Алгоритмы для проверки графов на изоморфизм

Фёдор Букреев

23 декабря 2019 г.

Содержание

1 Постановка задачи

Перед формулировкой задачи необходимо ввести базовые определения:

1.1 Основные определения

Определение 1. *Простым неориентированным графом G нозовём пару множеств V, E, где V - это множество вершин графа, а E - множество неупорядоченных пар вершин, называемых рёбрами. Будем считать, что две вершины $a, b \in V$ соединены(смежны) тогда и только тогда, когда $\{a, b\} \in E$.*

Определение 2. *Граф G назовём связным, если для любых вершин $a, b \in V$ существует путь по рёбрам графа из a в b.*

1.2 Задача

Необходимо проверить, изоморфны ли два данных графа G_1, G_2

Определение 3. *Графы $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ изоморфны тогда и только тогда, когда существует отображение $f : V_1 \to V_2$ такое, что $a, b \in V_1$ смежны тогда и только тогда, когда вершины $f(a), f(b) \in V_2$ - смежны*

2 Частные случаи

2.1 Безымянный класс

Приведённый ниже алгоритм проверяет изоморфизм двух графов, принадлежащих некоему классу K. По сути, сам алгоритм и задаёт этот класс.

Предисловие к алгоритму

На первый взгляд может показаться, что алгоритм этот бесполезен, но на самом деле, класс K состоит из почти всех графов. В [14] показано, что вероятность того, что процедура 1-6 успешно завершится на случайном графе не меньше, чем $1 - (\frac{1}{n})^{\frac{1}{7}}$ начиная с некоторого n, а значит она стремится к единице при стремлении n к бесконечности.

Алгоритм

0. Для обоих алгоритмов повторить пункты 1-6.
1. Для каждой вершины v посчитать $d(v)$ - её степень.
1. Упорядочить вершины по значенийм $d(v)$, то есть теперь $d(1) \geqslant d(2) \geqslant ... \geqslant d(n)$
2. Обозначим $k = [3log_2 n]$. Если существует вершина $v_i : i < k$ и $d(i) = d(i+1)$, то завершаемся неудачей.
3. Для каждого $i \in \{k+1, ..., n\}$ считаем бинарный вектор $f(i) = (e_1, ..., e_k)$, где $e_j = 1 \leftrightarrow (i, j) \in E$
4. Переупорядочим вершины v_i с номерами большими, чем k по $f(i)$. То есть $f(k+1) \geqslant ... \geqslant f(n)$
5. Если существует $i : f(i) = f(i+1)$, завершаемся неудачей.
6. Мы получили каноническую нумерацию.
7. Графы изоморфны $\Leftrightarrow$ их канонические нумерации совпадают.

Послесловие к алгоритму

Понятно, что если алгоритм завершился, то завершился он за квадратичное от количества вершин время, то есть за линейное от размера графа в модели с произвольным доступом, если представлять его матрицей смежности. В других представлениях, время может стать квадратичным от входа.

2.2 Планарные графы

Задача проверки двух планарных графов на изоморфизм не просто лежит в P, но решается за линейное время(в модели с произвольным доступом) [1] и лежит в классе Ł[2]. Отмечу, что дерево является частным случаем планарного графа.

Определение 4. *Планарным называется граф, который можно изобразить на плоскости так, чтобы никакие два его ребра не пересекались.*

Определение 5. *Плоским назовём граф, изображенный на плоскости*

Определение 6. *Граф k-связен, тогда и только когда он имеет больше чем k вершин и после удаления менее чем k любых вершин граф остаётся связным.*

Я приведу первый алгоритм в этой области, предназначенный для несколько меньшего класса графов - для планарных 3-связных графов[3], созданный Lois Weinberg. Этот алгоритм был улучшен и обобщён (Hopcroft и Tarjan [6]) до алгоритма, работающего за $O(n \cdot logn)$ для **произвольного** планарного графа. Затем данный алгоритм был ускорен до линейного времени (Hopcroft and Wong [1]). Что же особенного в планарных 3-связных графах? Дело в том, что в то время, как в общем случае порядок группы автоморфизмов графа может быть равен $|V|!$, доказано[4], что размер группы автоморфизмов для планарных 3-связных графов равен $4|E|$. Это позволяет существенно сократить перебор вариантов.

Предисловие к алгоритму

Алгоритм, описанный ниже опирается на несколько теоретических утверждений о простых планарных 3-связных графах, которые следует упомянуть до его описания:

1) Согласно [5], такой граф имеет уникальное представление на плоскости. Стоит отметить, что именно подразумевается под уникальностью. Имеется ввиду, что его грани, ограниченные рёбрами уникальны в смысле набора рёбер, которые их задают. То есть "зеркальный"граф, полученный обращением циклов граней, тоже может являться представлением исходного графа на плоскости.

2) Как было отмечено выше, в работе [4] показано, что порядок группы автоморфизмов такого графа не превосходит $4|E|$

Алгоритм

На вход алгоритм получает простой *плоский* ненаправленный 3-связный граф $G = (V, E)$. В случае компьютерной программы удобно считать, что полученный граф - это не изображение на плоскости, а структура данных, где каждой вершине сопоставлен набор рёбер в порядке обхода

против часовой стрелки таким образом, что изображение данного графа не имеет пересекающихся рёбер.

Каждому автоморфизму графа мы сопоставим некоторый однозначно задающий его код, сгенерированный во время обхода графа по эйлеровому циклу. Для того, чтобы гарантировать наличие эйлерового цикла мы продублируем каждое ребро $v \to v_1, v_2$ в графе, и выдадим полученым парам ребер противоположные направления. То есть из ребра $\{a, b\}$ мы получим направленные рёбра $(a, b), (b, a)$, $a, b \in V$. В полученном ориентированном графе G' входящая степень любой вершины равна её исходящей степени, а значит эйлеров цикл существует. Обойдём данный граф по следующему правилу:

0. Все вершины графа разделим на 2 типа: *старые* и *новые*, изначально все вершины графа - новые. Выберем начальное ребро и направление на нём в графе или его зеркальной копии. Это $2 \cdot 2|E|$ вариантов. Первой вершиной обхода станет первая вершина ребра, согласно выбранному направлению, второй - вторая.

1. Попадая в *новую* вершину, отмечаем её как старую и двигаемся по ребру, имеющему минимальный от нашего угол в направлении против часовой стрелки. В случае компьютерной программы, это будет следующее после того, по которому мы пришли к вершине, ребро в наборе.

2. Попадая в *старую* вершину по ребру, противоположное направление которого ещё не было посещено, возвращаемся по противоположному ребру.

3. Попадая в *старую* вершину по ребру, обратное которому уже было пройдено, двигаемся по ребру, имеющему минимальный от нашего угол в направлении против часовой стрелки и при этом непосещённому ранее.

Во время данного обхода каждый раз встречая *новую* вершину, присваиваем ей номер. То есть начальной вершине присвоен номер 1, второй - номер 2 и так далее. Попадая в очередную вершину, после присвоения номера(если оно было необходимо), добавляем в код текущего обхода номер посещённой вершины. Таким образом, для каждого обхода получим код длины $2|E| + 1$ который удобно представить ввиде вектора.

После повторения процедуры для всех $4|E|$ обходов, получим таблицу кодов размера $4|E| \times 2|E| + 1$. Затем сортируем все полученные коды в лексикографическом порядке. Сгенерировав(и отсортировав) такую матрицу для обоих графов, вопрос об изоморфизме сводится к проверке равенства первых столбцов в этих матрицах, то есть графы изоморфны тогда и только тогда, когда их лексикографически минимальные коды совпадают.

Послесловие к алгоритму

Стоит отметить, что если графы оказались изоморфны, то обратив процедуру обхода, описанную в алгоритме можно получить нумерацию вершин в каждом графе, сотвествующую изоморфизму.

Алгоритм так же опирается на следующие теоретических утверждения:

1) Простые 3-связные планарные графы G_1, G_2 изоморфны тогда и только тогда, когда множества их кодов совпадают

2) Простые 3-связные планарные графы G_1, G_2 изоморфны тогда и только тогда, когда выполнено одно из двух условий:

a) Первые строки матриц их кодов(минимальные в лексикографическом смысле коды) совпадают

b) Любые 2 строки их матриц совпадают

Отмечу, что второе утверждение позволяет сократить количество вычислений в большом количестве случаев.

Кроме того, на основе данного алгоритма можно проверять на изоморфизм и ориентированные графы. Необходимые для этого модификации описанны в оригинальной статье [3].

2.3 Теория Бабаи-Лакса

Бабаи и Лакс внесли большой вклад в изучение проблемы изоморфизма графов. Их подход активно использует теорию групп. В этом разделе я приведу некоторые результаты их теории, а в следующих - два алгоритма, которые их используют

Пусть G - группа перестановок, действующая на $\{1, ..., n\}$, а G_m - это подгруппа G, оставляющая $\{1, ..., m\}$ на месте. Мы получили *башню групп* $G = G_0 \geqslant G_1 \geqslant ... \geqslant G_n = id$.

Напомню, что системой представителей группы G по группе H называют такое подмножество X из G, что для любого $g \in G$ найдётся $x \in X$ такое, что $gH = xH$. Пусть K_m - система представителей левых смежных классов G_m по G_{m+1}. Каждый $g \in G_m$ можно однозначно представить как $g =$

$k_m k_{m+1} ... k_{n-1}$, где $k_i \in K_i$. Такое представление элемента назовём *каноническим*. Систему образующих $K = \cup K_i$ назовём *сильной*. Из соображения $|G_m : G_{m+1}| \leqslant n - m$ видно, что $|K| \leqslant n^2$.

Фурст, Хопкрофт и Лакс [нужна ссылка] предложили алгоритм, строящий сильную систему образующих по произвольной системе образующих за полиномиальное время. Данный алгоритм позволяет быстро решать некоторые важные задачи, например:

1. Разибть множество {1, ..., n} на орбиты группы G
2. Найти порядок группы G
3. Проверить, лежит ли $\delta \in S$ в G

Внутри этот алгоритм использует процедуру, на которой основано доказательство результата, который будет использован далее в алгоритме. Но сначала определение:

Определение 7. *Назовём подгруппу $H \leqslant G$ полиномиально распознаваемой, если существует алгоритм полиномиальной сложности, который для любого элемента G проверяет лежит ли этот элемент в H.*

Теорема 1. *Пусть $p(x), q(x)$ - полиномы, $A = A_0 \geqslant A_1 \geqslant ... \geqslant A_m$ - башня подгрупп группы A, удовлетворяющая следующим условиям:*

1. $[A_{i-1} : A_i] \leqslant p(n)$

2. $m \leqslant q(n)$

3. A_i полиномиально распознаваема в A_{-1}

Тогда существует полиномиальный аллгоритм, который по системе образующих любой из A_i строит систему представителей смежных классов G_{i-1} по G_i и сильную систему образующих G_i для любого i

2.4 Графы с ограниченной кратностью цветов

В этом алгоритме мы рассматриваем только связные графы. Если графы несвязны, достаточно попарно проверить на изоморфизм из компоненты.

Определение 8. *Пару (G, f) назовём раскрашенным графом, если $G = (V, E)$ - это граф, а $f : V \to \mathbb{N}$ - отображение, определённое на всём X.*

Определение 9. *Цветом вершины v назовём $f(v)$.*

Определение 10. *Цветным классом $i \in \mathbb{N}$ назовём $f^{-1}(i)$*

Определение 11. *Кратность цвета i - это размер его цветного класса.*

Задача проверки изоморфизма раскрашенного графа отличается от определённой ранее тем, что дополнительно к условию сохранения рёбер добавляется условие сохранения цвета. То есть, если $(G = (V, E), f), (G' = (V', E'), f')$ - 2 раскршенных графа, то отображение $\phi : V \leftrightarrow V'$ должно удовлетворять двум условиям:

1) $(x, y) \in E \Leftrightarrow (\phi(x), \phi(y)) \in E'$

2) $f(x) = k \Leftrightarrow f'(\phi(x)) = k$

Приведённый ниже алгоритм решает задачу проверки двух раскрашенных графов на изоморфизм за полином, степень которого линейно зависит от максимальной кратности его цветов.

Предисловие к алгоритму

Данный алгоритм сводит задачу проверки двух раскрашенных графов на изоморфизм к задаче нахождения системы образующих определённой группы, которая затем решается с помощью теории приведённой в предыдущем пункте.

Пусть нам даны раскрашенные графы $(G_1, f_1), (G_2, f_2)$, кратности цветов которых не превосходят константы C. Рассмотрим раскрашенный граф $G' = (V_1 \sqcup V_2, E_1 \sqcup E_2)$ с соответствующей раскраской. Степень кратности цветов G' не превышает $2C$. Теперь рассмотрим автоморфизмы G' в себя. Понятно, что исходные графы изоморфны тогда и только тогда, когда существует изоморфизм, переводящий V_1 в V_2. Чтобы проверить существование такого зоморфизма достаточно найти образующие группы автоморфизом G'. Алгоритм за авторством Бабаи приведённый ниже находит нужную систему образующих за полиномиальное время.

Алгоритм

Дан граф G, кратности цветных классов которого не превосходят c, а $V = C_1 \sqcup C_2 \sqcup ... \sqcup C_k$ - разибение множества его вершин на цветные классы. Обозначим подграф G, порождённый объединением $C_i \sqcup C_j$ как $G_{i,j}$ (считаем, что $i < j$). Пронумеруем все такие графы и обозначим как $H_i, i \in \{1, ..., \binom{k}{2}\}$. Построим теперь последовательность графов G_i:

$G_0 = (V, \emptyset)$

$G_i = G_{i-1} \cup H_i$

Пусть теперь $A_i = Aut(G_i)$. Понятно, что $A_i \geqslant A_{i+1}$, а значит мы получили башню групп, где $A_{\binom{k}{2}} = Aut(G)$ и $A_0 = Sym(C_1) \times ... \times Sym(C_k)$. A_i полиномиально распознаваема в A_{i-1}, $[A_{i-1} : A_i] \leqslant (c!)^2 = const$. Система порождающих A_0 очевидна. Тогда согласно теореме 1 в пункте 2.2 мы можем построить систему образующих $Aut(G)$ за полиномиальное время. А это значит, что мы сможем проверить наличие нужного автоморфизма. Заметим, что $|Aut(G)| = [A_0 : A_1][A_1 : A_2]...[A_{\binom{k}{2}-1} : A_{\binom{k}{2}}] \leqslant (c!)^2(\binom{k}{2} - 1)$, то есть перебор тоже будет произведён за полиномиальное время.

Послесловие к алгоритму

Задача проверки "обычного" изоморфизма графов сводится к данной раскрашиванием всех вершин в один цвет, но это не имеет смысла, так как в таком случае максимальная кратность цветов графа равна количеству вершин и верхняя оценка времени работы получается хуже, чем в случае тривиального алгоритма. Но данный алгоритм используется для проверки на изоморфизм некоторых других классов графов.

3 Общий алгоритм

Понятно, что тривиальный алгоритм (пробовать все возможные нумерации одного из графов, пока они не закончатся, либо графы не совпадут) работает за $O(n!)$, так как $|Aut(V)| = n!$. Современные результаты таковы: есть алгоритм Бабаи и Лакса, работающий за время $2^{O(\sqrt{nlog(n)})}$ [12]. Данный алгоритм опирается на теорему о классификации простых конечных групп. Без этой теоремы он тоже работает, но оценка его времени работы увеличивается. В 2017 году Бабаи представил последнюю версию своего квазиполиномиального алгоритма, но этот алгоритм ещё не был проверен [13]. Последний алгоритм так же опирается на теорему о классификации простых конечных групп. Он описан в статье на 89 страниц. У меня нет возможности описать ни один из этих сложных алгоритмов, поэтому я опишу другой, чуть менее сложный, но всё же субфакториальный.

Предисловие к алгоритму

Мы вновь рассмотрим несколько модифицированную проблему изоморфизма графов:

Пусть $G_1 = (V, E), G_2 = (VV, EE)$ - два графа. Пусть также $\tau = (V_1, ..., V_l), \sigma = (VV_1, ..., VV_l)$ -отношения эквивалентности на на V_1, V_2 соответственно. Тогда от изоморфизма ϕ дополнительно требуется, чтобы для любого $i \in \{i, ..., l\}$ выполнялось $\phi(V_i) = VV_i$

Понятно, что стандартная проблема изоморфизма - это частный случай нашей: можно взять отношения эквивалентности, где все равны.

Определение 12. *Пару $A, B \in V$ назовём безразличной, если либо $\forall a \in A, b \in B \quad (a, b) \notin E$, либо $\forall a \in A, b \in B \quad (a, b) \in E$*

Определение 13. *Обозначим за $E(x)$ все вершины, соединённые с x*

Определение 14. *Обозначим за $v_\tau(x) = (v_1(x), ..., v_n(x))$, где $x \in V$, $\tau = (V_1, ..., V_l)$ - это отношение эквивалентности на V, а $v_i(x) = |E(x) \cap V_i|$*

Определение 15. *Отношение эквивалентности τ назовём стабильным, если $x \tau y \Rightarrow v_\tau(x) = v_\tau(y)$, где $x \tau y = 1$, если x эквивалентен y относительно τ и 0 иначе.*

Определение 16. *Стабильную эквивалентность τ^* назовём замыканием τ, если $\tau^* \leqslant \tau^*$ и для любого стабильного σ такого, что $\sigma \leqslant \tau$ верно, что $\sigma \leqslant \tau^*$*

Определение 17. *Подмножество вершин S назовём секцией G относительно $\tau = (V_1, ..., V_l)$, если для любых $i, j \in \{1, ..., l\}$ верно, что пара $V_i \cap S, V_j \,|S$ - безразлична. Вектор $q(S) = (S \cap V_1, ..., S\ capV_l)$ назовём вектором секции. Обозначим множество неотсортированных пар (x, y), где $x \in V_i \cap S, y \in V_j \,|S$ через $S_{i,j}$. Тогда матрицу $M = (m_{i,j})$,*

$$m_{i,j} = \begin{cases} 1, \text{если } S_{i,j} \cup S_{j,i} \neq \emptyset \text{ И } S_{i,j} \cup S_{j,i} \subset E \\ 0, \text{если } S_{i,j} \cup S_{j,i} \neq \emptyset \text{ И } S_{i,j} \cup S_{j,i} \cap E = \emptyset \\ undefined, \text{если} S_{i,j} \cup S_{j,i} = \emptyset \end{cases}$$

назовём матрицей секции.

Например, если I - это все классы τ из одной вершины, то если $B \subset I$, $C \supset V \backslash I$, то B, C - секции относительно τ. Такие секции будем называть тривиальными. Если нетривиальная секция S содержит нетривиальную секцию $T \neq S$ такую, что $M(T) = M(S)$, назовём её минимальной.

Теорема 2. *Пусть τ, σ - стабильные эквивалентности. Графов G, H соответственно. Тогда если S - секция относительно τ, то*

1. Если ϕ - изоморфизм $(G, \tau) \to (H, \sigma)$, то $\phi(S)$ - секция относительно σ

2. Если T - секция H относительно σ, такая, что $M(S) = M(T)$, то любые изоморфизмы $\phi_1 = (S, \tau(S)) \to (T, \sigma(T), \phi_2 = (G\,|S, \tau(G\,|S)) \to (H\,|T, \sigma(H\,|T))$ образуют изоморфизм $\phi = \phi_1 \cup \phi_2$

3. Если $M(S) = M(T)$ и пары $(G, \tau), (H, \sigma)$ и $(S, \tau(S)), (T, \sigma(T))$ - изоморфны, то пара $(G\,|S, \tau(G\,|S)), (H\,|T, \sigma(H\,|$ тоже изоморфна.

Алгоритм

Считаем, что все эквивалентности стабильны. В противном случае можно использовать замыкание эквивалентности, которое можно найти за полиномиальное время.

Назовём класс эквивалентности эффективым, если его размер больше единицы.

Пусть изначальная задача - проверить на изоморфизм пары $(G, \tau), (H, \sigma)$.

Найдём все минимальные секции(это можно сделать за полином, но для наших целей подойдёт и проверка в лоб за 2^n) $(S_1, ..., S_k)$ в G и $(T_1, ..., T_k)$ в H и разделим на подмножества согласно их секционным векторам. Пусть $A_i = \{S_{i,1}, ..., S_{i,p_i}\}$ $B_i = \{T_{i,1}, ..., T_{i,p_i}\}$, $v(S_{i,j} = T_{i,j}) = v_i$, где $i \in \{1, ..., q\}$

Понятно, что $S = \cup_{i=1}^{q} \cup_{j=1}^{p_i} S_{i,j}$ и $S_{i,j} \cap S_{i',j'} = \emptyset$, если $(i, j) \neq (i', j')$.

Теперь согласно Теореме 2, мы можем рабить исходную задачу на ряд более мелких подзадач $K_1, ..., K_q$. Рассмотрим, например K_1: для $S_{i,1}$ решаем задачи проверки на изоморфизм таких пар:

$P_{i,1} = ((S_{i,1}, \tau(S_{i,1})), (T_{i,1}, \sigma(i, 1))),$ ..., $P_{i,t} = ((S_{i,t}, \tau(S_{i,t})), (T_{i,t}, \sigma(i, t))),$

где t - первая задача, ответ на которую "да". Выкидываем $S_{i,1}.T_{i,1}$ из A_i, B_i соответственно и повторям процесс для $S_{i,2}$. Ответ на изначальную задачу - "да" тогда и только тогда, когда для каждого i найдётся t_i такой, что ответ на задачу P_{i,t_i} - "да". Каждую задачу из $P_{i,j}$ разбиваем аналогичным образом. В конце концов мы приходим к набору проблем, для которых невозможно найти минимальные секции. Назовём этот набор $Ex(P)$. Замечу, что возможна ситуация, когда $P = Ex(P)$, и описанная только что часть алгоритма пропускается.

Теперь строим изоморфизм ввиде дерева: корень будет фиктивным и назовём мы его Z_0. Затем каждой проблеме Z в $Ex(P)$(и одновременно узлу в нашем дереве) добавит в качестве детей $Ex(Re(Z))$, где $Re(P) = \{P_i = [(G, \tau^*_{x_i}), (H, \sigma^*_{y_i})]\}$. И через некоторое количество повторений все эквивалентности превратятся в тривиальные, где никто никому не равен, что эквивалентно заранее заданной нумерации и проверяется за линию.

Послесловие к алгоритму

Этот алгоритм представлен в [14] и там же показано, что его время работы - $O(c^n)$, где c - некоторая константа.

Список литературы

[1] Hopcroft, John; Wong, J. (1974), "Linear time algorithm for isomorphism of planar graphs Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, pp. 172–184, doi:10.1145/800119.803896.

[2] Datta, S.; Limaye, N.; Nimbhorkar, P.; Thierauf, T.; Wagner, F. (2009), "Planar graph isomorphism is in log-space 2009 24th Annual IEEE Conference on Computational Complexity, p. 203, arXiv:0809.2319, doi:10.1109/CCC.2009.16, ISBN 978-0-7695-3717-7.

[3] Lois Weinberg, "A Simple and Efficient Algorithm for Determining Isomorphism of Planar Triply Connected Graphs"

https://ieeexplore.ieee.org/document/1082573/

[4] Lois Weinberg, "On the Maximum Order of the Automorphism Group of a Planar Triply Connected Graph"

https://epubs.siam.org/doi/abs/10.1137/0114062

[5] H. Whitney, "2-isomorphic graphs Amer. J. Math., 55 (1933) 245–254,

https://www.jstor.org/stable/2371127?seq=1

[6] E. Hopcroft and R.E. Tarjan. "Efficient planarity testing". Journal of the ACM, 21, 1974.

[7] Rudolf MATHON (March 1979), "A note on the graph isomorphism counting problem"

[8] Babai, László; Grigoryev, D. Yu.; Mount, David M. (1982), "Isomorphism of graphs with bounded eigenvalue multiplicity Proceedings of the 14th Annual ACM Symposium on Theory of Computing, pp. 310–324, doi:10.1145/800070.802206, ISBN 0-89791-070-2.

[9] Merrick Furst ; John Hopcroft ; Eugene Luks, "Polynomial-time algorithms for permutation groups Published in 21st Annual Symposium on Foundations of Computer Science (sfcs 1980)

[10] Leighton F. T.; Miller G. L., "Numerical analysis of Gaussian elimination and eigenspace calculation in preparation

[11] Семинары Ленинградского отделения института Стеклова TOFIX

[12] Babai, László; Luks, Eugene M. (1983), "Canonical labeling of graphs Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing (STOC '83), pp. 171–183, doi:10.1145/800061.808746, ISBN 0-89791-099-0

[13] Babai, László, "Graph Isomorphism in Quasipolynomial Time"arXiv:1512.03547

[14] M. K. Goldberg, "A nonfactorial algorithm for testing isomorphism of two graphs,"Com- binatorics and Optimization, Research Reports, No. 80-36, Faculty Math. Univ. Waterloo (1980).