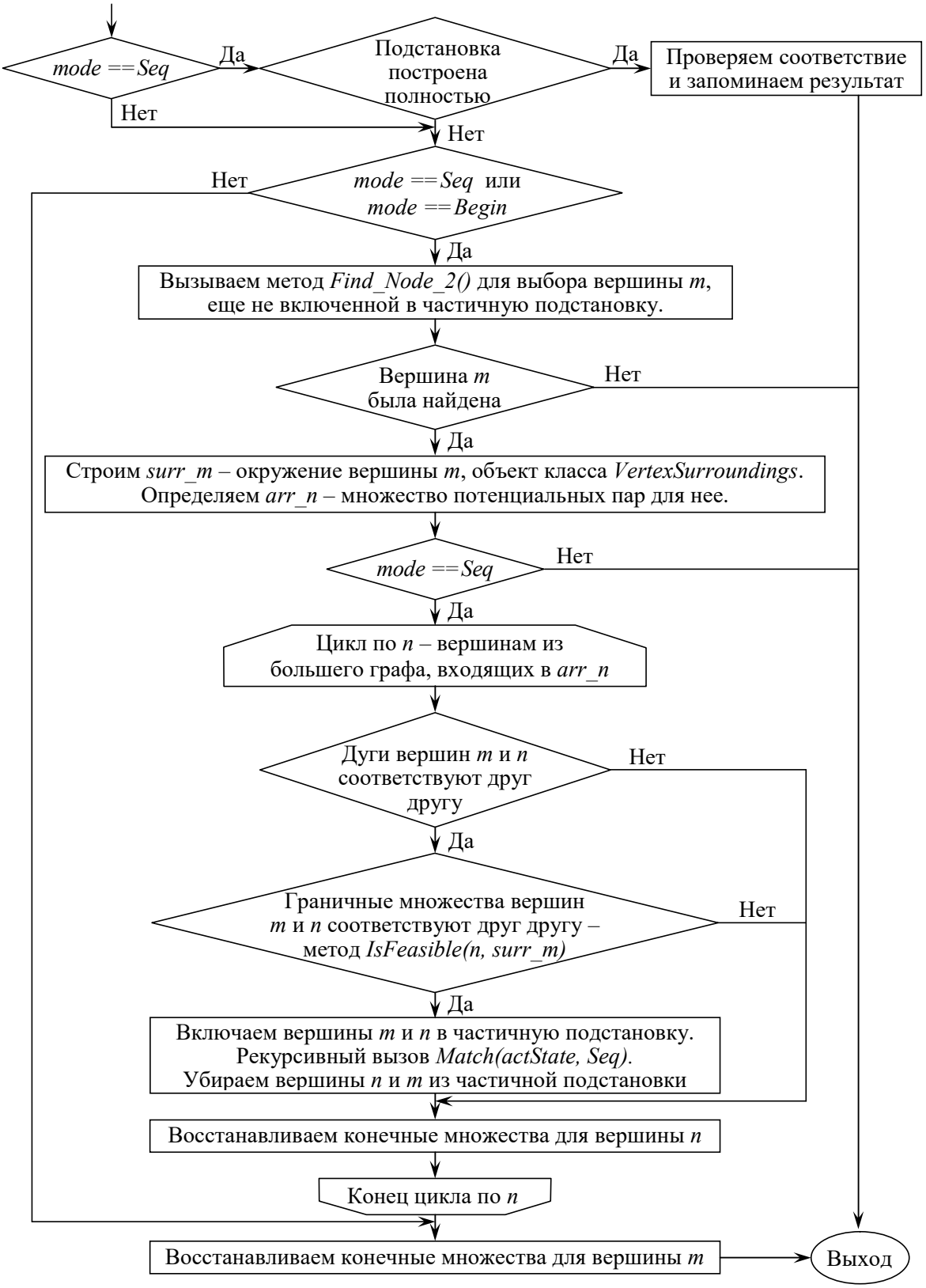

Fig. 2. Block diagram of parallel variant of the method *Match(actState, mode)*

Рис. 2. Блок-схема параллельного варианта метода *Match(actState, mode)*

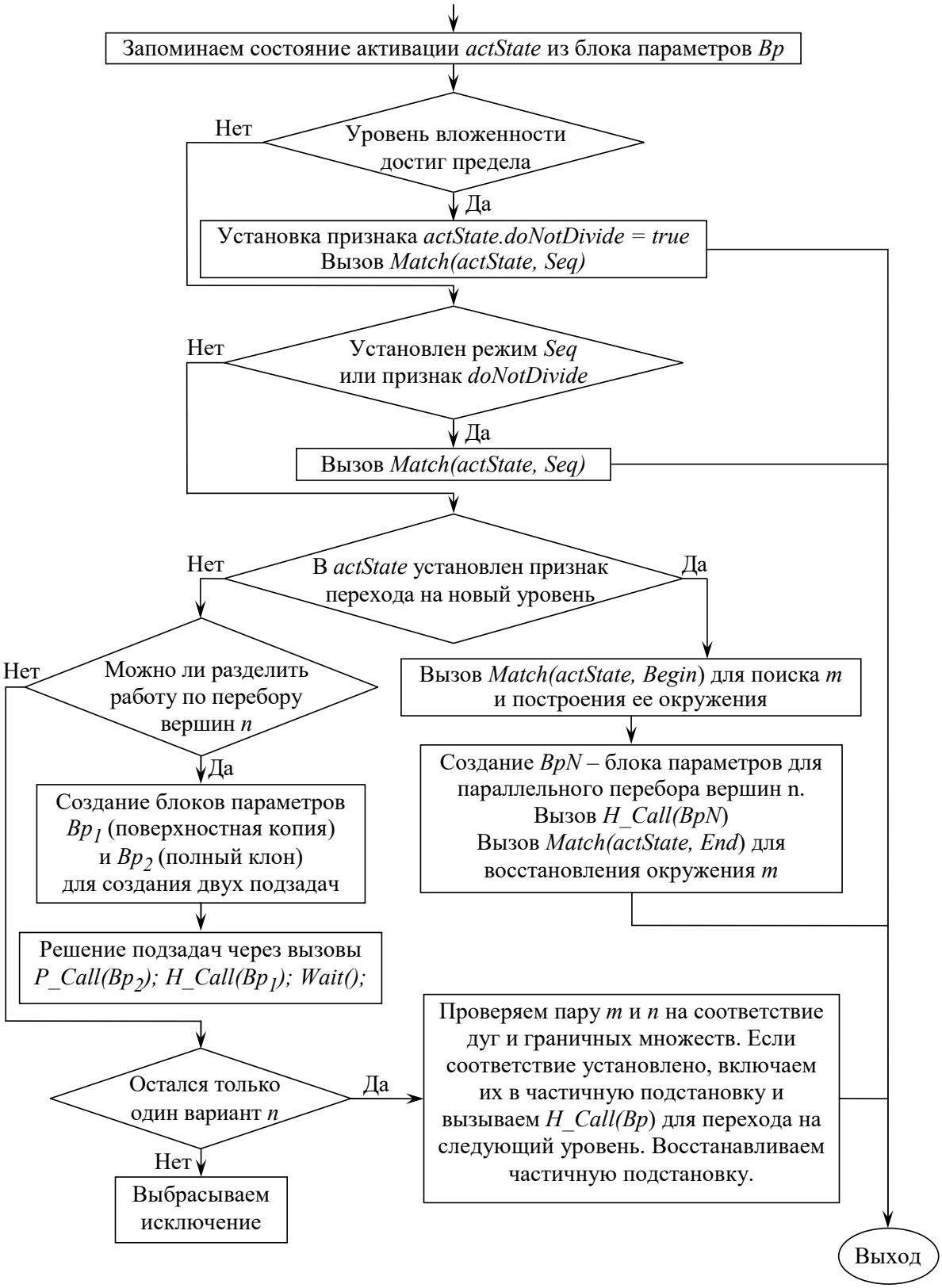

Fig. 3. Block diagram of the method *Parallel_VF2(Bp)*

Рис. 3. Блок-схема метода *Parallel_VF2(Bp)*

В нашем случае основным претендентом на распараллеливание представляется цикл по перебору n – вершин из большего графа внутри метода $Match(s)$. Оно легко может быть осуществлено в соответствии со стандартной РП-схемой [11], то есть рекурсивным делением пополам списка претендентов на роль n. При этом подзадача для обработки одной половины претендентов остается для решения на текущем процессорном модуле (ПМ), другая оформляется, как потенциально мигрирующая подзадача (ПМП) и при необходимости может быть передана для исполнения на другом ПМ. То, что порождаемые при этом подзадачи имеют, очевидно, очень разную и непредсказуемую трудоемкость, при применении РП-подхода не является проблемой. Библиотека поддержки рекурсивно-параллельного стиля программирования [14] обеспечивает достаточно равномерное распределение работы по системе на начальном этапе вычислений и при необходимости динамическое его перераспределение на последующих этапах. Требуется только создать достаточное количество ПМП.

Однако в отличие от стандартной схемы рекурсивного деления работы здесь присутствует вычисление $surr\_m$ – окружения вершины m, объекта, содержащего довольно большой объем информации, для сбора которой требуется порядка $O(N)$ операций. Собранная однажды, эта информация требуется при рассмотрении всех претендентов на роль n. Здесь приходится выбирать из двух вариантов: либо заниматься повторными вычислениями в каждой новой активации подзадачи, которая может выполняться на каком угодно ПМ, либо сохранять ее в блоке параметров и передавать как часть ПМП. После проведенных экспериментов предпочтение было отдано второму варианту.

Естественно, возникает вопрос о том, когда следует прекратить деление задачи и продолжить вычисления на данной ветви последовательно. Если этого не сделать, затраты на порождение параллельных подзадач и их оформление в виде ПМП сведут на нет (если не хуже) весь выигрыш от распараллеливания. Автором было принято самое естественное решение: ограничить глубину рекурсии, то есть по достижении некого значения $maxDepth$ (фактически это размер построенной части подстановки $Core$) мы переключаемся в последовательный режим.

Необходимость вычисления окружения вершины m и включения его в блок параметров, а также восстановления состояния активации на обратном ходе рекурсии потребовали внесения изменений в метод $Match(s)$. В нем были выделены три основных этапа вычислений: нахождение окружения вершины m до цикла по n, собственно цикл и восстановление состояния на обратном ходе. В параллельной версии метод $Match(actState, mode)$ имеет поэтому два параметра: объект, хранящий текущее состояние активации, и режим, определяющий, какие этапы следует выполнить. Предусмотрено три значения этого режима: выполняется только начальный этап, все этапы (последовательное выполнение) или только восстановление состояния после возврата из рекурсивного вызова метода $Match(s)$. Соответствующая блок-схема представлена на Рис. 2.

За порождение параллельных подзадач, их оформление в виде активаций, вызов (на данном вычислительном модуле или как ПМП), а также синхронизацию на обратном пути рекурсии отвечает метод $Parallel\_VF2(Bp)$. Его блок-схема представлена на Рис. 3.

Описание эксперимента и его результаты

Целью эксперимента была оценка эффективности работы предложенного параллельного алгоритма на сгенерированных случайным образом парах графов с различными характеристиками. В [18] предложена обширная база данных с разбитым по разным категориям набором графов именно для решения задач о граф-граф или граф-подграф изоморфизме. В ней представлены случайные графы различной плотности, регулярные и нерегулярные сети размерности 2D, 3D и 4D, а также регулярные и нерегулярные графы с ограниченной степенью вершины.

Мы исследовали работу нашего алгоритма на нескольких сотнях пар графов из этой базы. На исходных данных из некоторых групп даже последовательный алгоритм обеспечивал слишком быстрое получение результата, поэтому на них исследовать работу параллельного алгоритма

представлялось неинтересным. В результате мы отобрали те примеры исходных данных, время решения задачи на которых составляло для имевшихся в нашем распоряжении компьютерах несколько минут или несколько десятков минут. Далее будут представлены результаты решения задачи для двух категорий данных из этой базы:

- поиск всех изоморфизмов для графов на 1000 вершин с плотностью заполнения 0.1 (группа iso_r01_m1000);
- поиск всех граф-подграф изоморфизмов для пары графов размера 200 и 40 соответственно с плотностью заполнения 0.1 (группа $si2\_r01\_m200$).

Также автором была написана программа генерации пар случайных графов для данной задачи, которые удовлетворяли бы некоторым дополнительным условиям. На этих исходных данных также производилось исследование эффективности параллельного алгоритма. Ниже представлены результаты решения задачи для двух групп исходных данных:

- поиск всех граф-подграф изоморфизмов для пары регулярных графов размера 400 и 80 соответственно с плотностью заполнения 0.5 (группа $Gpi\_Regular\_400x50x80\_\_1$);
- поиск всех граф-подграф изоморфизмов для пары случайных графов размера 300 и 50 соответственно с плотностью заполнения 0.4 с гарантированным наличием не менее двух изоморфизмов (группа $Gpi\_Random\_300x0.4\_\_50x0.4\_\_2$).

В процессе тестирования использовались компьютеры на базе двухъядерного процессора Intel Core i3-7100 (максимальное количество потоков 4) с тактовой частотой 3.90 GHz и 8 GB оперативной памяти, работающие под управлением 64-разрядной ОС Windows 10. Пропускная способность сети равнялась 100 Mb/s. Как было отмечено выше одним из важных параметров, управляющих ходом вычислений, в нашем алгоритме является значение $maxDepth$ ограничивающее глубину рекурсии. В ходе эксперимента мы задали ему значение 2, что позволило создать достаточное для балансировки нагрузки количество подзадач и в то же время избежать неоправданных накладных расходов на их оформление в виде ПМП.

В таблицах приведены результаты вычислений для некоторых пар графов из упомянутых четырех групп. В них указано количество ветвей для порождаемого дерева поиска ($Branches$), время решения задачи последовательным алгоритмом и время решения параллельным алгоритмом при использовании нескольких компьютеров (РМ). Тот факт, что параллельный алгоритм даже на одном компьютере работал примерно вдвое быстрее последовательного, объясняется наличием двух ядер и многопоточной организацией вычислений. Можно также отметить заметную тенденцию к росту ускорения с увеличением объема вычислений, что позволяет рассчитывать на хорошую масштабируемость предложенного алгоритма.

Table 1. The Acceleration of the RP-algorithm on examples iso_r01_m1000

Таблица 1. Ускорение РП-алгоритма на примерах iso_r01_m1000

Source Data		**Sequently**	**1 PM**	**2 PM**	**4 PM**	**6 PM**	**8 PM**	**12 PM**	**16 PM**
1 (A77)	Exec. Time (s)	556.33	302.03	176.12	107.95	87.78	77.44	61.96	54.52
Branches (mln):	Ratio to Seq		***1.84***	***3.16***	***5.15***	***6.34***	***7.18***	***8.98***	***10.20***
119.25	Ratio to 1 PM		**1.00**	**1.71**	**2.80**	**3.44**	**3.90**	**4.87**	**5.54**
2 (A97)	Exec. Time (s)	642.94	337.16	279.02	170.72	154.45	122.18	102.47	89.20
Branches (mln):	Ratio to Seq		***1.91***	***2.30***	***3.77***	***4.16***	***5.26***	***6.27***	***7.21***
146.87	Ratio to 1 PM		**1.00**	**1.21**	**1.97**	**2.18**	**2.76**	**3.29**	**3.78**
3 (A72)	Exec. Time (s)	660.73	344.99	203.05	130.40	101.50	89.16	69.07	63.84
Branches (mln):	Ratio to Seq		***1.92***	***3.25***	***5.07***	***6.51***	***7.41***	***9.57***	***10.35***
136.06	Ratio to 1 PM		**1.00**	**1.70**	**2.65**	**3.40**	**3.87**	**4.99**	**5.40**
4 (A03)	Exec. Time (s)	779.31	404.12	241.05	150.40	116.62	94.34	75.58	65.37
Branches (mln):	Ratio to Seq		***1.93***	***3.23***	***5.18***	***6.68***	***8.26***	***10.31***	***11.92***
176.68	Ratio to 1 PM		**1.00**	**1.68**	**2.69**	**3.47**	**4.28**	**5.35**	**6.18**
5 (A92)	Exec. Time (s)	955.58	474.45	286.76	175.38	126.64	141.27	79.83	70.36
Branches (mln):	Ratio to Seq		***2.01***	***3.33***	***5.45***	***7.55***	***6.76***	***11.97***	***13.58***
169.38	Ratio to 1 PM		**1.00**	**1.65**	**2.71**	**3.75**	**3.36**	**5.94**	**6.74**
6 (A57)	Exec. Time (s)	1161.75	577.64	365.49	212.59	174.97	129.84	110.98	84.29
Branches (mln):	Ratio to Seq		***2.01***	***3.18***	***5.46***	***6.64***	***8.95***	***10.47***	***13.78***
271.67	Ratio to 1 PM		**1.00**	**1.58**	**2.72**	**3.30**	**4.45**	**5.20**	**6.85**
7 (A64)	Exec. Time (s)	1566.34	771.90	515.62	296.38	223.50	178.24	152.70	147.13
Branches (mln):	Ratio to Seq		***2.03***	***3.04***	***5.28***	***7.01***	***8.79***	***10.26***	***10.65***
356.51	Ratio to 1 PM		**1.00**	**1.50**	**2.60**	**3.45**	**4.33**	**5.06**	**5.25**
8 (A16)	Exec. Time (s)	1735.61	850.66	481.77	285.07	213.69	180.66	140.23	116.73
Branches(mln):	Ratio to Seq		***2.04***	***3.60***	***6.09***	***8.12***	***9.61***	***12.38***	***14.87***
367.71	Ratio to 1 PM		**1.00**	**1.77**	**2.98**	**3.98**	**4.71**	**6.07**	**7.29**
9 (A74)	Exec. Time (s)	1715.23	871.33	492.44	278.35	195.31	158.87	118.27	95.88
Branches (mln):	Ratio to Seq		***1.97***	***3.48***	***6.16***	***8.78***	***10.80***	***14.50***	***17.89***
371.62	Ratio to 1 PM		**1.00**	**1.77**	**3.13**	**4.46**	**5.48**	**7.37**	**9.09**
10 (A45)	Exec. Time (s)	1918.89	907.84	603.72	368.71	269.00	207.04	177.11	135.57
Branches (mln):	Ratio to Seq		***2.11***	***3.18***	***5.20***	***7.13***	***9.27***	***10.83***	***14.15***
397.53	Ratio to 1 PM		**1.00**	**1.50**	**2.46**	**3.37**	**4.38**	**5.13**	**6.70**

Table 2. The Acceleration of the RP-algorithm on examples si2_r01_m200

Таблица 2. Ускорение РП-алгоритма на примерах si2_r01_m200

Source Data		Sequently	1 PM	2 PM	4 PM	6 PM	8 PM	12 PM	16 PM
1 (A40)	Exec. Time (s)	711.20	385.67	204.40	108.65	81.57	63.72	52.62	47.49
Branches (mln):	Ratio to Seq		***1.84***	***3.48***	***6.55***	***8.72***	***11.16***	***13.51***	***14.98***
771.14	Ratio to 1 PM		**1.00**	**1.89**	**3.55**	**4.73**	**6.05**	**7.33**	**8.12**
2 (A68)	Exec. Time (s)	728.63	372.65	201.00	112.33	83.52	71.21	51.63	46.19
Branches (mln):	Ratio to Seq		***1.96***	***3.63***	***6.49***	***8.72***	***10.23***	***14.11***	***15.78***
802.31	Ratio to 1 PM		**1.00**	**1.85**	**3.32**	**4.46**	**5.23**	**7.22**	**8.07**
3 (A13)	Exec. Time (s)	782.61	410.75	211.30	118.89	85.21	79.62	58.66	47.23
Branches (mln):	Ratio to Seq		***1.91***	***3.70***	***6.58***	***9.18***	***9.83***	***13.34***	***16.57***
856.15	Ratio to 1 PM		**1.00**	**1.94**	**3.45**	**4.82**	**5.16**	**7.00**	**8.70**
4 (A88)	Exec. Time (s)	812.73	457.84	217.13	121.96	91.51	84.19	55.10	48.92
Branches (mln):	Ratio to Seq		***1.78***	***3.74***	***6.66***	***8.88***	***9.65***	***14.75***	16.61
898.74	Ratio to 1 PM		**1.00**	**2.11**	**3.75**	**5.00**	**5.44**	**8.31**	**9.36**
5 (A62)	Exec. Time (s)	833.84	423.90	220.64	120.87	95.26	83.36	57.88	51.68
Branches (mln):	Ratio to Seq		***1.97***	***3.78***	***6.90***	***8.75***	***10.00***	***14.41***	***16.13***
904.91	Ratio to 1 PM		**1.00**	**1.92**	**3.51**	**4.45**	**5.09**	**7.32**	**8.20**
6 (A73)	Exec. Time (s)	1099.85	578.71	297.23	155.17	111.92	97.58	68.71	59.10
Branches (mln):	Ratio to Seq		***1.90***	***3.70***	***7.09***	***9.83***	***11.27***	***16.01***	***18.61***
1 199.167	Ratio to 1 PM		**1.00**	**1.95**	**3.73**	**5.17**	**5.93**	**8.42**	**9.79**
7 (A52)	Exec. Time (s)	1160.73	557.99	293.32	156.87	120.87	92.00	67.15	62.34
Branches (mln):	Ratio to Seq		***2.08***	***3.96***	***7.40***	***9.60***	***12.62***	***17.28***	***18.62***
1 247.93	Ratio to 1 PM		**1.00**	**1.90**	**3.56**	**4.62**	**6.07**	**8.31**	**8.95**
8 (A32)	Exec. Time (s)	1218.52	616.86	308.85	170.32	121.14	103.42	69.23	60.69
Branches(mln):	Ratio to Seq		***1.98***	***3.95***	***7.15***	***10.06***	***11.78***	***17.60***	***20.08***
1 308.56	Ratio to 1 PM		**1.00**	**2.00**	**3.62**	**5.09**	**5.96**	**8.91**	**10.16**
9 (A90)	Exec. Time (s)	1361.54	691.69	351.73	184.57	135.65	122.36	77.83	64.71
Branches (mln):	Ratio to Seq		***1.97***	***3.87***	***7.38***	***10.04***	***11.13***	***17.49***	***21.04***
1 475.29	Ratio to 1 PM		**1.00**	**1.97**	**3.75**	**5.10**	**5.65**	**8.89**	**10.69**
10 (A35)	Exec. Time (s)	1531.67	749.91	388.59	206.69	146.86	113.03	86.31	78.30
Branches (mln):	Ratio to Seq		***2.04***	***3.94***	***7.41***	***10.43***	***13.55***	***17.75***	***19.56***
1 668.56	Ratio to 1 PM		**1.00**	**1.93**	**3.63**	**5.11**	**6.63**	**8.69**	**9.58**

Table 3. The Acceleration of the RP-algorithm on examples Gpi_Regular_400x50x80__1

Таблица 3. Ускорение РП-алгоритма на примерах Gpi_Regular_400x50x80__1

Source Data		Sequently	1 PM	2 PM	4 PM	6 PM	8 PM	12 PM	16 PM
1 (V05)	Exec. Time (s)	1333.11	727.86	372.66	188.56	139.69	110.83	78.00	66.59
Branches (mln):	Ratio to Seq		***1.83***	***3.58***	***7.07***	***9.54***	***12.03***	***17.09***	***20.02***
796.41	Ratio to 1 PM		**1.00**	**1.95**	**3.86**	**5.21**	**6.57**	**9.33**	**10.93**
2 (V03)	Exec. Time (s)	1344.30	740.92	375.28	193.61	137.74	116.74	82.62	66.09
Branches (mln):	Ratio to Seq		***1.81***	***3.58***	***6.94***	***9.76***	***11.52***	***16.27***	***20.34***
801.77	Ratio to 1 PM		**1.00**	**1.97**	**3.83**	**5.38**	**6.35**	**8.97**	**11.21**
3 (V08)	Exec. Time (s)	1354.75	750.12	379.48	203.57	164.28	121.22	82.53	66.22
Branches (mln):	Ratio to Seq		***1.81***	***3.57***	***6.65***	***8.25***	***11.18***	***16.42***	***20.46***
807.60	Ratio to 1 PM		**1.00**	**1.98**	**3.68**	**4.57**	**6.19**	**9.09**	**11.33**
4 (V01)	Exec. Time (s)	1319.81	696.24	357.22	191.56	137.92	124.07	81.89	66.55
Branches (mln):	Ratio to Seq		***1.90***	***3.69***	***6.89***	***9.57***	***10.64***	***16.12***	19.83
792.02	Ratio to 1 PM		**1.00**	**1.95**	**3.63**	**5.05**	**5.61**	**8.50**	**10.46**
5 (V07)	Exec. Time (s)	1386.27	738.58	367.57	193.48	138.88	109.58	81.70	64.77
Branches (mln):	Ratio to Seq		***1.88***	***3.77***	***7.17***	***9.98***	***12.65***	***16.97***	***21.40***
803.88	Ratio to 1 PM		**1.00**	**2.01**	**3.82**	**5.32**	**6.74**	**9.04**	**11.40**
6 (V00)	Exec. Time (s)	1343.85	732.08	366.71	200.75	140.48	123.63	81.51	65.26
Branches (mln):	Ratio to Seq		***1.84***	***3.66***	***6.69***	***9.57***	***10.87***	***16.49***	***20.59***
801.47	Ratio to 1 PM		**1.00**	**2.00**	**3.65**	**5.21**	**5.92**	**8.98**	**11.22**
7 (V04)	Exec. Time (s)	1346.08	731.88	368.80	192.22	136.93	121.54	80.59	67.68
Branches (mln):	Ratio to Seq		***1.84***	***3.65***	***7.00***	***9.83***	***11.08***	***16.70***	***19.89***
803.17	Ratio to 1 PM		**1.00**	**1.98**	**3.81**	**5.34**	**6.02**	**9.08**	**10.81**
8 (V06)	Exec. Time (s)	1352.78	728.87	368.90	190.83	135.72	123.45	82.09	69.15
Branches(mln):	Ratio to Seq		***1.86***	***3.67***	***7.09***	***9.97***	***10.96***	***16.48***	***19.56***
806.28	Ratio to 1 PM		**1.00**	**1.98**	**3.82**	**5.37**	**5.90**	**8.88**	**10.54**
9 (V02)	Exec. Time (s)	1486.93	766.66	406.07	220.99	152.59	127.02	84.95	70.14
Branches (mln):	Ratio to Seq		***1.94***	***3.66***	***6.73***	***9.74***	***11.71***	***17.50***	***21.20***
881.90	Ratio to 1 PM		**1.00**	**1.89**	**3.47**	**5.02**	**6.04**	**9.02**	**10.93**
10 (V09)	Exec. Time (s)	2757.84	1416.25	722.60	375.27	252.99	216.88	140.86	108.71
Branches (mln):	Ratio to Seq		***1.95***	***3.82***	***7.35***	***10.90***	***12.72***	***19.58***	***25.37***
1 631.54	Ratio to 1 PM		**1.00**	**1.96**	**3.77**	**5.60**	**6.53**	**10.05**	**13.03**

Table 4. The Acceleration of the RP-algorithm on examples Gpi_Random_300x0.4__50x0.4__2

Таблица 4. Ускорение РП-алгоритма на примерах Gpi_Random_300x0.4__50x0.4__2

Source Data		**Sequently**	**1 PM**	**2 PM**	**4 PM**	**6 PM**	**8 PM**	**12 PM**	**16 PM**
1 (V07)	Exec. Time (s)	597.71	326.64	176.33	99.33	75.61	62.60	49.74	43.26
Branches (mln):	Ratio to Seq		***1.83***	***3.39***	***6.02***	***7.91***	***9.55***	***12.02***	***13.82***
465.25	Ratio to 1 PM		**1.00**	**1.85**	**3.29**	**4.32**	**5.22**	**6.57**	**7.55**
2 (V01)	Exec. Time (s)	857.16	475.14	242.08	127.16	98.30	81.80	56.37	49.49
Branches (mln):	Ratio to Seq		***1.80***	***3.54***	***6.74***	***8.72***	***10.48***	***15.20***	***17.32***
668.94	Ratio to 1 PM		**1.00**	**1.96**	**3.74**	**4.83**	**5.81**	**8.43**	**9.60**
3 (V00)	Exec. Time (s)	891.87	472.17	252.58	133.73	97.87	84.28	58.61	55.11
Branches (mln):	Ratio to Seq		***1.89***	***3.53***	***6.67***	***9.11***	***10.58***	***15.22***	***16.18***
693.16	Ratio to 1 PM		**1.00**	**1.87**	**3.53**	**4.82**	**5.60**	**8.06**	**8.57**
4 (V02)	Exec. Time (s)	1283.61	672.86	354.47	179.13	129.66	112.08	77.30	60.59
Branches (mln):	Ratio to Seq		***1.91***	***3.62***	***7.17***	***9.90***	***11.45***	***16.61***	***21.19***
1 001.86	Ratio to 1 PM		**1.00**	**1.90**	**3.76**	**5.19**	**6.00**	**8.70**	**11.11**
5 (V03)	Exec. Time (s)	1393.92	715.05	381.60	197.44	139.02	139.02	81.60	64.96
Branches (mln):	Ratio to Seq		***1.95***	***3.65***	***7.06***	***10.03***	***10.03***	***17.08***	***21.46***
1 071.65	Ratio to 1 PM		**1.00**	**1.87**	**3.62**	**5.14**	**5.14**	**8.76**	**11.01**
6 (V04)	Exec. Time (s)	1615.98	854.22	440.69	227.83	161.15	137.82	91.46	71.15
Branches (mln):	Ratio to Seq		***1.89***	***3.67***	***7.09***	***10.03***	***11.73***	***17.67***	***22.71***
1 263.30	Ratio to 1 PM		**1.00**	**1.94**	**3.75**	**5.30**	**6.20**	**9.34**	**12.01**
7 (V08)	Exec. Time (s)	2507.05	1181.36	603.40	312.92	217.82	185.49	119.96	93.84
Branches (mln):	Ratio to Seq		***2.12***	***4.15***	***8.01***	***11.51***	***13.52***	***20.90***	***26.72***
1 800.63	Ratio to 1 PM		**1.00**	**1.96**	**3.78**	**5.42**	**6.37**	**9.85**	**12.59**
8 (V09)	Exec. Time (s)	2630.02	1349.70	683.68	344.68	237.55	227.27	130.56	106.76
Branches(mln):	Ratio to Seq		***1.95***	***3.85***	***7.63***	***11.07***	***11.57***	***20.14***	***24.63***
2 051.08	Ratio to 1 PM		**1.00**	**1.97**	**3.92**	**5.68**	**5.94**	**10.34**	**12.64**

References

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co, San Francisco, 1979.

[2] J. R. Ullmann, "An Algorithm for Subgraph Isomorphism", *Journal of the Association for Computing Machinery*, vol. 23, no. 1, pp. 31–42, 1976.

[3] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "Performance evaluation of the VF Graph Matching Algorithm", in *Proc. of the 10th ICIAP. IEEE Computer SocietyPress*, 1999, pp. 1172–1177.

[4] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "An Improved Algorithm for Matching Large Graphs", in *Proc. of the 3rd IAPR-TC-15 International Workshop on Graph-based Representations – Italy*, 2001, pp. 149–159.

[5] V. Carletti, P. Foggia, A. Saggese, and M. Vento, "Challenging the Time Complexity of Exact Subgraph Isomorphism for Huge and Dense Graphs with VF3", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 4, pp. 804–818, 2018.

[6] M. B. Il'yashenko, "Unificirovannyj podhod k resheniyu zadach morfizma na grafah", *Elektronnoe modelirovanie*, vol. 30, no. 1, pp. 19–41, 2008.

[7] M. B. Il'yashenko, "Reshenie zadachi poiska graf-podgraf izomorfizma dlya semanticheskogo analiza specializirovannyh cifrovyh sistem", *Nauchnye trudy DonTU. Seriya "Informatika, kibernetika i vychislitel'naya tekhnika"*, vol. 16, pp. 46–57, 2012.

[8] J. R. Ullmann, "Bit-Vector Algorithms for Binary Constraint Satisfaction and Subgraph Isomorphism", *Journal of Experimental Algorithmics*, vol. 15, no. 1, pp. 1–64, 2011.

[9] M. B. Il'yashenko, "Razrabotka i issledovanie parallel'nogo algoritma proverki graf-podgraf izomorfizma", *Radioelektronika. Informatika. Upravlenie*, vol. 1, pp. 63–69, 2006.

[10] V. Carletti, P. Foggia, P. Ritrovato, M. Vento, and V. Vigilante, "A parallel Algorithm for Subgraph Isomorphism", in *International Workshop on Graph-Based Representations in Pattern Recognition*, ser. LNCS, vol. 11510, Springer, Cham, 2019, pp. 141–151.

[11] V. V. Vasilchikov, *Sredstva parallelnogo programmirovaniya dlya vychislitelnykh sistem s dinamicheskoy balansirovkoy zagruzki*. YarGU, Yaroslavl, 2001.

[12] V. V. Vasilchikov, *Kommunikatsionnyy modul dlya organizatsii polnosvyaznogo soedineniya kompyuterov v lokalnoy seti s ispolzovaniem .NET Framework*, 2013.

[13] V. V. Vasilchikov, *Biblioteka podderzhki rekursivno-parallelnogo programmirovaniya dlya .NET Framework*, 2013.

[14] V. V. Vasilchikov, "On the recursive-parallel programming for the. NET framework", *Automatic Control and Computer Sciences*, vol. 48, no. 7, pp. 575–580, 2014.

[15] V. V. Vasilchikov, "On optimization and parallelization of the Little algorithm for solving the travelling salesman problem", *Automatic Control and Computer Sciences*, vol. 51, no. 7, pp. 551–557, 2017.

[16] V. V. Vasilchikov, "On a recursive-parallel algorithm for solving the knapsack problem", *Automatic Control and Computer Sciences*, vol. 52, no. 7, pp. 810–816, 2018.

[17] V. V. Vasilchikov, "Parallel algorithm for solving the graph isomorphism problem", *Modeling and analysis of information systems*, vol. 27, no. 1, pp. 86–94, 2020.

[18] P. Foggia, C. Sansone, and M. Vento, *A Database of Graphs for Isomorphism and Sub-Graph Isomorphism Benchmarkin*, 2001. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.6803&rep=rep1&type=pdf.