ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

УДК 519.178

Отчет об исследовательском проекте на тему:

Алгоритмы поиска замыкаемых путей в графах

Выполнил:
студент группы БПМИ205
Стёпкин Степан Максимович

(подпись) (дата) 18.05.2023

Принял руководитель проекта:
Вялый Михаил Николаевич
Профессор
Факультета компьютерных наук НИУ ВШЭ

(подпись) 18.05.2023 (дата)

Москва 2023

Содержание

Алгоритмы поиска замыкаемых путей в графах

Аннотация

Индуцированный путь в графе называется замыкаемым, если любое его двустороннее индуцированное расширение содержится в индуцированном цикле. Замыкаемый путь на k вершинах существует в любом графе, в котором есть такой путь [1]. Остается открытой задача поиска замыкаемого пути. Одним из естественных способов построения такого пути является переход от некоторого заранее заданного пути к замыкаемому с помощью сдвигов (с одного конца ребро добавляем, с другого удаляем). Такой переход всегда возможен [2], но нет хороших оценок на требуемое количество сдвигов. Данный проект ставит перед собой цель исследовать задачу нахождения замыкаемого пути: либо построить достаточно эффективные алгоритмы, либо доказать трудность этой задачи в одном из классов сложности.

Ключевые слова

Теория графов, алгоритмы на графах, алгоритмическая сложность, Индуцированный путь, замыкаемый путь, сдвиг, расширение, стягивание

Введение

Основные определения и обозначения

В отчете я следую терминологии используемой в [3]. Все графы, которые упомянуты в этом отчете конечные, неориентированные, могут иметь петли и кратные ребра. Я считаю, что визуализация просто необходима, чтобы действительно понять что происходит, а потому буду стараться визуализировать все нетривиальные операции.

$V(G)$ - множество вершин графа G. $E(G)$ - множество рёбер графа G.

$N(v)$ - множество, состоящее из соседей вершины v. $N[v]$ - множество, состоящее из вершины v и ее соседей.

$G - H$ - граф G из которого удалили все вершины и соответствующие им ребра подграфа H.

Определение 0 (Стягивание вершин). Пусть u и v - вершины связанные ребром в графе G. Тогда *стягиванием вершин* u и v в G называют замену вершин u и v на вершину u', которая соединена ребром со всеми вершинами из множества $N(u) + N(v)$. Обозначение: $G/_{uv \to u'}$.

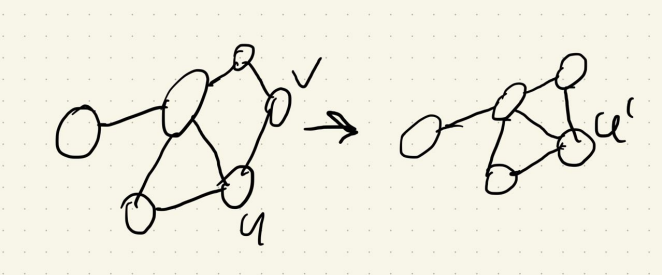

Рис. 0.1: Стягивание вершин u и v в u'

Используемая терминология

Определение 1 (Индуцированный подграф). Подграф H графа G называется *индуцированным подграфом* G, если множество рёбер в H это в точности множество рёбер из G, которые имеют обе вершины в $V(H)$. То есть путь индуцирован, если $\forall |i-j| \neq 1 : p_i p_j \notin E(G)$.

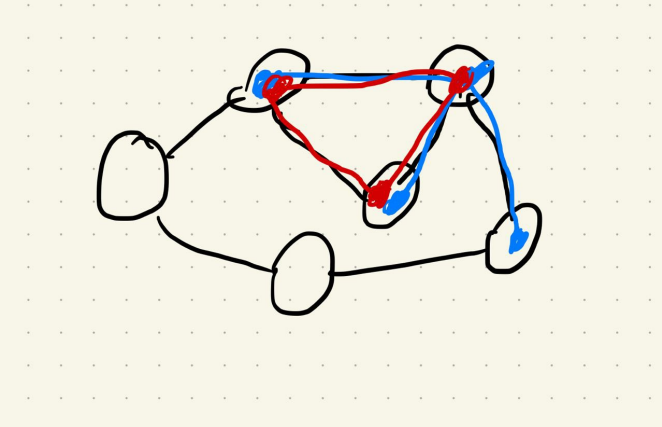

Рис. 0.2: Красным обозначен индуцированный подграф. Синий подграф не являтся индуцированным

Заметим, что путь в графе G может быть рассмотрен как подграф G. Таким образом, мы говорим что путь в G индуцирован, если соответствующий ему подграф индуцирован в G.

Определение 2 (Двустороннее индуцированное расширение). Пусть W и W' - индуцированные пути в графе G. Пусть $W = (v_1, e_1, v_2, \ldots, e_{k-1}, v_k)$, для каких-то $v_1, \ldots, v_k \in V(G)$ и $e_1, \ldots, e_{k-1} \in E(G)$. Тогда W' называется *двусторонним индуцированным расширением* W, если $W' = (v_0, e_0, v_1, e_1, v_2, \ldots, e_{k-1}, v_k, e_k, v_{k+1})$ для каких-то $v_0, v_{k+1} \in V(G)$ и $e_0, e_k \in E(G)$.

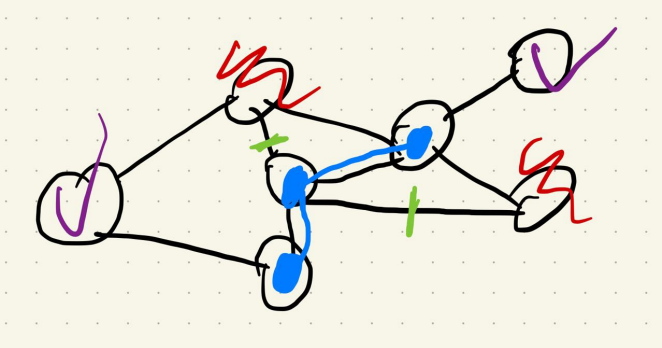

Рис. 0.3: p_1, p_2, p_3 - обозначен синим. Фиолетовым обозначено двустороннее индуцированное расширение. Красные вершины не являются индуцированным расширением. В данном случае W - синие вершины, W' - синие + фиолетовые.

Определение 3 (Симплицированный индуцированный путь). Индуцированный путь W в графе G называется *симплицированным*, если у него не существует двусторонних индуцированных расширений в G.

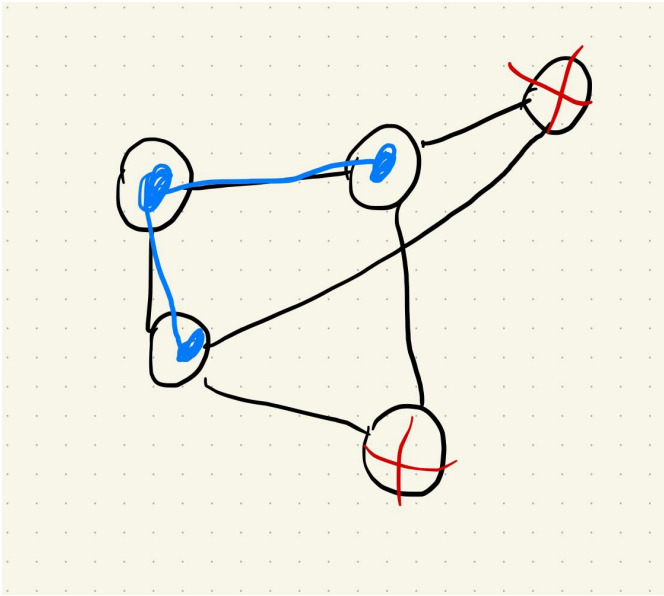

Рис. 0.4: Синим обозначен индуцированный путь. У него нет расширений, так как красные вершины связаны с более чем одной синей вершиной

Определение 4. (Закрытый индуцированный путь). Индуцированный путь W в графе G называется *закрытым*, если он подпуть индуцированного цикла в G.

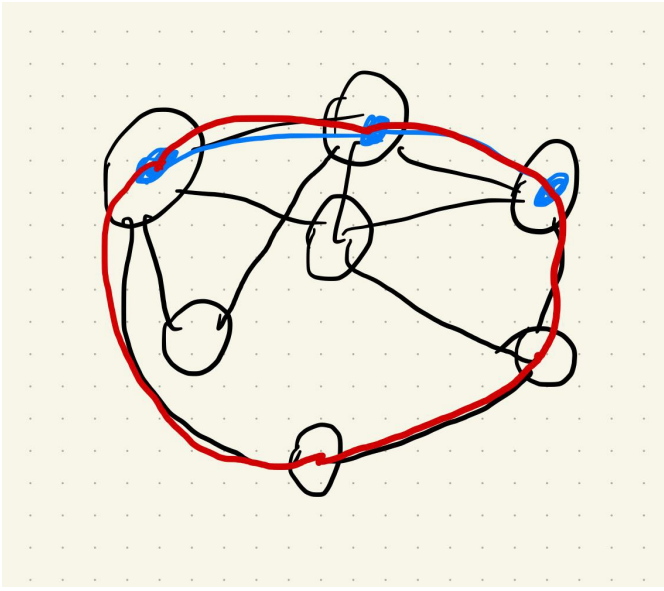

Рис. 0.5: Синим обозначен индуцированный путь. Красным обозначен индуцированный цикл

Определение 5. (Замыкаемый индуцированный путь). Индуцированный путь W в графе G называется *замыкаемым*, если любое его двустороннее индуцированное расширение в G является закрытым.

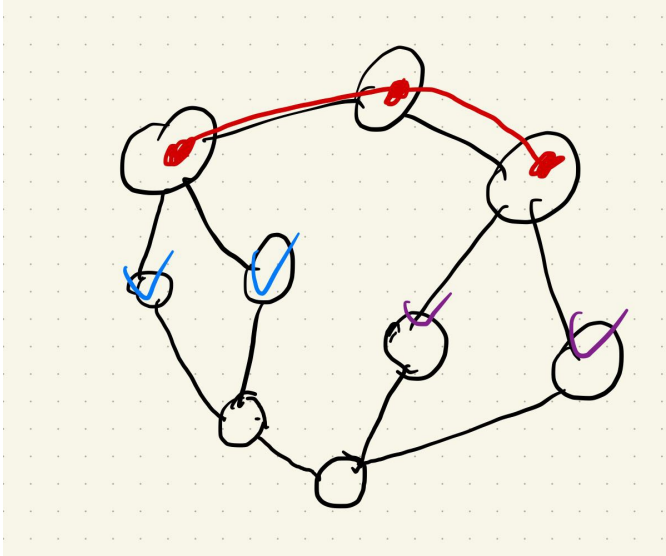

Рис. 0.6: Любое двустороннее расширение красного пути (с одной стороны синее, с другой фиолетовое) является закрытым

Из этого определения также следует, что любой упрощенный (симплицированный) индуцированный путь является замыкаемым, так как у него нет расширений.

Определение 6. (Сдвиг индуцированного пути). Пусть W - индуцированный путь в графе G с хотя бы одним ребром. Пусть $W = (v_0, e_0, v_1, \ldots, e_{k-1}, v_k)$, для каких-то $v_0, \ldots, v_k \in V(G)$ и $e_0, \ldots, e_{k-1} \in E(G)$. Тогда мы говорим, что $W' = (v_1, \ldots, v_{k-1}, e_{k-1}, v_k)$ и $W'' = (v_0, e_0, v_1, \ldots, v_{k-1})$ *индуцированные сдвиги* друг друга в G.

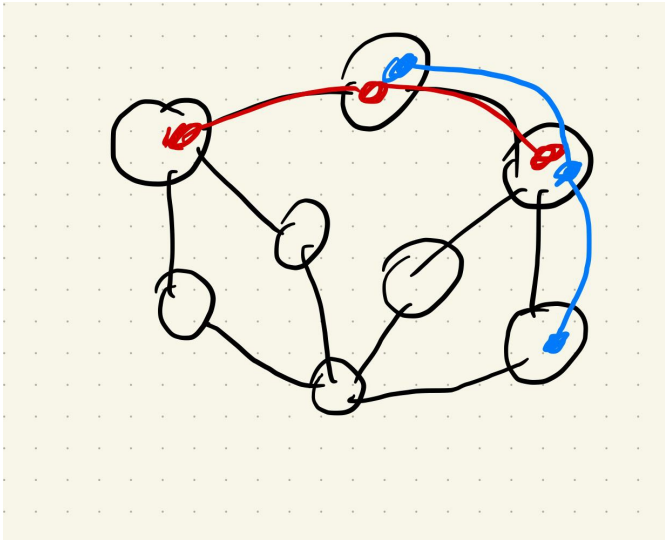

Рис. 0.7: Красный и синий пути являются индуцированными сдвигами друг друга

Простыми словами - если представить путь как "змейку"в графе, то сдвиг это один её "шаг с сохранением свойств пути. Для двух индуцированных путей W и W' в графе G, мы говорим что W может быть сдвинут в W', если существует последовательность сдвигов $W = W_0, W_1 \dots W_k = W'$.

Постановка задачи

Целью данной работы является установить алгоритмическую сложность задачи поиска замыкаемого пути.

Алгоритм решения задачи существует[2] и предлагается также разработать алгоритм на основе доказательства теоремы о замыкаемых путях[4].

Нужно либо доказать полиномиальную сложность алгоритмов, либо привести контрпример с экспоненциальной сложностью. Кроме того ничего из вышеперечисленного может не получиться. В таком случае нужно доказать полиномиальность для как можно большего числа графов, чтобы в будущем это можно было обобщить для всех графов.

Если же доказать полиномиальность рекурсии получится, то нужно будет реализовать нужные методы и функции с наименьшей сложностью, чтобы получить оптимальную оценку.

Актуальность и значимость

По ссылкам несложно заметить, что научные работы по замыкаемым путям активно проводятся прямо сейчас. Некоторые из статей, на которые опирается данная работа еще не были опубликованы. В данный момент неизвестна сложность задачи поиска замыкаемого пути. Несмотря на то, что аналитически лучшая известная оценка экспоненциальна, на практике сложности более чем линейной не встречалось. Можно найти большой объем NP-полных задач в исследованиях на похожие темы. Данная работа внесет вклад в развитие нового направления исследований теории графов.

Существующие работы и решения

Основные две работы на которые я буду опираться во время проекта - "Shifting paths to avoidable ones"[2] и "Avoidability beyond paths"[4]. В первом уже существует алгоритм нахождения замыкаемых путей, сдвигом индуцированных путей. Доказательство корректности этого алгоритма есть в источниках, оно длинное, потому здесь не приводится. Сам алгоритм состоит из двух процедур:

Procedure 1 SHIFTING(G, P)

Input: a graph G and an induced path $P = p_1 p_2 \dots p_k$ in G
Output: a sequence S of paths shifting P to an avoidable induced path in G

if there exists an extension xPy of P **then**
    $Q \leftarrow y p_k \dots p_1 x$
    return P, REFINEDSHIFTING(G, Q)
else
    return P

Procedure 2 REFINEDSHIFTING(G, P)

Input: a graph G and an induced path $P = p_1 \dots p_{k+2}$ in G
Output: a sequence S of paths in $G - N[p_{k+2}]$ shifting $p_1 \dots p_k$ to an avoidable induced path in G

$P' \leftarrow p_1 \dots p_k$
$S \leftarrow$ the one-element sequence containing path P'
if there exists an extension $xP'y$ in $G - N[p_{k+2}]$ **then**
    $S \leftarrow S$, REFINEDSHIFTING$(G - N[p_{k+2}], xP'y)$
$Q \leftarrow$ the end path of S
if Q has an extension xQy in G such that y is the unique neighbor of p_{k+2} in $\{x, y\}$ **then**
    let $Q = q_1 \dots q_k$ such that y is adjacent to q_k
    $Q' \leftarrow x q_1 \dots q_k$
    $G' \leftarrow G/_{p_{k+2}y \to y'}$
    $S' \leftarrow$ REFINEDSHIFTING$(G', Q'y')$
    return S, S'
else
    return S

Рис. 0.8: Псевдокод алгоритма из статьи "Shifting paths to avoidable ones"

Процедура 1 ищет расширение индуцированного пути длины k, и если его нет, то путь уже замыкаем. Затем она передает **процедуре 2** это расширение. **Процедура 2** получает на вход граф и путь длины $k + 2$. Отдает она замыкаемый путь длины k. Путь длины $k + 2$ используется для удобных сдвигов и запоминания откуда мы пришли. Функция нетривиальна, но подробно расписана пояснениями для ее изучения. В статье не сказано как выбирается расширение, так как статья не ставила перед собой цель оптимизировать сложность процедуры.

Из второй работы, несмотря на 30 страниц текста, нас интересует только одна строчка - теорема 5.1. Из нее формируется абсолютно аналогичное решение, в котором мы "рандомно" избавляемся от некоторых веток рекурсии - ассимптотически тот же алгоритм.

Предлагаемые подходы и методы

Заметим, что экспоненциальная оценка видна сразу. Мы можем рекурсивно войти в **Процедуру 2** два раза. В первом случае, количество вершин в итоговом графе сокращается хотя бы на 2 (p_{k+2} и ее сосед p_{k+1}). Во втором случае две вершины заменяются на одну. Если $F(N)$ - сложность рекурсии, для графа с N вершинами, то $F(N) = F(N-1)+F(N-2)$, то есть последовательность Фиббоначи. Экспоненциальность последовательности Фиббоначи общеизвестна.

Также можно заметить, что на самом деле, если у p_{k+2} всего 1 сосед, то путь $(p_2, p_3, \ldots, p_{k+2})$ упрощен, а значит замыкаем. Но рекурсивная формула $F(N) = F(N-3) + F(N-1)$ всё еще экспоненциальна.

Больше об алгоритме

Будем рассматривать граф рекурсии. Наша задача понять количество вершин в нем. В обоих вызовах рекурсия вызывается на уже меньшем графе, назовем вызовом D первую рекурсию на графе $G - N[p_{k+2}]$, а C - на графе $G/_{uv \to u'}$.

Так например выглядит рекурсия D в графе:

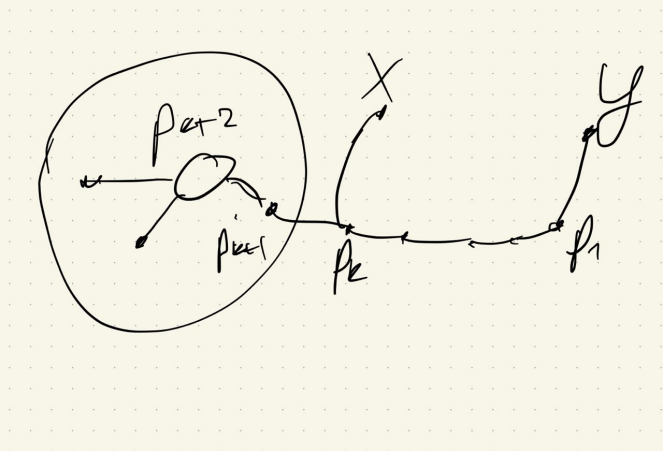

Рис. 0.9: Граф после операции D. $N[p_{k+2}]$ удален, $y, p_1, ...p_k, x$ - новый путь.

Попробуем оценить сколько же может быть D на пути из корня в лист рекурсии. Сразу становится понятно, что D удаляет хотя бы две вершины (p_{k+2} и p_{k+1}). То есть может быть не более $\frac{n-k}{2}$ D. К сожалению это лучшая оценка, несмотря на то что кажется, что расстояние между удаленными вершинами как минимум 3, на самом деле оно может быть равно 2:

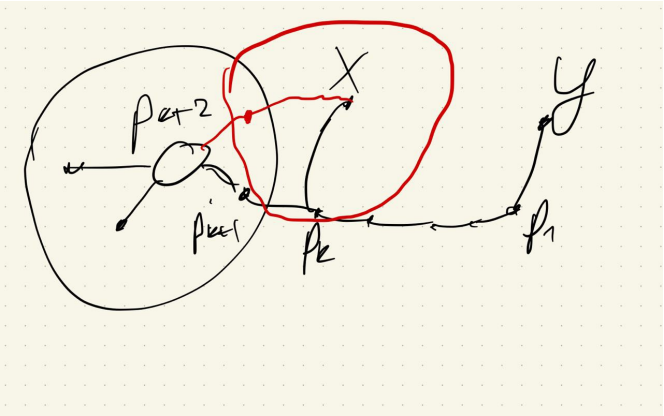

Рис. 0.10: Вершина X могла быть связана ребром с какой-то удаленной вершиной из $N[p_{k+2}]$

Идея - попробуем оценить максимальное количество рекурсий D подряд через дополнение. Ведь если произошел сдвиг индуцированного пути, значит мы "запрещаем"какие-то ребра, другими словами этих ребер не было в графе, то есть они есть в дополнении. Так как нам дан индуцированный путь, то это уже означает, что в графе дополнении есть как минимум $\frac{k(k-1)}{2} - (k-1) = \frac{(k-2)(k-1)}{2}$ рёбер. Теперь, заметим, что на каждом шаге D, мы добавляем хотя бы $(2k-1)$ ребро в граф дополнение. Действительно, ведь если существует индуцированное расширение $x, p_1, ...p_k, y$, то нет ребер между y и $x, p_1, ...p_{k-1}$, и нет ребер между x и $y, p_k, p_{k-1}, ...p_2$. Кроме того знаем, что в графе дополнении рёбер не более $\frac{n(n-1)}{2}$. Таким образом, если t - количество рекурсий D, то мы получаем оценку:

$$\frac{(k-2)(k-1)}{2} + t(2k-1) \leq \frac{n(n-1)}{2}$$

Которая после несложных операций приводится к

$$t \leq \frac{n(n-1) - (k-2)(k-1)}{2(2k-1)}$$

Ниже приводится таблица значений ассимптотики при некоторых k.

Значение k	Ассимптотическая оценка
$k = 1$	$O(n^2)$
$k = \frac{n}{4}$	$O(\frac{15}{16}n)$
$k = \frac{n}{2}$	$O(\frac{3}{8}n)$
$k = \frac{n}{2}$	$O(\frac{3}{8}n)$
$k = \frac{3}{4}n$	$O(\frac{7}{48}n)$
$k = n$	$O(1)$

Оценку можно улучшить. Например заметить что граф связный, а потому в графе-дополнении будет не более $\frac{(n-2)(n-1)}{2}$ рёбер, но ассимптотических улучшений это не дает.

Все оценки до этого момента касались только числа рекурсий D. Теперь наконец попробуем понять, что же там с C. Будем представлять граф на котором сейчас работает функция следующим способом - вершина графа это дерево полученное операциями C (Рис 0.11) . Если же была операция D, то добавим ребро между вершинами (Рис 0.12).

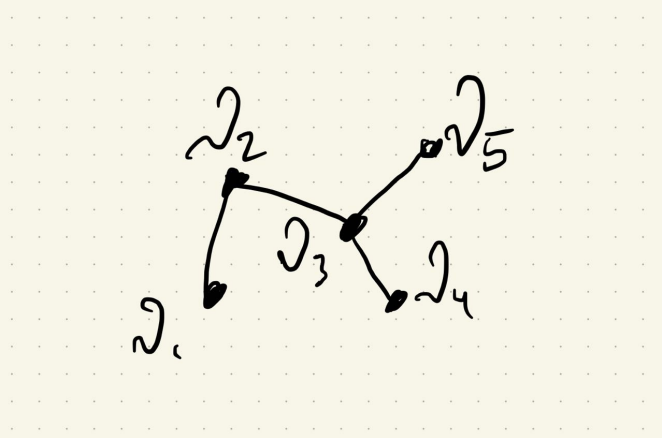

Рис. 0.11: Так например выглядит 4 подряд C - (v_1, v_2), (v_2, v_3), (v_3, v_5), (v_3, v_4).

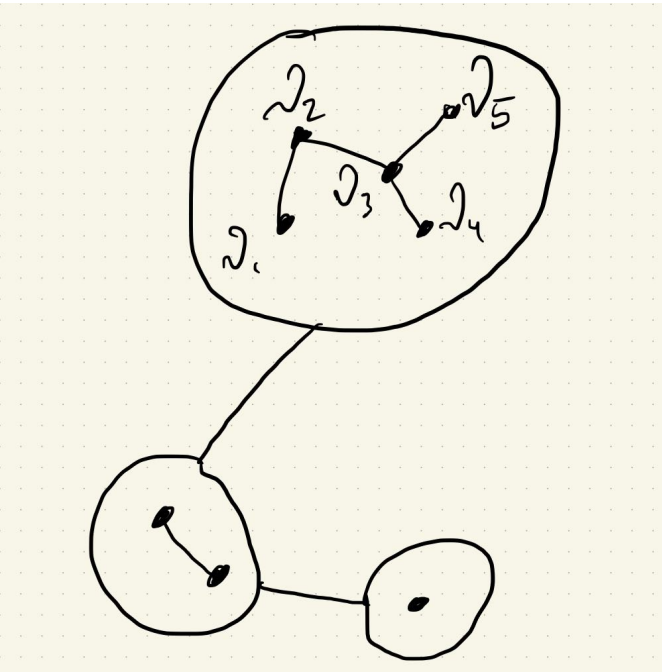

Рис. 0.12: А вот так выглядит следующая последовательность преобразований графов: $4C$, D, C, D.

Узел рекурсии - последовательность деревьев $T_1, T_2, ...T_s$, расстояние между которыми как минимум 2. Кроме того, узел задает путь длины k, один из концов которого находится на расстоянии 2 от T_s, а весь путь лежит вне объединения окрестностей деревьев. Назовем весом узла количество вершин в объединении окрестностей $N[t_i]$ всех этих деревьев. Теперь заметим, что на каждом шаге рекурсии вес узла увеличивается хотя бы на 2. Для шага D это очевидно. Для шага C

это тоже очень просто: ведь в окрестность расширенного дерева входят и вершина y, и вершина q_k, с ней смежная (в обозначениях строки 6 процедуры 2). Так как вес узла не больше $n - k$, получаем равномерную оценку глубины рекурсии $\frac{n-k}{2}$. К сожалению с этой точки оценки глубины алгоритма сдвинуться не удалось, хотя на практике контр-примеров также не удалось построить.

Тестирования алгоритма

Мной были реализованы обе процедуры, и пока не было найдено ни одного примера с более чем полиномиальной сложностью работы от количества вершин при проведении экспериментов. Все помогающие методы (найти расширение, стянуть вершину, удалить множество соседей из графа) реализованы не оптимально, в целях экономии времени на более важные эксперименты. Если будет доказана линейность рекурсии, нужно будет оптимизировать данные методы. Кроме того, авторами статьи не упомянуто как выбирать расширения. В данный момент они выбираются случайно, что, также может быть не оптимально.

Весь код реализован на C++, компилируется со всевозможными оптимизациями. В коде есть глобальный счетчик, который считает сколько раз мы зашли в функцию. Как тестируются алгоритмы - полный перебор графов и индукционных путей, стресс-тесты и крайние случаи.

В данный момент работы происходят попытки найти контрпример к линейной сложности. В случае неудач, на основе опытов будут сформулированы доказательства, почему же на самом деле сложность линейна. В основе могут лежать полученные данные о степенях вершин, количестве ребер, количестве вершин или же некие локальные величины (количество вершин на расстоянии k от пути длины k).

Например, интуитивно кажется, что чем длиннее индуцированный путь, тем дольше будет работать алгоритм. На практике больше всего рекурсий у пути длины 2. Может показаться что их не более чем $O(n^2)$, но так как графы в рекурсии динамичны, точная оценка пока не известна.

Очевидный граф для линейной оценки при любых входных данных - простой цикл. В нем как только мы удаляем $N[p_{k+2}]$ у нас больше нет дополнений, то есть путь упрощенный. Любой упрощенный путь замыкаемый, а потому рекурсия сразу завершится.

Результаты тестов

Весь код лежит на Github, любой желающий кроме того что может с ним ознакомиться, может также запустить перебор на своих графах, чтобы найти какие-то свойства. procedure2.cpp - многопоточная программа, procedure2_no_multithred.cpp - однопоточная. Сборка в ./build.sh.

Было произведено тестирование на графе сетке, n-мерном кубе и некоторых других графах

для выявления свойств. Так как это практические тесты, то в этом разделе просто будут приведены всевозможные факты об алгоритме с небольшими комментариями.

Например, при абсолютно любых входных данных было верно следующее - чем больше k, тем быстрее отрабатывает алгоритм. То есть несмотря на то, что длинному путю сложнее "развернуться"(Рис. 0.13, 0.14), то обстоятельство что при больших k глубина рекурсии меньше оказывается решающим.

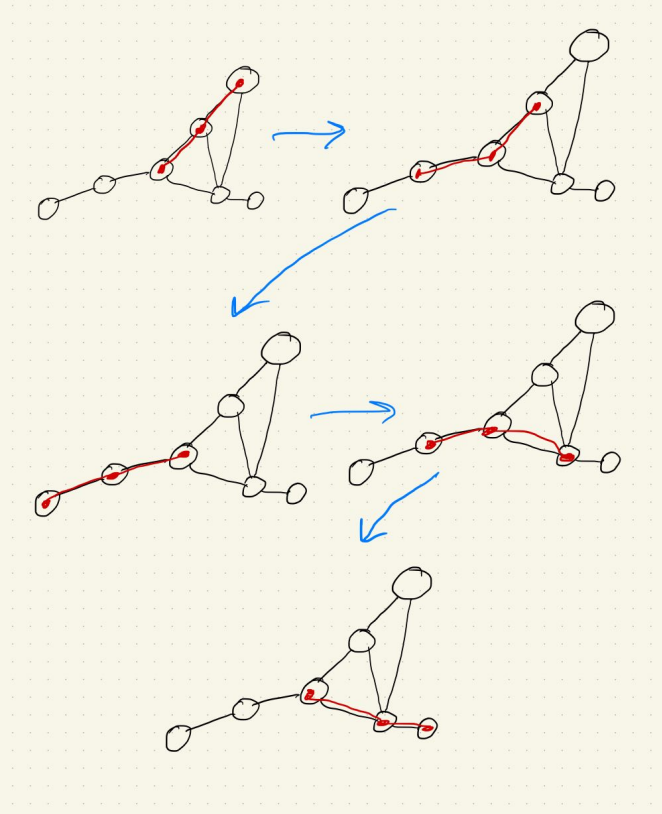

Рис. 0.13: $k = 3$. Путь "развернулся"за 4 сдвига

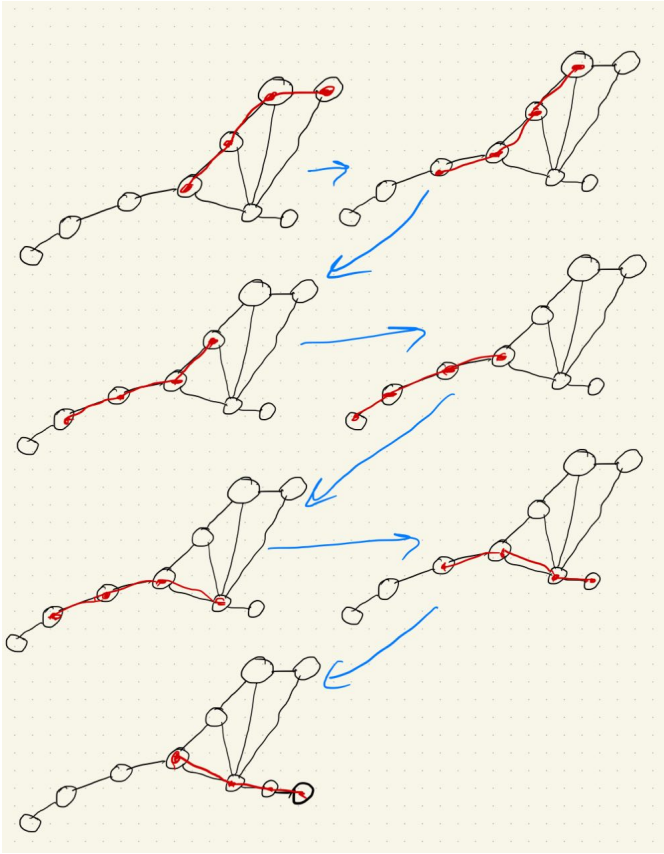

Рис. 0.14: $k = 4$. Путь "развернулся"за 6 сдвигов

Так, например, перебрав все пути длины 3 в 8-мерном кубе, максимальное количество рекурсий - 90. А среди путей длины 2 - 707. По всем вышеописанным причинам далее будут обсуждаться только пути длины 2 - если для них доказать полиномиальность алгоритма, то почти наверное и для путей любой длины это будет верно.

Ниже приводится таблица максимального количества рекурсий алгоритма при $k = 2$ среди всевозможных путей на n-мерном кубе при разных n.

Значение n	Количество вершин в графе	Максимальное количество рекурсий
$n = 2$	4	0
$n = 3$	8	3
$n = 4$	16	25
$n = 5$	32	74
$n = 6$	64	162
$n = 7$	128	427
$n = 8$	256	707
$n = 9$	512	2469
$n = 10$	1024	9625

Точно так же, приведем таблицу максимального количества рекурсий алгоритма при $k = 2$ среди всевозможных путей на сетке $n \times n$ при различных n

Значение n	Количество вершин в графе	Максимальное количество рекурсий
$n = 10$	100	95
$n = 15$	225	158
$n = 20$	400	188
$n = 25$	625	317
$n = 30$	900	373
$n = 35$	1225	513
$n = 40$	1600	625
$n = 45$	2025	797
$n = 50$	2500	978

Так как мы точно знаем, что глубина дерева не более $\frac{n-k}{2}$, то имеет смысл посмотреть на количество листьев. Кроме того, знаем, что в дереве рекурсий количество листьев - это количество разветвлений + 1. Разветвление в данном случае - из одной вершины графа рекурсий была применена и операция C и операция D. Из полученных результатов видно, что количество разветвлений крайне мало, тем не менее хочется привести цифры.

Например, в том же графе сетке 30×30, у тех же изначальных входных данных, при которых получилось максимальное количество рекурсий - всего 1 вершина, из которой вызвались оба ребенка. 246 раз был только переход D, 124 только переход C. Максимальное количество разветвлений в этом графе же составляло 3, оно было достигнуто при другом входном пути. При этом рекурсивных вызовов было всего 255.

В графе 8-мерном кубе тем временем, максимальное количество разветвлений - 45. Из этого сразу понятно, почему алгоритм завершается полиномиально. Но вот почему разветвлений столько и как оценить их количество оказалось нетривиальной задачей.

Может быть если посмотреть как устроен алгоритм внутри все станет понятно? Например запустим его и отметим все ребра, которые входили в какой-нибудь путь. Как будет выглядеть подграф? Нередко это дерево. Нередко, вершин в нем столько же сколько и ребер. Однако случается и такое(Рис. 0.15):

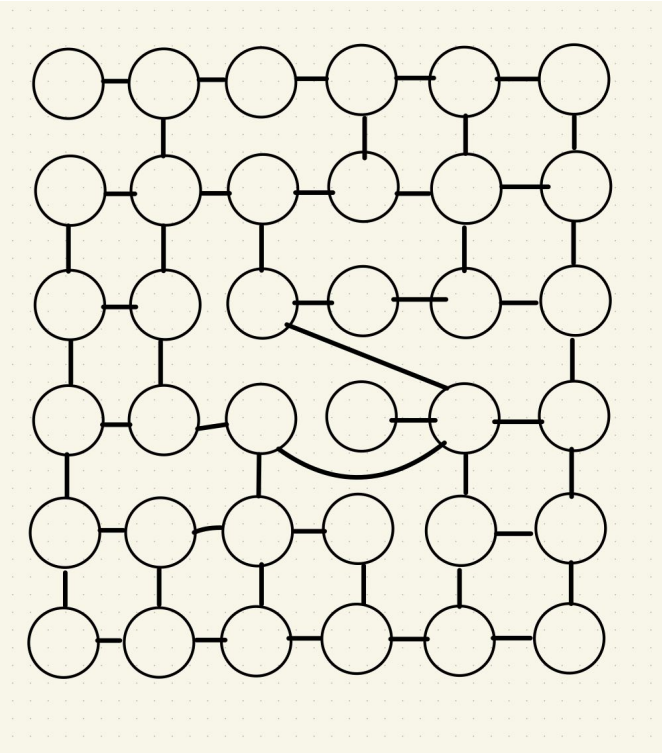

Рис. 0.15: Подграф полученный алгоритмом, запущенным на сетке 6×6.

У таких страшных подграфов к сожалению нет разумных свойств для полиномиальной оценки.

Тут также хочется упомянуть некоторые опровергнутые некорректные идеи:

1) Если мы хоть раз встретили дополнение xPy, то больше в алгоритме нам не встретится та же пара вершин x, y в дополнении какого-то другого рассматриваемого пути P'.

2) Ребро xy может быть добавлено в рассматриваемый путь в алгоритме единожды среди всех вершин рекурсии.

3) Количество вершин в графе рекурсии в поддереве D больше или равно количеству вершин в поддереве C той же вершины.

4) Количество вершин в графе рекурсии в поддереве D больше или равно количеству вершин в поддереве D поддерева C той же вершины.

5) Один и тот же путь P не рассматривается в разных подграфах алгоритма

Новый алгоритм

В прошлом алгоритме рекурсия была нетривиальной - например, путь который мы используем при вызове C, мы получаем из рекурсии D. То есть состояние не начальное и меняется после того как мы вернулись из D. А может быть нам вовсе и не нужна ветка D? Попробуем рассмотреть все случаи и сделать нечто похожее на C. К сожалению алгоритм работает только для графов без

индуцированных циклов длины $k+2$. Об этом еще будет сказано после его описания. Алгоритм поддерживает множество вершин U, которое индуцирует связный подграф, и индуцированный путь $p_1, \dots p_k$, из вершин которого не идет ребер в U, но при этом есть такая вершина p_{k+1} в $N[U]$, что $p_1, ..., p_k, p_{k+1}$ - индуцированный путь. Алгоритм использует полиномиальность проверки на индуцированную замыкаемость. На вход нашему алгоритму также поступает индуцированный путь длины k.

Инициализация: Проверяем на замыкаемость путь. Если путь замыкаем, то вернем $p_1, p_2, ...p_k$ - мы нашли нужный нам путь. Иначе существует незамыкаемое расширение. Пусть это $p_0, p_1, ...p_k, p_{k+1}$. Тогда $U = p_{k+1}$ и теперь рассматриваем путь $p_0, p_1, ...p_{k-1}$.

Основной шаг:

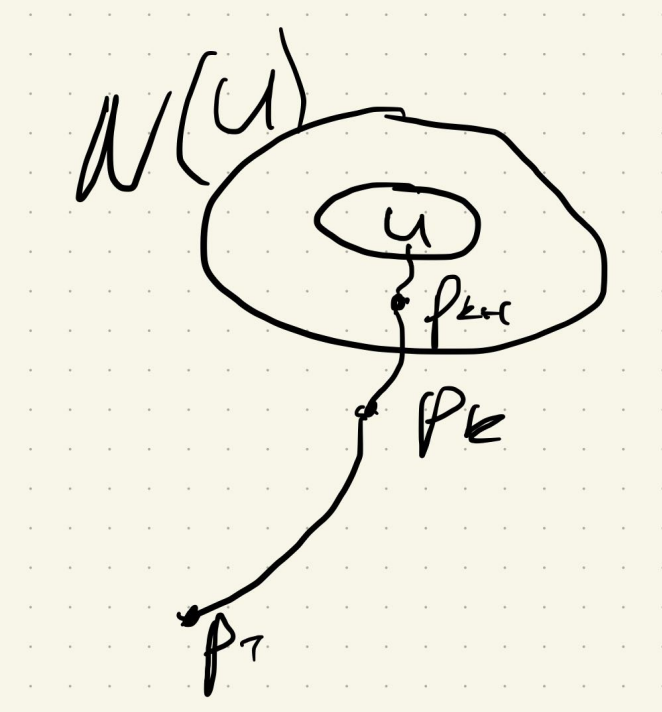

Рис. 0.16: $p_{k+1} \in N[U]$, $p_1, ...p_k, p_{k+1}$ - индуцированный путь

$p_1...p_k$ - индуцированный путь вне $N[U]$. Существует ребро (p_k, p_{k+1}), такое, что $p_{k+1} \in N[U]$, а значит - $p_1, ...p_{k+1}$ - индуцированный путь.

Если $p_1, ...p_k$ замыкаем - победа. Иначе, точно так же, существует незамыкаемое расширение. Рассмотрим всевозможные варианты.

1) Если в расширении x и $y \in N[U]$ (Рис 0.17), то оно замыкаемо, так как можем замкнуть через U. Значит такой случай невозможен.

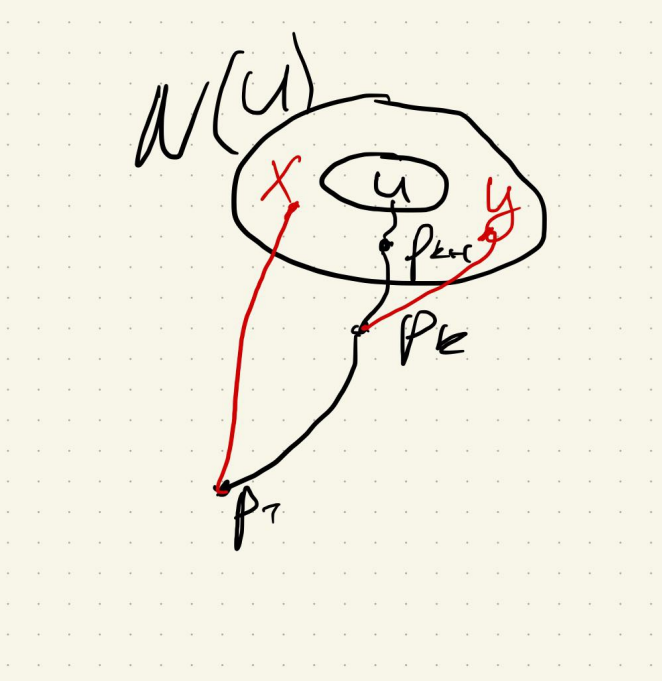

Рис. 0.17: X и Y лежат в $N[U]$, а значит можем замкнуть следующим образом: $x, U, y, p_k, ...p_1$

2) Если в расширении x не принадлежит $N[U]$ и не имеет ребра с p_{k+1}(y может быть где угодно, Рис 0.18), то рассматриваем путь $x, p_0, ...p_{k_1}$ и добавляем p_{k+1} в множество U.

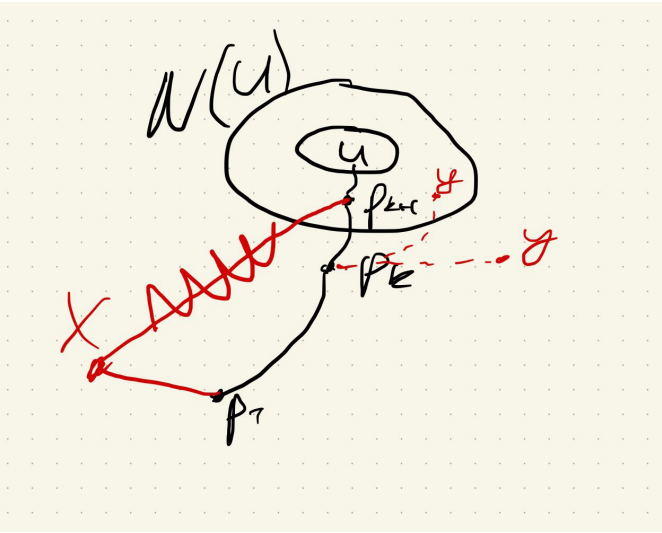

Рис. 0.18: X не связан ребром с p_{k+1}, y лежит где угодно $N[U]$

3) Если в расширении x не принадлежит $N[U]$ и связан ребром с p_{k+1}, то существует индуцированный цикл длины $k+2$, что невозможно.

4) Остался последний случай - $x \in N[U]$ и y не лежит в $N[u]$(Рис 0.19). Тут все просто - путь $y, p_k, ...p_2$, в U добавляем X.

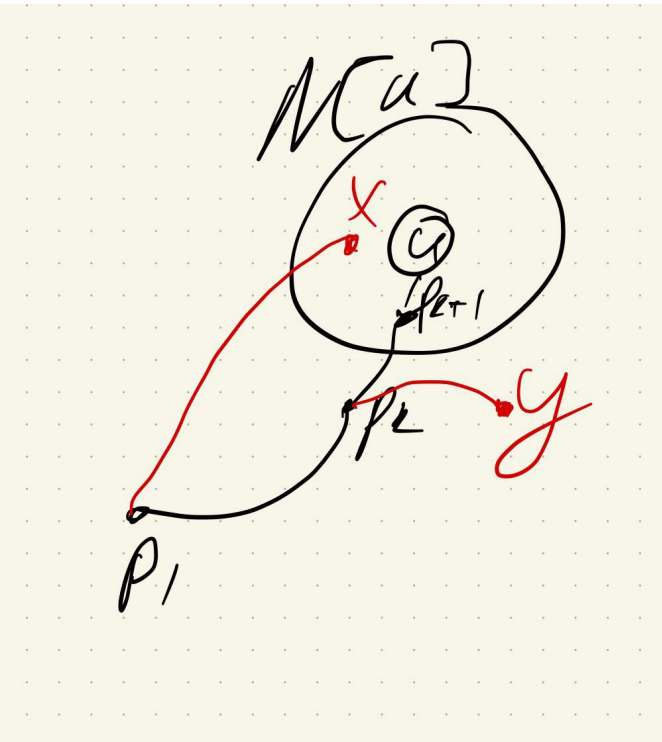

Рис. 0.19: x лежит в $N[U]$, y не лежит

Алгоритм полиномиален. Переходов - $O(N-K)$, ведь каждый раз размер множества увеличивается на 1. Поиск незамыкаемого расширения, тоже полиномальная задача. Например, всего расширений $O(N^2)$. Как найти незамыкаемое - обычным поиском в глубину ищем любой путь от вершины x до вершины y в графе $G - N[P]$. А значит, за $O(N^2(N+E))$ умеем находить незамыкаемое расширение. Таким образом итоговая ассимптотика зависящая только от G составляет $O(N^3(N+E))$

Псевдокод

Procedure 1 ShiftingWithSet(G, P)

Require: a graph G, induced path $P = p_1 p_2 \dots p_k$ in G, U
Ensure: a sequence S of paths shifting P to an avoidable path in G

if there exists an extension xPy of P not contained in any induced cycle, such that $x \notin N[U]$ **then**
    $Q \leftarrow xp_0 \dots p_{k-1}$
    $U' = U + \{p_{k+1}\}$
    return P, ShiftingWithSet(G, Q, U')
if there exists an extension xPy of P not contained in any induced cycle, such that $x \in N[U]$ and $y \notin N[U]$ **then**
    $Q \leftarrow yp_k \dots p_1$
    $U' = U + \{x\}$
    return P, ShiftingWithSet(G, Q, U')
return P

Общий алгоритм из полученного

Заметим, что единственное в чем нам нужно ограничение отсутствия индуцированных циклов длины $k+2$ - 3 пункт алгоритма. Но почему? Посмотрим, как мы пробовали сделать этот самый третий пункт.

3) Если в расширении x не принадлежит $N[U]$ и связан ребром с p_{k+1}(y не принадлежит $N[U]$, иначе путь замыкаем через U), то мы не можем сделать то же самое что и в шаге 2, ведь инвариант сломается (U будет связано ребром с p_0).

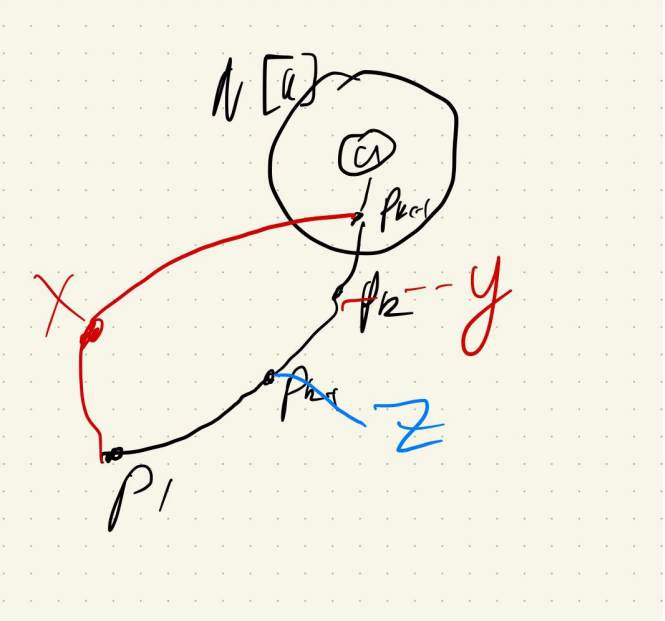

Рис. 0.20: X связан ребром с p_{k+1}, y не принадлежит $N[U]$. Z может лежать или не лежать, рассматриваем это ниже

Самый сложный случай. Знаем что у $x, p_1, ...p_{k-1}$ есть незамыкаемое расширение. Если Z не лежит в $N[U]$, то переход это путь $p_1, ...p_{k-1}, Z$ и в U добавляем p_{k+1}. Инвариант сохранен. Если же $Z \in N[U]$, то новый путь это $x', x, p_1, ...p_{k-2}$ - новый путь (p_{k-1}, z - замыкается через U, а значит существует $x' \neq p_{k+1}$). А в U теперь входит Z.

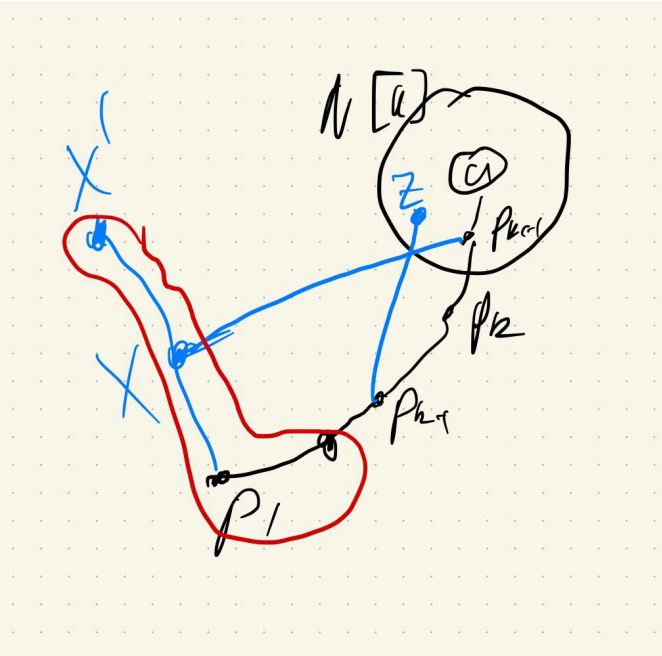

Рис. 0.21: Красным обозначен итоговый путь. $Z \in N[U]$

Что же с ним не так? Читатель может попробовать найти это самостоятельно, но все же опишем.

Z не обязательно существует! Действительно, мы нигде не гарантировали себе существование Z. Возможно, что незамыкаемым расширением было $X', X, p_1, ...p_{k-1}, p_k$! Если Z не существует переход моментально рушится.

Тем не менее, по данному переходу можно заметить, что алгоритм может работать и на графах с индуцированным циклом длины $k+2$, но вот гарантировать корректность не получится.

Можно попробовать довести алгоритм, например у опытного читателя может возникнуть описанная идея. Поставим пункт 3 после 4го. То есть сначала будем искать незамыкаемые расширения из 4 пункта, а затем из 3го. Что это нам дает? А например следующее: рассмотрим путь $p_1, x, p_{k+1}, p_k \dots p_4$. У него тоже должно существовать незамыкаемое расширение. Заметим, что у p_1 этим расширением может быть только p_2. Действительно, если мы проверили все незамыкаемые расширения на свойства до этого, то теперь оставались только те расширения, где x - сосед p_{k+1}. Из этого сразу же следует, что у вершины p_4 есть Z вне нашего пути.

К сожалению даже это рассуждение не является корректным. Например, p_1 имеет соседа T, связанного с p_3. Такое возможно, ведь условия были для расширений $p_1, ...p_k$. Тем, что мы теперь обходим этот путь по-другому, мы разрешаем такое расширение.

И даже тут кажется, но ведь тогда путь обходим! Ведь мы сможем обойти его через T, p_3, p_4. И это означает что у p_4 все же есть сосед вне пути. И, это конечно правда. Но этот сосед тоже может быть связан с p_2 или p_3, а значит никак не применим в изначальном пути $p_k, ...p_1$.

Попробуем переделать алгоритм. Что если хранить множество множеств U? Например, скажем что теперь ищем замыкаемый путь в графе $G - N[U]$, а новый U и P зададим как X и $y, p_k, ...p_1$ соответственно? Можно заметить, что такое множество из $N[U_i]$ будет деревом, ведь p_{k+1} был связан с x. Ведь тогда, если расширение ведут в разные U_i - это дерево а значит расширение замыкаемо. На самом деле, заметим, что в данном переходе p_k будет связан с $p_{k+1} \in U_0$. То есть если существует ребро в U_0 из нашего пути, то оно вполне может быть незамыкаемо - ведь цикл проходит через p_{k+1}, который может быть связан с вершиной индуцированного пути.

Эти мысли показывают, что данный алгоритм скорее всего невозможно довести до общего. Тем не менее, задача была сильно сведена. Ведь теперь, осталось лишь найти полиномиальный алгоритм который будет работать для графов с индуцированным циклом длины $k + 2$. Несмотря на то, что нахождение таких циклов в задаче NP-полная задача, мы можем для начала воспользоваться нашим алгоритмом, и в случае 3 - мы нашли индуцированный цикл длины $k + 2$! А значит сможем перейти к алгоритму решающему задачу на нем.

Результат

В результате работы, мы доказали лучшую на данный момент оценку алгоритмов поиска замыкаемого пути при помощи сдвигов $O(2^{\frac{n-k}{2}})$. Кроме того мы привели некоторые свойства алгоритма. Было проведено тестирование алгоритма, результаты которого также могут помочь для достижения лучшей оценки.

Также мы привели полиномиальный алгоритм, с ассимптотикой $O((N - K)^3(N - K + E))$ решающий данную задачу на графах без индуцированных циклов длины $k + 2$.

Список литературы

[1] Marthe Bonamy, Oscar Defrain, Meike Hatzel и Jocelyn Thiebaut. “Avoidable Paths in Graphs”. В: *Electron. J. Comb.* 27.4 (2020), с. 4. doi: 10.37236/9030. url: https://doi.org/10.37236/9030.

[2] Vladimir Gurvich, Matjaz Krnc, Martin Milanic и Mikhail N. Vyalyi. “Shifting paths to avoidable ones”. В: *J. Graph Theory* 100.1 (2022), с. 69—83. doi: 10.1002/jgt.22766. url: https://doi.org/10.1002/jgt.22766.

[3] West D. B. “Introduction to graph theory”. В: *Upper Saddle River : Prentice hall*. 1996.

[4] Vladimir Gurvich, Matjaz Krnc, Martin Milanic и Mikhail N. Vyalyi. “Avoidability beyond paths”. В: *CoRR* abs/2208.12803 (2022). doi: 10.48550/arXiv.2208.12803. arXiv: 2208.12803. url: https://doi.org/10.48550/arXiv.2208.12803.