Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Системное программирование

Бзикадзе Александр Важевич

Отличия двух помеченных направленных ациклических графов с повторяющимися метками

Курсовая работа

Научный руководитель:
к.т.н., доц. Литвинов Ю.В.

Научный консультант:
Сотрудник ООО «Яндекс» Амосов Ф.А.

Санкт-Петербург
2018

Оглавление

Введение

Задачи на графах являются актуальной темой на текущий момент, так как большое множество объектов из областей математики и информатики, а также практических приложений данных наук, представляются в виде графов. Примером данного представления являются как такие элементарные примеры, как дорожная сеть, где перекрестки есть вершины графа, а дороги — ребра между ними, или отношение на множестве, где вершины представляют элементы рассматриваемого множества, а направленные ребра задают, находятся ли данные элементы в отношении, так и не совсем тривиальные, например, представление программы в виде потока управления [15]. Задача поиска отличий графов являет собой интереснейшую область исследований в связи с возможным приложением к более практическим проблемам, таким как поиск отличий двух алгоритмов, представленных в виде графов, разница UML-диаграмм и многим другим.

Задача поиска отличий двух графов является трудновычислимой. Впоследствии в этой работе будет показана NP-полнота данной задачи.

В данной работе будет рассмотрен частный случай данной задачи в условиях ацикличных ориентированных помеченных графов с повторяющимися метками, будут получены теоретические оценки и алгоритмы решения, а также приложения этих алгоритмов к поиску отличий графов неспециализированной платформы «Нирвана» [16] — внутренней разработки компании Яндекс для представления произвольных процессов в виде абстрактных графов. В связи с предполагаемой NP-полнотой рассматриваемой задачей предлагается рассматривать эвристические алгоритмы для поиска решения и адаптировать их для данной задачи.

1. Постановка задачи

Требуется решить задачу нахождения отличий двух ацикличных графов для абстрактных графов, а затем применить полученные результаты для платформы «Нирвана». Таким образом, планируется решить следующие задачи.

- Формализовать задачу для нахождения эквивалентных проблем
- Выяснить вычислительную сложность задачи
- Изучить существующие алгоритмы, реализовать некоторые из них и по возможности улучшить
- Изучить возможности обратимых преобразований графов платформы «Нирвана» в абстрактные графы с целью приложения существующего решения
- Опционально предложить некую визуализацию получаемых отличий

2. Обзор предметной области

Проведенная формализация задачи позволяет судить, что рассматриваемая проблема является эквивалентной задаче поиска наибольшего по количеству ребер изоморфных подграфов в данных графах (англ. MCESP) [8], расширенной до графов с помеченными вершинами. Подробное сведение к данной задаче будет рассмотрено далее.

Данная задача является NP-полной. Алгоритмы [8], традиционно применяемые для решения данной задачи основываются на сведении её к другим NP-полным задачам, таким как поиск максимальной клики [5]. Все подобные алгоритмы обладают экспоненциальной сложностью и в лучших случаях уменьшаются в сложности при увеличении разреженности графов [6].

В данной ситуации особый интерес представляют классические эвристические алгоритмы и их применения к данной задаче. Генетические алгоритмы [10], идея которых основана на естественном отборе, а именно отборе результирующей популяции и ее последующем развитии, известны значительной зависимостью от размерности искомого результата, потому не могут рассматриваться для решения задачи отличия двух графов. Другой классический эвристический метод — метод имитации отжига [14], идея которого заключается в поиске лучшего решения через вероятностный переход к «ближайшему» решению, часто используется для улучшения существующего достаточно хорошего решения, позволяя покидать локальные минимумы. Наиболее привлекательным приближенным решением является муравьиный алгоритм [9], подход к решению которого является итеративный выбор некого пути в зависимости от текущей привлекательности выбора и предыдущих итераций. Все рассмотренные алгоритмы имеют полиномиальную сложность от размеров входных графов и доказанные оценки сходимости на бесконечности.

2.1. Ant Algorithm for the Graph Matching Problem

Рассмотрим одну из спецификаций муравьиного алгоритма [11], используемую для решения Graph Matching Problem, являющуюся эквивалентной MCESP. Авторы работы приводят абстрактный алгоритм, но не приводят теоретических оценок времени исполнения. Наивная реализация предполагает зависимость в четвертую степень от количества вершин в графе. Тем не менее в ходе работы была выдвинута гипотеза о возможности квадратичной реализации, связи с чем алгоритм был принят к исследованию. Как показало дальнейшее исследование, гипотеза, вероятно, была неверна из-за сильной зависимости данных, из-за чего были ослаблены требования данного алгоритма на вероятностный выбор и получен квадратичный алгоритм.

2.2. A Simulated Annealing Algorithm for Maximum Common Edge Subgraph Detection in Biological Networks

Для уже упомянутой задачи MCESP существует ряд точных и приближенных решений. Рассмотрим некоторые из них. Существует специализация метода имитации отжига [12] для расширения данной задачи, когда вместо одного из сравниваемых графов рассматривается множество графов. В этой реализации возможно использование метода с различными законами изменения времени: классическим, линейным («быстрый» отжиг), адаптивный. Сложность алгоритма по утверждению авторов является в худшем случае квадратичной от количества вершин в графах. Авторы также приводят реализацию на языке C++. В данной работе будет предложена специализация метода имитации отжига, обладающая линейной сложностью, потому задача переиспользования существующего решения оставляется для дальнейшего исследования, но не исключается. Разумным действием будет написание интерфейса для данной реализации после реализации других более простых алгоритмов.

2.3. The maximum common edge subgraph problem: A polyhedral investigation

Другая работа предлагает точное решение задачи MCESP [13], рассматривая ее как задачу целочисленного линейного программирования. Так как решение точное, оно обладает экспоненциальной сложностью, но по утверждению авторов возможно использование на графах, количество вершин в которых равно 30 в среднем, а в крайних случаях не превосходит 40 вершин. В задачи отличия графов «Нирваны» ожидаемые размеры графов составляют большее количество. В связи с этим, а также с тем, что авторы не приводят собственную реализацию, данный вариант решения задачи решено оставить для дальнейшего исследования.

2.4. Fast Computation of Graph Edit Distance

Далее в работе будет показано, что рассматриваемая задача является специализацией проблемы GED, потому требуется рассмотреть существующие решения касательно данной задачи. Одна из последних работ на данную тему [1] предлагает решение, эффективно работающее на графах, количество вершин в которых не превышает 21. Авторы не приводят собственной реализации. По совокупности данных фактов принято решение исключить данную работу из рассмотрения как недостаточно эффективную, требующую реализации.

3. Теоретические исследование

3.1. Формализация задачи

Рассматриваются два ацикличных ориентированных помеченных графа, метки вершин которых не являются уникальными. Требуется определить «минимальное» преобразование одного графа в другой. Введем более формальное определение.

Определение 3.1. Ацикличный ориентированный помеченный граф с повторяющимися вершинами (Далее АОПГПВ) G — пара *(V, E)*, где V — множество вершин v, обладающих уникальными в совокупности свойствами *(label(v), num(v))*, а E — множество ребер (пар вершин).

Зафиксируем два АОПГПВ G_1, G_2. Рассмотрим пространство отображений $M{=}G_1 \to G_2$.

Определение 3.2. $\mathrm{m} = \{(v,\ u)\colon v \in Vertices(G_1),\ u \in Vertices(G_2),\ label(v) = label(u)\ \} \in M$ — отображение графа G_1 в граф G_2, если $(v,\ u_1) \in m,\ (v,\ u_2) \in m \implies u_1 = u_2$.

Отметим, что не для всех $v \in Vertices(G_1)$ $\exists!\ u$: $(v,\ u) \in m$ и аналогично для графа G_2. Введем понятия удаленных вершин — $deletedv(m) = \{v \in Vertices(G_1)\colon \nexists\ u \in Vertices(G_2),\ (v,\ u) \in m\}$, совпадающих вершин — $matchedv(m) = \{v \in Vertices(G_1)\colon \exists!\ u \in Vertices(G_2),\ (v,\ u) \in m\ \}$, добавленных вершины — $addedv(m) = \{u \in Vertices(G_2)\colon \nexists\ v \in Vertices(G_1),\ (v,\ u) \in \mathrm{m}\}$. Аналогичным образом введем понятия добавленных, совпадающих и удаленных ребер.

Введем частичный порядок на множестве M. $m_1 \in M$, $m_2 \in M$, $m_1 \leq m_2$, если

$$\begin{cases} |addedv(m_1)| + |deletedv(m_1)| \leq |addedv(m_2)| + |deletedv(m_2)|, \\ |addede(m_1) + deletede(m_1)| \leq |addede(m_2) + deletede(m_2)| \end{cases}$$

.

Тогда искомое минимальное отображение $m \in M$ — минимальное относительно введенного частичного порядка, то есть для любого $m' \in M$ $\Longrightarrow$ $m \leq m'$.

3.2. Минимальное отображение

Рассмотрим два АОПГПВ G_1, G_2. Рассмотрим пространство отображений $M{=}G_1 \to G_2$.

Утверждение 3.1. Отображение $m \in M$ является минимальным $\Longleftrightarrow$ m максимально относительно метрики $\rho(m') = |matchede(m')|$.

Покажем сначала, что при выборе минимального отображения однозначно определяется количество добавленных и удаленных вершин. Допустим, $m \in M$ — минимальное отображение и $\exists$ $v \in deletedv(m)$, $u \in addedv(m)$: $label(v) = label(u)$. Заметим, что принадлежность вершины множествам $deletedv$, $addedv$ означает удаление или добавление всех ребер, связанных с данной вершиной. Тогда можно построить отображение $m' = m \cup (v, u)$. Очевидно, что это будет корректное отображение, так как $(v, \_) \notin m$, $(\_, u) \notin m$, $label(v) = label(u)$. Тогда исходное отображение m не является минимальным, так как $|addedv(m')| + deletedv(m')| < |addedv(m)| + |deletedv(m)|$.

Таким образом, внутри множества вершин с одной меткой могут быть строго либо только добавления, либо только удаления. Для любой метки l графов G_1, G_2 очевидна следующая система (обозначим для краткости $mv = matchedv$, $dv = deletedv$, $av = addedv$, $V = Vertices$, $label(v) = l(v)$):

$$\begin{cases} |\{v \in mv(m) \cup dv(m) : l(v) = l\}| = |\{v \in V(G_1) : l(v) = l\}|, \\ |\{v \in mv(m) \cup av(m) : l(v) = l\}| = |\{v \in V(G_2) : l(v) = l\}| \end{cases}$$

Тогда система разрешима однозначно, что и требовалось. Очевидно, что из того, что результат получен для каждой метки двух графов, он верен для всех одновременно и размеры множеств *mv(m), dv(m), av(m)* определяются однозначно. Рассмотрим следующую систему:

$$\begin{cases} |matchede(m)| + |deletede(m)| = |Edges(G_1)|, \\ |matchede(m)| + |addede(m)| = |Edges(G_2)| \end{cases}$$

Очевидно, что $|addede(m)|$, $|deletede(m)|$ линейно зависят от $|matchede(m)|$ и уменьшаются при максимизации данного параметра. Таким образом, максимизация параметра $|matchede(m)|$ дает минимальность отображения m и наоборот.

3.3. Сведение к MCESP

Сведем рассматриваемую задачу к проблеме поиска максимального по числу ребер общего подграфа. Приведем формальную поставку задачи.

Определение 3.3. Maximum Common Edge Subgraph Problem: по двум графам найти общий подграф, не обязательно один, чье количество ребер максимально. Под общим подграфом двух графов понимается пара изоморфных подграфов данных графов соответственно.

Заметим, что в литературе встречаются различные постановки данной задачи, например, требующие связности графов и одинакового количества вершин [13]. Так как предметом изучения рассматриваемой в работе задачи являются графы направленные с помеченными вершинами, то под эквивалентностью MCESP подразумевается эквивалентность естественному обобщению данной проблемы: во-первых, рассматриваемые графы являются АОПГПВ; во-вторых, изоморфизмом АОПГПВ считается отображение m, такое что множества *addede*, *addedv*, *deletedv*, *deletede* пусты, иначе говоря изоморфизм графов, сохраняющий метку отображаемой вершины.

Было показано, что для нахождения минимального отображения необходимо и достаточно максимизировать метрику $\rho(m) = |matchede(m)|$. Из этого прямо следует эквивалентность задачи отличия двух помеченных направленных ацикличных графов с повторяющимися метками задаче поиска максимального по количеству ребер изоморфных подграфов.

Следствие 3.1. Рассматриваемая задача является эквивалентной MCESP

Следствие 3.2. Рассматриваемая задача является NP-полной.

3.4. Сведение к GED

Покажем, что рассматриваемая задача является специализацией задачи редакционного расстояния графов. Приведем формальную поставку задачи.

Определение 3.4. Graph Edit Distance: по двум графам с помеченными вершинами и ребрами найти оптимальную последовательность изменений одного графа, такую что под действием данной последовательности он преобразуется в изоморфный второму граф. Возможны следующие изменения графов: добавление, удаление вершины или ребра и изменение метки вершины или ребра. Оптимальность последовательности изменений $P = (o_1,...,o_n)$ определяется с использованием заданной функции веса изменений c_e:

$$\gamma(P) = \sum_{k}^{n} c_e(o_i)$$

В исходной постановки задачи требовалось найти минимальное отображение по количеству измененных (добавленных и удаленных) вершин. В таком случае, если для отображения m рассматривать множества *addedv*, *addede*, *deletedv*, *deletede* как последовательность изменений, а функцию веса c_e считать равной 1 на всех операциях кроме из-

менения метки, где функция равна нулю, то

$$\gamma(m) = \sum_{k}^{n} c_e(o_i) = |addedv(m)|+|deletedv(m)|+|addede(m)|+|deletede(m)|$$

Как показано ранее, для любого оптимального отображения значение $|addedv(m)| + |deletedv(m)|$ является известным, и требуется минимизировать оставшуюся часть $\gamma(m)$. В таком случае данная специализация GED эквивалентна рассматриваемой задаче. Тогда GED является обобщением рассматриваемой задачи.

Следствие 3.3. Рассматриваемая задача является специализацией GED

4. Описание решения абстрактной задачи

4.1. Используемые технологии

В качестве основного языка разработки был выбран Python3, как один из самых быстро развивающихся языков программирования, предназначенных в том числе для научных исследований. Для визуализации графов, обычных и «Нирваны», был выбран набор утилит Graphviz, использующий язык описания графов Dot [4]. Для использования данного инструментария добавлена зависимость от пакета pydot, являющегося интерфейсом на языке Python3. Для реализации алгоритмов, требующих высокоэффективных вычислений, используется язык C++. Для использования C++ с интерфейсом на Python используется пакет cppimport, являющейся надстройкой над пакетом Pybind11 [7]. Для распараллеливания вычислений на C++ используется стандарт OpenMP.

4.2. Реализованные алгоритмы

В ходе работы были реализованы несколько версий полного перебора, муравьиного алгоритма, а также метод имитации отжига.

4.2.1. Полный перебор

Полный перебор всех решений выполнен через рекурсивный проход через все множество решений, используя как порядок перебора вершины графа, минимального по количеству вершин из двух данных. Асимптотика данного решения экспоненциально зависит от максимального количества вершин с одной меткой в обоих графах. Эксперименты показали, что данный алгоритм не успел закончить работу за сутки на графах с данным показателем, превышающим 12 вершин.

4.2.2. Перебор с отсечениями

Основной идеей перебора с отсечениями является переупорядочевание перебираемых вершин и отсечение по верхней границе. Таким об-

разом, в первую очередь алгоритм сортирует входящие вершины обоих графов, чтобы изначально выбирались вершины с наибольшим количеством исходящих ребер. Данный выбор сильно сокращает верхнюю границу оставшегося перебираемого множества. Верхняя граница оставшегося множества перевычисляется после каждого выбора. Данный алгоритм обладает той же сложностью, что и полный перебор, но в некоторых очень специальных случаях работает намного быстрее, например, в случае двух изоморфных полных направленных ациклических графов алгоритм обладает линейной сложностью. Тем не менее, пространство таких специальных случаев, вероятно, сильно ограничено.

4.2.3. Неупорядоченный муравьиный алгоритм

Данный алгоритм основан на уже известных наработках в области применения муравьиного алгоритма для задачи поиска лучшего отображения графов. Таблица феромонов — необходимая часть любой специализации данного алгоритма — представляет собой отображение PhT: $\mathbb{N}_0^4 \to \mathbb{R}_+$, то есть для любых двух вершин первого графа u, v и двух вершин второго графа u', v' значение $PhT(u, u', v, v')$ означает привлекательность сопоставления вершин u, u' при условии, что сопоставлены вершины v, v'. Изначально PhT инициализировано единицей. Алгоритм выполняется итеративно, на каждой итерации объекты-первопроходцы вероятностно выбирают новые отображения графов, из которых выбирается лучший, после чего таблица феромонов обновляется. Для всех возможных пар сопоставлений вершин, выбранных на данной итерации, к значению в таблице феромонов добавляется $\frac{1}{1+gbs-cs}$, где gbs — количественная характеристика глобально лучшего на текущий момент выбора, а cs — выбора на текущей итерации. Как было показано прежде, количественной характеристикой является количество отображаемых ребер в данном отображении графов. После чего все значения в таблице умножаются на коэффициент выветривания феромона (1-p), где p — изначально заданный параметр, определяющийся экспериментально. Вероятностный выбор объектом-первопроходцем отображения графа происходит следующим образом: из множества всевозможных

сопоставлений вершин выбирается u, u' с вероятностью

$$\sum_{(v,v')\in CUR} PhT(u,u',v,v')SC(u,u')$$

и добавляется к текущему отображению графов CUR, где SC — количество вершин, добавляемых данным сопоставлением с учетом текущего отображения, после чего из множества всевозможных сопоставлений исключаются те, в которых присутствуют вершины u, u'. Данный процесс продолжается, пока множество всех сопоставлений не опустеет.

Представляется вероятным, что данный алгоритм не реализуем быстрее, чем за кубическую сложность относительно размеров графов. Для получения верхней оценки, достигаемой при изоморфных графах, будем считать, что количество вершин в первом графе V_1 равно аналогичному параметру второго графа V_2, обозначать который мы будем V. Максимальное возможное количество выборов есть V, каждый вероятностный выбор требует V^2 операций, так как это минимальная верхняя оценка для размера множества всех сопоставлений. Данное множество с каждым выбором уменьшается, потому можно записать общую сложность всех выборов $\sum_{k=1}^{V} V(V-k)$, также обладающую кубической асимптотикой. В связи с неоднородностью содержимого таблицы феромонов и неупорядоченностью выборов сопоставлений, такие приемы как корневая оптимизация и предподсчет частичных сумм не дают должных результатов.

Незатронутые в данных рассуждениях операции — обновление таблицы феромонов — обладают квадратичной сложностью при условии некоторых усовершенствований. Так как размер таблицы V^4 и ее обновление должно занимать столько же, принято решение поддерживать таблицу в ленивом состоянии, что возможно благодаря особенностям операции умножения на фиксированную константу. Пусть изначально таблица не хранит значений. Если запрашиваемое у нее значение не находится в данный момент в таблице, то его значение все равно известно

— $(1-p)^{it}$, где it — номер текущей итерации. При условии, что значение по адресу (u, u', v, v') находится в таблице, то возвращаемое значение есть

$$contained[u, u', v, v'](1-p)^{(it\ -\ last\_update[u,u',v,v'])}$$

где $last\_update[u, u', v, v']$ — последняя итерация, на которой значение обновлялось, а $contained[u, u', v, v']$ — это значение. В таком случае по завершении каждой итерации алгоритма требуется обновлять только те значения, которые содержатся в выбранном объектами-первопроходцами отображении, что обладает квадратичной сложностью.

Таким образом простейшая идея ленивой инициализации и обновления позволяет ускорить алгоритм. Заметим, что идея, примененная к таблице феромонов, обобщается на случай отличной от четвертой размерности таблицы. Единственная часть реализации данной таблицы, не поддерживающая данное обобщение, — обновление значений, так как зависит от детализации самого алгоритма.

4.2.4. Упорядоченный муравьиный алгоритм

Кубическая сложность алгоритма в случае графов превышает разумное время выполнения алгоритма для больших графов, потому решено отказаться от свойства неупорядоченности выбора в пользу заданного порядка для получения квадратичной сложности. Отображение на каждой итерации строится с нуля, потому неразумно рассчитывать на сложность, меньшую чем квадратичная, в общем случае (для вычисления количественной характеристики требуется минимум $O(V^2)$ в случае полного графа). Тогда рассматриваемая таблица феромонов будет отображением PhT: $\mathbb{N}_0^2 \to \mathbb{R}_+$ и означать для каждых двух вершин u, u' двух графов соответственно привлекательность их сопоставления. Так как выбор сопоставлений происходит в определенном порядке, осуществлять вероятностный выбор требуется из множества размером $O(V)$, а количество выборов — $O(V)$. В качестве одной из опций алгоритма предложено предоставлять дополнительную вероятность выбора

вершин, близких по показателю количества входящих и исходящих вершин, что является полезным в случае, если заранее известно, что один граф является модификацией другого. В остальном данный алгоритм повторяет неупорядоченный. Дальнейшие эксперименты показывают, что данный алгоритм работает сравнительно точно.

4.2.5. Метод имитации отжига

Метод имитации отжига основан на идее малого вероятностного изменения текущего ответа с целью покидания локальных минимумов в зависимости от текущей температуры. Параметрами алгоритма являются закон изменения температуры, начальная температура (определяемая экспериментально) и, наиболее важный параметр, способ изменения текущего ответа. В данной работе решено рассматривать закон изменения температуры, называемый «быстрым отжигом» — гиперболическая зависимость температуры от времени, как наиболее распространенный. Начальная температура, требуемая алгоритмом, определена с помощью существующих оценок [2]. Было рассмотрено два варианта малого изменения выбранного отображения — случайное пересопоставление одной вершины и случайная перестановка в рамках вершин с одной меткой. Последний способ показывает свою несостоятельность в случае графов с одинаковыми метками, так как в данном случае метод вырождается в случайный выбор отображения. Выбранный в работе метод позволяет для случайно выбранной вершины сопоставить случайно выбранную вершину второго графа, изменяя текущее отображение максимум для двух вершин (в случае если сопоставляемая вершина уже выбрана для сопоставления с другой вершиной первого графа). Такое изменение позволяет эффективно за линейное время выбирать обе вершины, а также осуществлять пересчет количества совпадающих ребер. Тогда метод в данной реализации позволяет вычислять решение за линейное время относительно размера графов.

Эксперименты показывают больший рост процента ошибок по сравнению с упорядоченным муравьиным алгоритмом, тем не менее важной

особенностью данного метода является возможность использовать его в связанных вычислениях, принимая за начальное решение отображение, данное другим алгоритмом, по возможности улучшая его.

4.3. Структура решения

Каждый алгоритм является реализацией общего интерфейса *GraphDiffAlgorithm*, обладающего методом *construct_ diff*, возвращающим объект *GraphMap*, описывающий отображение одного графа в другой и предоставляющий методы исследования данного отображения. Помимо этого предоставляется интерфейс *GraphDiffAlgorithmWithInit*, расширяющий предыдущий посредством возможности запуска представляемого алгоритма с учетом переданного начального решения. Таким образом осуществляется композиция разных алгоритмов с целью уточнения получаемого отображения.

Композиция различных алгоритмов реализуется специальным классом-композитором *ComposedGraphDiffAlgorithm*. Стратегия композиции осуществляется классической *reduce*-композицией.

Алгоритмы, реализуемые на C++, предоставляются как подклассы *CppImport*. Имена данных алгоритмов представлены в конфигурационном файле и загружаются в приложение вместе с загрузкой класса-контейнера. Одними из рассматриваемых алгоритмов являются версии муравьиного алгоритма. Они обладают общим принципом работы, отличаясь в способе поиска отображения объектами-первопроходцами и таблицей феромонов. В связи с этим муравьиный алгоритм представлен шаблонным классом, принимающим класс первопроходца как аргумент. Последний помимо реализации поиска отображения содержит наименование класса, который следует использовать в данном алгоритме как таблицу феромонов. Так как последние различаются только в способе обновления внутренней таблицы по завершении итерации, то они являются наследниками обобщенной таблицы феромонов, прини-

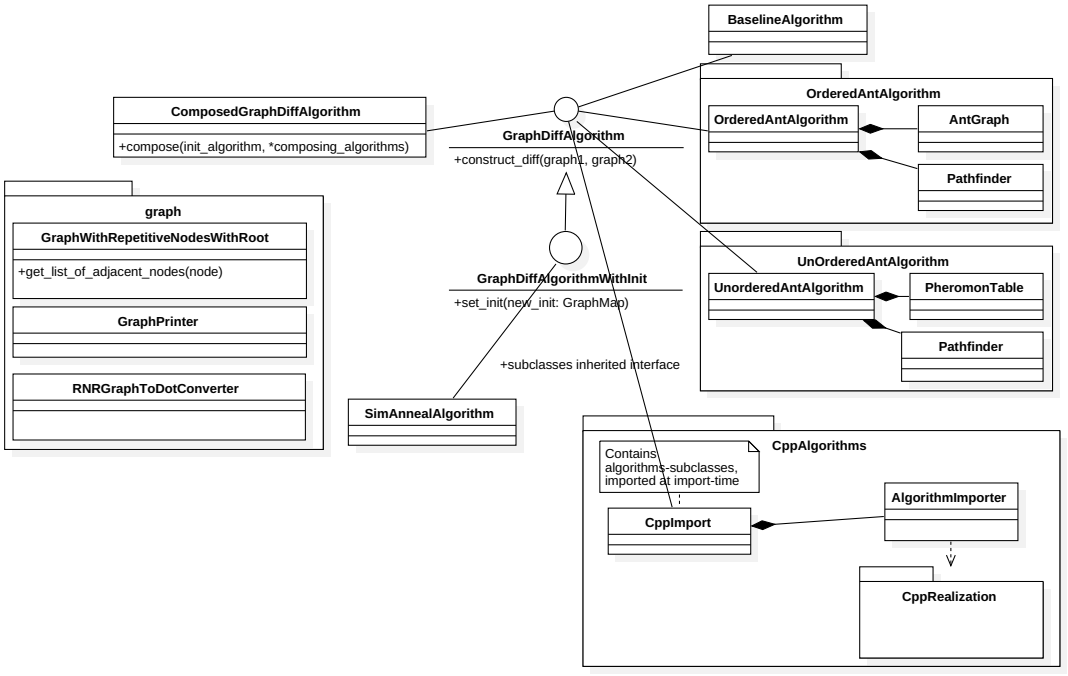

Рис. 1: Диаграмма классов

мающей переменное количество шаблонных аргументов.

Приведем упрощенную диаграмму классов, исключающую почти все методы, кроме основных, а также не специализирующую диаграмму классов алгоритмов на языке C++.

5. Применение абстрактной задачи для поиска отличий графов «Нирваны»

5.1. Требования

Граф «Нирваны» представляет собой поток работ (англ. *workflow* [3]), состоящий из блоков и связей между ними. Блоки представляют из себя операцию, набор неких key/value значений, а также входов и выходов по данным. Связь между блоками может быть по выполнению, то есть представлять собой зависимость только по времени исполнения, а может быть по данным — в таком случае такая связь характеризуется данными, а у связанных блоков имеются соответствующие вход и выход.

Требуется найти отличия таких графов с заданной точностью. Вариантами такой точности являются:

- Определение изменений факта наличия связи между парами блоков, удаления и добавления блоков
- Определение изменений связей с сохранением информации о полном типе связи, изменения блоков.

5.2. Преобразование графов «Нирваны»

Рассмотрено два способа преобразования графов «Нирваны» в абстрактные — полное и простейшее. Простейшее преобразование отображает каждый блок в соответствующую вершину, теряя информацию о типах связей между блоками. Такое преобразование является необратимым, но не добавляет дополнительных вершин, а также удовлетворяет первому случаю требований. Полное преобразование сопоставляет блоку набор вершин, связанных звездообразной топологией, представляющих собой операцию, входы и выходы по данным. Связи по данным между блоками отображаются в ребра между соответствующими

вершинами-данными, а связи по выполнению соединяют элементы операции. Очевидно, что подобное преобразование является обратимым при должном преобразовании вершин, сохраняющим всю информацию о блоке, которое не представляет научного интереса и является частной технической задачей.

После преобразования происходит поиск отличий полученных графов, после чего встает вопрос обратного преобразования. В ходе полного преобразования рассматривается каждая вершина-операция, и собирается информация о ее соседях-данных, при этом в отображенном смысле. Их наличие или отсутствие позволяет определить изменения, произошедшие с рассматриваемым блоком. Очевидным образом по изменениям вершин определяются изменения связей между блоками. Обратное преобразование в случае простейшего преобразования графов «Нирваны» является очевидным в силу простоты самого преобразования.

5.3. Визуализация

Для визуализации используется язык описания графов Dot. Блоки представлены вершинами формы record прямоугольной формы, разделенный на три ряда, представляющих собой в движении сверху вниз ряд входных данных, операции, выходных данных. Крайние ряды разделены на ячейки, представляющие конкретные входы и выходы данных. В ряд выходных данных включена специальная ячейка для соединения по выполнению. Связи соединяют требуемые участки вершин в зависимости от типа связи. Так зависимости по данным соединяют входные и выходные участки соответствующих вершин, а связи по выполнению проведены от специальной ячейки одного блока до аналогичной ячейки другого блока. В результате работы создается совмещенный граф «Нирваны» и способ его раскраски с заданными константами-цветами, являющимися ожидаемым способом раскраски отличий: зеленый для добавления, красный для удаления, черный как стандартный

цвет языка *Dot* для элементов без изменений.

5.4. Структура решения для графов «Нирваны»

Для удобства использования был реализован класс *Pipeline*, параметризуемый алгоритмом и способом преобразования графа «Нирваны». Данный класс обладает методами *get_ diff*, *get_ dot_ diff*, *print_ diff*. Первый принимает два графа «Нирваны» и возвращает совмещенный граф и объект-раскраску данного графа. Остальные два метода создают и печатают граф в формате языка *Dot* соответственно.

6. Сценарии использования

6.1. Абстрактные графы

Пользователю требуется найти отличия двух графов. Тогда ему требуется решить вопрос, какой из алгоритмов использовать. В случае малых графов рекомендуется использовать распараллеленный перебор с отсечениями. В случае больших графов рекомендуется по умолчанию использовать упорядоченный муравьиный алгоритм в композиции с методом имитации.

Рассмотрим задачу проверки изоморфизма достаточно крупных графов и ее решение. Пользователю требуется по данным связным *graph1*, *graph2* запустить базовый алгоритм с отсечениями с реализацией на языке C++. Результатом запуска является объект *GraphMap*. Если графы изоморфны, то множества добавленных и удаленных ребер должны быть пусты. Так как запрос на множества добавленных и удаленных вершин не является стандартным, то объекту *GraphMap* требуется предварительно запустить метод *eval_ difference_ complete()* для завершения исследования.

```
# graph1, graph2

from graph_diff.cpp_algorithms.algorithms import CppImport

algo = CppImport.BaselineWithChopAlgorithmOmp()
diff = algo\
    .construct_diff(graph1, graph2)\
    .eval_difference_complete()
assert len(diff.get_edges_in_1_not_in_2()) \
       + len(diff.get_edges_in_2_not_in_1()) == 0
```

6.2. Графы «Нирваны»

Пользователю требуется найти отличия двух графов «Нирваны». Тогда ему требуется решить не только вопрос, какой из алгоритмов использовать, но и какой способ преобразования графов выбрать. По умолчанию предлагается использовать полное преобразование в виду обратимости. Тем не менее пользователь может воспользоваться возможностью простейшего преобразования в случае, если, например, рассматриваемые графы чрезмерно большие.

Рассмотрим задачу печати отличий двух графов «Нирваны». Требуется создать объект *CompleteWorkflowToGraphConverter* для определения типа преобразования. Затем создается алгоритм-композиция упорядоченного муравьиного алгоритма и метода имитации отжига. После чего создается объект *Pipeline*, аргументами которого являются тип преобразования и алгоритм. Затем по данным *workflow*, *workflow_update* требуется запустить метод указанного объекта *print_diff* с переданным путем сохранения результата.

```
# workflow, workflow_update

from graph_diff.nirvana_object_model \
    import Pipeline
from graph_diff.graph_diff_algorithm \
    import ComposedGraphDiffAlgorithm
from graph_diff.cpp_algorithms.algorithms \
    import CppImport
from graph_diff.simulated_annealing_algorithm \
    import SimAnnealAlgorithm
from graph_diff.nirvana_object_model\
    .workflow_to_graph_converter \
    import CompleteWorkflowToGraphConverter

converter = CompleteWorkflowToGraphConverter()
```

```
algorithm = ComposedGraphDiffAlgorithm(
    CppImport.OrderedAntAlgorithm(),
    SimAnnealAlgorithm())
Pipeline(algorithm=algorithm,
         workflow_converter=converter)\
    .print_diff(workflow,
                workflow_update,
                path=chosen_path)
```

Рассмотрим задачу импортирования графов «Нирваны». В связи с закрытой структурой данной системы, графы предоставляются в сериализованном виде json-файлов, и для импортирования используется фукнция *deserialize*, реализованная не в рамках данной работы.

```
# filename
import json
from graph_diff.nirvana_object_model import workflow_deserializer


with open(filename) as f:
    json_input = json.loads(f.read())
    workflow = workflow_deserializer.deserialize(json_input)
```

7. Эксперименты

Так как сложность и точность алгоритмов полного перебора не составляет сомнения, эксперименты были проведены только на эвристических алгоритмах. В качестве самой трудновычислимой специализации рассматриваемой проблемы была выбрана задача проверки изоморфизма полных графов с одной меткой. Во-первых, подобное задание графов простейшим образом сводится к зависимости от количества вершин в графе. Во-вторых, рассмотрение задачи изоморфизма позволяет перейти от двумерного пространства к одномерному, что в свою очередь дает большую наглядность для оценок. Эксперименты в выбранной области предоставляют оценку сверху для многих характеристик, из которых будут оцениваться время работы и процент ошибочного сопоставления, т.е. отношение разности требуемого количества сопоставленных ребер (количества ребер в полном графе) и числа действительно сопоставленных ребер к требуемого количеству. Следующие алгоритмы были подвергнуты тестированию: неупорядоченный и упорядоченный муравьиные алгоритмы, композиция упорядоченного муравьиного алгоритма и метода имитации отжига, метод имитации отжига. Для краткости занумеруем представленные алгоритмы от 1 до 4 в предложенном порядке. В связи с ранее предоставленными теоретическими оценками сложности алгоритм 1 тестировался до 40 вершин в графе, в то время как остальные до 100. Данное тестирование проведено с шагом в 10 вершин и с размером выборки в 100 тестов.

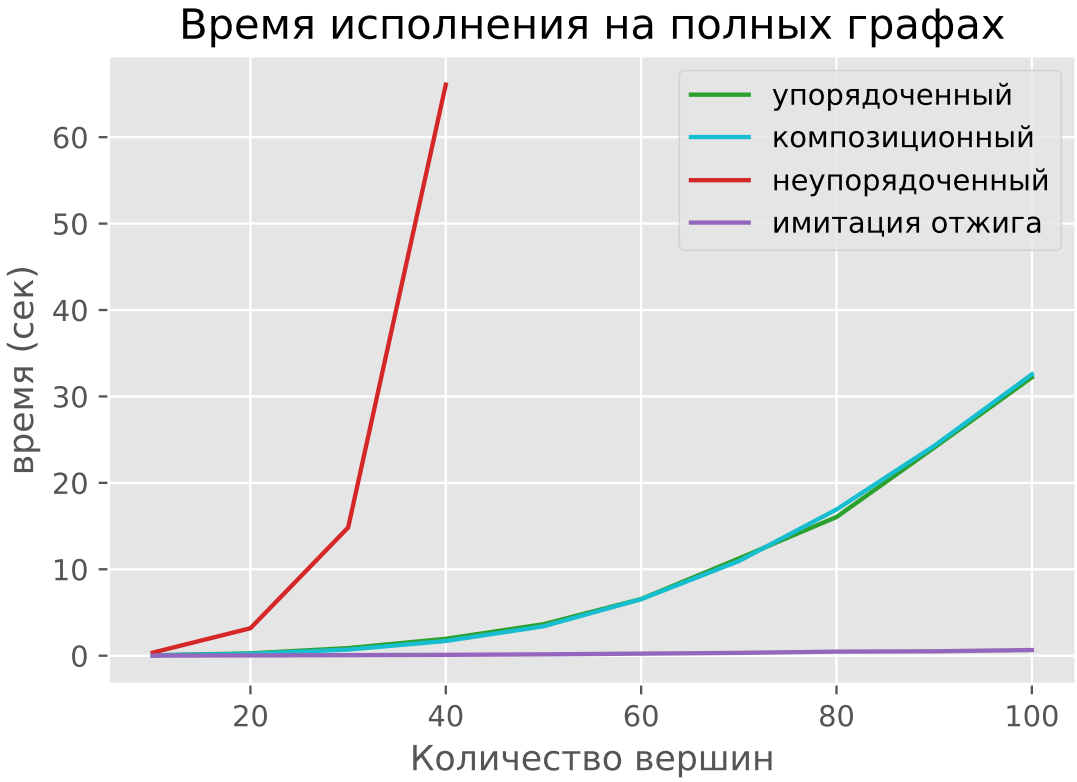

Рис. 2: Время работы алгоритмов

Алгоритм 1 показывает очень быстрый рост времени исполнения, из-за чего дальнейшее точное исследование его работы не представляется возможным. В то же время время работы алгоритма 4 является пренебрежительно малым в сравнении с временем работы алгоритмов 2,3. Приведем показатели максимальных выборочных и усеченных (0.1, 0.1) дисперсий рассматриваемых данных.

Алгоритм	Выборочная дисперсия	Усеченная дисперсия
(1) Неупорядоченный	56411.105570062	10.080270244697
(2) Упорядоченный	12.223421995	6.958452753547
(3) Композиционный	11.757216180	4.896001973837
(4) Имитация отжига	0.034229817	0.00621532293

Таким образом полученные оценки времени можно считать достаточно точными.

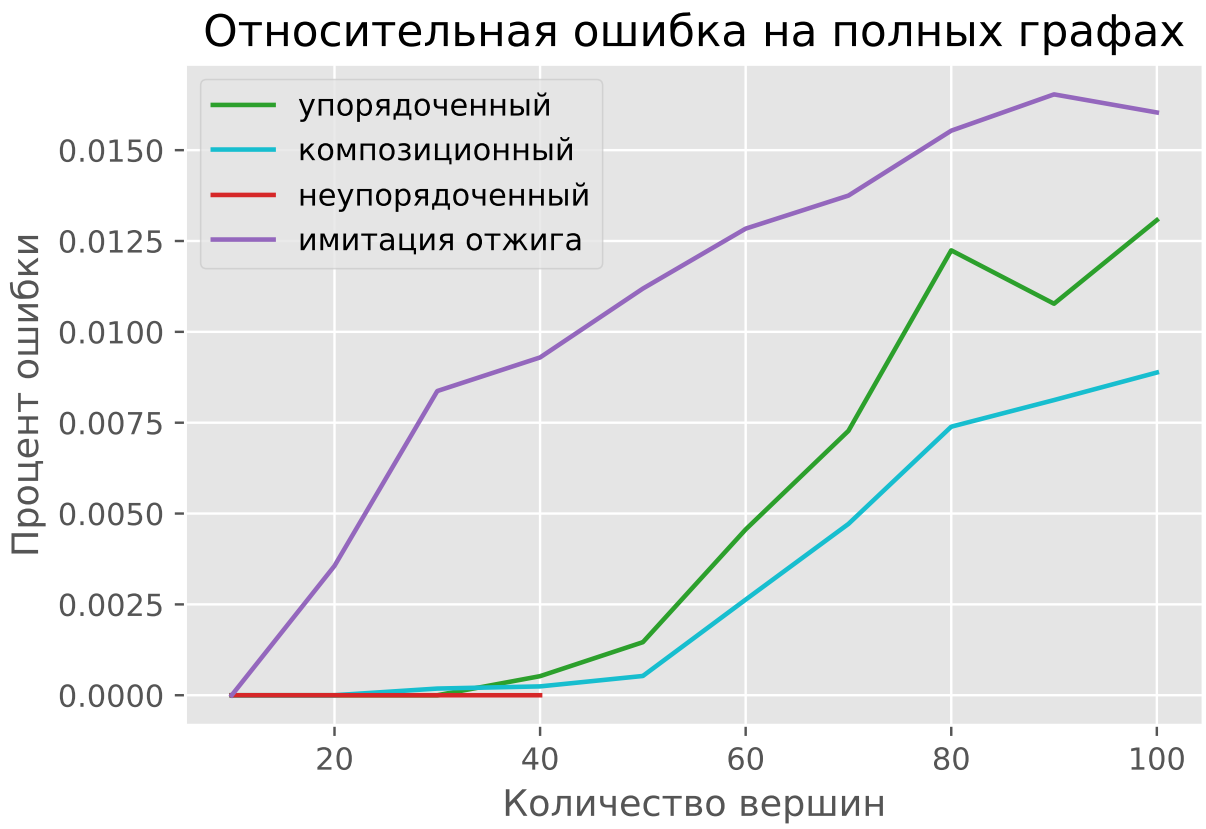

Рис. 3: Относительная точность алгоритмов

Рассмотрим полученные оценки точности.

Процент ошибки всех рассматриваемых алгоритмов составляет менее 1.75%. К сожалению, невозможно сравнить полученные данные с результатами неупорядоченного муравьиного алгоритма [11]. Для наличия какого-либо сравнения отметим, что авторы неупорядоченного муравьиного алгоритма заявляют, что их процент совпадений на тестовых данных в среднем составляет 90%, но размеры рассматриваемых графов превышают 100 вершин, хоть и не более, чем в два раза, и обладают метками на ребрах. Наиболее интересным фактом является существенное улучшение композиционного алгоритма в сравнении с отдельно взятыми его составляющими. Рассмотрим разбросы полученных значений.

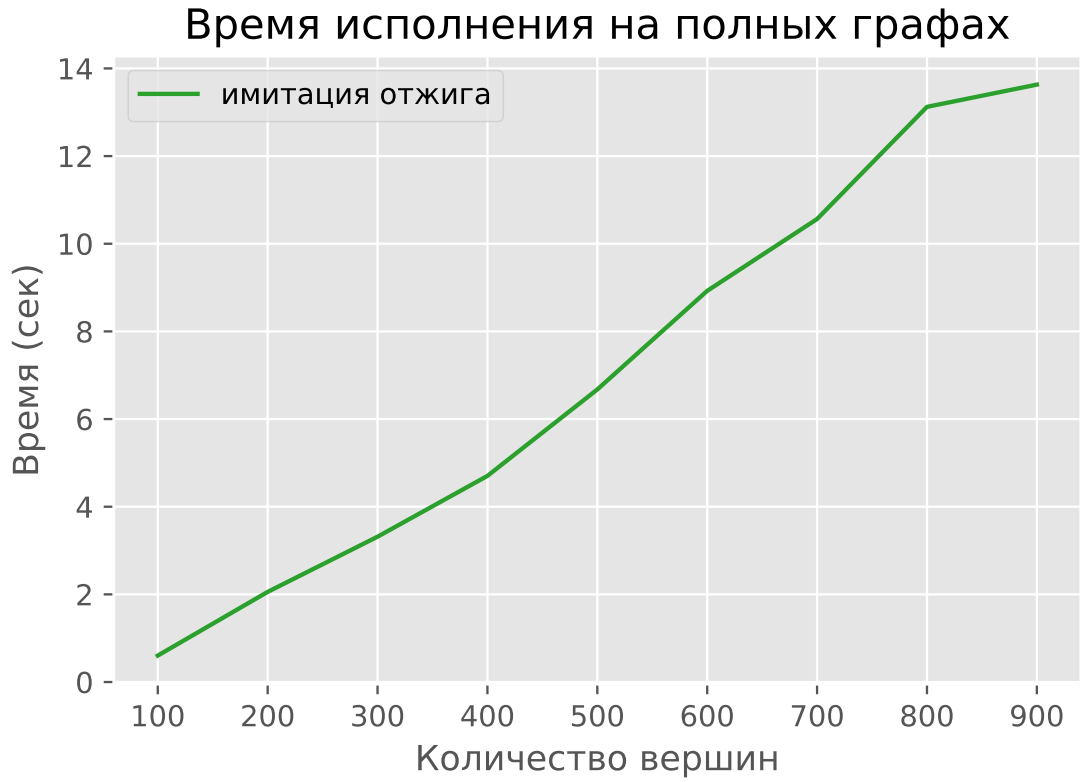

Рис. 4: Время работы метода имитации отжига

Алгоритм	Выборочная дисперсия	Усеченная дисперсия
(1) Неупорядоченный	0.0	0.0
(2) Упорядоченный	0.001325	0.000057
(3) Композиционный	0.000045	0.000025
(4) Имитация отжига	0.000035	0.000016

Таким образом композиция алгоритмов (2), (4) дает существенное увеличение точности, в то время как время работы композиционного алгоритма (3) и алгоритма (2), добавляющего основное время работы, является пренебрежимо малым. Так максимальное различие среднего времени работы, зафиксированное во время экспериментов, составляет 0.872238 секунд, когда максимальное время работы во время экспериментов обоих алгоритмов превышает 30 секунд.

В ходе дальнейших экспериментов с алгоритмом 4 получено достижение верхней оценки времени работы алгоритма от размера графа — линейной. Более того, максимальное зафиксированное время работы не превосходит 15 секунд, что позволяет использовать данный алгоритм для вычислений на больших графах в ситуациях, когда точность ответа не является приоритетом в сравнении с задачей его нахождения.

Экспериментальная оценка точности также дает линейную зависимость от размера графа и не превышает ошибки в 10 процентов, что хоть и не является достижением, позволяет получать некоторое приближение, приемлемое в некоторых ситуациях.

Тестирование проводилось на машине с одним процессором Intel Core i5 с тактовой частотой 2,7 GHz, двумя ядрами, разделенными на два логических ядра, восьмью ГБ оперативной памяти DDR3.

8. Заключение

- Абстрактная задача была формализована и сведена к известной задаче поиска общего максимального подграфа
- Изучена предметная область для применения эвристических алгоритмов в решении задачи
- Реализован наивный алгоритм, точно решающий задачу, но не проходящий любые разумные временные пороги, и его оптимизационные модификации, обладающие аналогичными временными свойствами
- Реализованы эвристические алгоритмы, в том числе неупорядоченный муравьиный алгоритм и алгоритм имитации отжига
- Разработан упорядоченный муравьиный алгоритм, не обладающий теоретическими подтверждениями сходимости к точному ответу при бесконечном числе итераций, но соответствующий всем общим принципам построения алгоритмов данного типа
- Предоставлена возможность связывать вычисления различных алгоритмов в композицию для повышения точности вычислений
- Был создан способ использовать существующие средства для визуализации абстрактных графов и их отличий
- Было проведено тестирование реализованных алгоритмов на специализированной задаче сопоставления полных графов самим себе (Стоит отметить, что так как алгоритмы эвристические и стремятся получить лишь приближение ответа, не следует использовать данные методы для задачи поиска изоморфизма, так как для этого существуют более специализированные алгоритмы, учитывающие особенности данной задачи)
- Эксперименты подтвердили теоретические оценки сложности алгоритмов, и показали возможность использования разработанно-

го упорядоченного муравьиного алгоритма для рассматриваемой задачи для графов таких размерностей, для которых неупорядоченная версия требует неразумное время

- Вместе с тем экспериментально показана целесообразность использования метода имитации отжига для улучшения точности вычислений упорядоченного муравьиного алгоритма
- Было описано два преобразования графов платформы «Нирвана», одно из которых полное, обратимое, а второе необратимое, но менее трудоемкое, сохраняющее лишь факт наличие связи между вершинами графов платформы «Нирвана», но не их тип
- Полученные результаты были применены для нахождения отличий графов платформы «Нирвана»
- Предоставлена возможность визуализации получаемых результатов

Исходный код проекта доступен в открытом доступе: `https://github.com/alexander-bzikadze/graph_diff`.

Список литературы

[1]

[2] Ben-Ameur Walid. Computing the Initial Temperature of Simulated Annealing // Computational Optimization and Applications. — 2004. — Dec. — Vol. 29, no. 3. — P. 369–385. — URL: https://doi.org/10.1023/B:COAP.0000044187.23143.bd (online; accessed: 13.05.2018).

[3] Curcin Vasa, Ghanem Moustafa, Guo Yike. The design and implementation of a workflow analysis tool // PHILOSOPHICAL TRANSACTIONS- ROYAL SOCIETY OF LONDON SERIES A MATHEMATICAL PHYSICAL AND ENGINEERING SCIENCES. — 2010. — 9. — Vol. 368, no. 1926. — P. 4193–4208.

[4] Gansner Emden R., Koutsofios Eleftherios, North Stephen. Drawing graphs withdot. — URL: https://www.graphviz.org/pdf/dotguide.pdf.

[5] InaKoch. Enumerating all connected maximal common subgraphs in two graphs. — elsevier, 2000. — URL: https://www.sciencedirect.com/science/article/pii/S0304397500002863 (online; accessed: 10.12.2017).

[6] J.Ostergard Patric R. A fast algorithm for the maximum clique problem. — elsevier, 2001. — URL: http://www.dcs.gla.ac.uk/~pat/jchoco/clique/indSetMachrahanish/papers/DAM120.pdf.

[7] Jakob Wenzel. pybind11 Documentation. — URL: https://www.graphviz.org/pdf/dotguide.pdf.

[8] Manic G., Bahiense Laura, de Souza Cid. A branch&cut algorithm for the maximum common edge subgraph problem // Electronic Notes in Discrete Mathematics. — 2009. — Vol. 35. — P. 47 – 52. — LAGOS'09 – V Latin-American Algorithms, Graphs

and Optimization Symposium. URL: http://www.sciencedirect.com/science/article/pii/S1571065309001620 (online; accessed: 06.05.2018).

[9] Marco Dorigo Gianni Di Caro Luca M. Gambardella. Ant Algorithms for Discrete Optimization. — URL: http://people.idsia.ch/~luca/ij_23-alife99.pdf (online; accessed: 10.12.2017).

[10] Potvin Jean-Yves. Genetic algorithms for the traveling salesman problem. — Centre de Recherche sur les Transports, Universitd de Montrgal. — URL: https://iccl.inf.tu-dresden.de/w/images/b/b7/GA_for_TSP.pdf (online; accessed: 10.12.2017).

[11] Sammoud Olfa, Solnon Christine, Ghédira Khaled. Ant Algorithm for the Graph Matching Problem // Proceedings of the 5th European Conference on Evolutionary Computation in Combinatorial Optimization. — EvoCOP'05. — Berlin, Heidelberg : Springer-Verlag, 2005. — P. 213–223. — URL: http://dx.doi.org/10.1007/978-3-540-31996-2_20 (online; accessed: 20.05.2018).

[12] A Simulated Annealing Algorithm for Maximum Common Edge Subgraph Detection in Biological Networks / Simon J. Larsen, Frederik G. Alkærsig, Henrik J. Ditzel et al. // Proceedings of the Genetic and Evolutionary Computation Conference 2016. — GECCO '16. — New York, NY, USA : ACM, 2016. — P. 341–348. — URL: http://doi.acm.org/10.1145/2908812.2908858 (online; accessed: 20.05.2018).

[13] The maximum common edge subgraph problem: A polyhedral investigation / Laura Bahiense, Gordana Manić, Breno Piva, Cid C. de Souza // Discrete Applied Mathematics. — 2012. — Vol. 160, no. 18. — P. 2523 – 2541. — V Latin American Algorithms, Graphs, and Optimization Symposium — Gramado, Brazil, 2009. URL: http://www.sciencedirect.com/science/article/pii/S0166218X12000340 (online; accessed: 20.05.2018).

[14] А.С. Лопатин. Метод отжига. — Санкт-Петербургский Государственный Университет. — URL: http://www.math.spbu.ru/user/gran/sb1/lopatin.pdf (online; accessed: 10.12.2017).

[15] В.А.Серебряков М.П.Галочкин. Основы конструирования компиляторов. — М. : URSS, 1999.

[16] Щекалёв Александр. Нирвана — вычислительная платформа для произвольных процессов. — URL: https://events.yandex.ru/lib/talks/4194/.