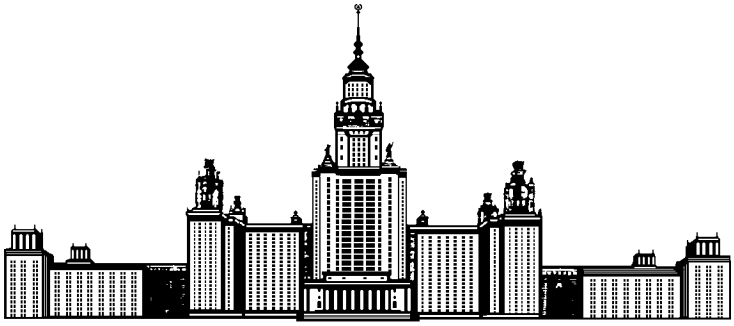
Московский государственный университет им. М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра суперкомпьютеров и квантовой информатики



МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему:

АЛГОРИТМЫ ПОИСКА ПОДГРАФОВ ТИПА «ЧЕРНАЯ ДЫРА» В ОРИЕНТИРОВАННОМ ГРАФЕ

01.04.02 Прикладная математика и информатика

Выполнил:
студент гр. М-218
Иванов Денис Евгеньевич

Научный руководитель:
к.т.н. Семенов Александр Сергеевич

Рецензент:
Фролов Александр Сергеевич,
начальник отдела в АО «НИЦЭВТ»

Москва, 2020

Содержание

Аннотация

Магистерская диссертация на тему: Алгоритмы поиска подграфов типа «черная дыра» в ориентированном графе.

Выполнил: студент гр. М-218 Иванов Денис Евгеньевич.

Научный руководитель: к.т.н. Семенов Александр Сергеевич.

Работа состоит из 6 глав и списка использованных источников.

В первой главе раскрывается актуальность выбранной темы.

Во второй главе дается определение черной дыры и постановка задачи поиска этоого паттерна на графе.

В третьей главе описываются существующие решения поставленной задачи, опубликованные до сих пор другими авторами. В частности, описывается алгоритм iBlackhole и его модификация iBlackholeDC, а также алгоритм BruteForce.

В четвертой главе описан процесс разработки нового алгоритма TopSortBH, его модификации SkipFast и комбинированного алгоритма TopOver. Доказываются сопутствующие теоремы. Приводятся детали реализации разработанных алгоритмов и способы их оптимизации. Раскрываются аспекты параллелизма в предложенных решениях.

В пятой главе приводятся результаты экспериментальных исследований всех рассмотренных в работе алгоритмов, как ранее опубликованных, так и впервые представленных в данной работе.

В шестой главе приводятся достигнутые в ходе работы результаты и формулируются выводы.

Работа содержит 43 страницы текста, 1 таблицу, 15 рисунков, 7 алгоритмов, 6 листингов. В тексте работы использованы 12 источников литературы.

1 Введение

Одним из важных свойств графов, которые описывают объекты реального мира, является присутствие в них кластеров или сообществ.

Задача поиска этих сообществ имеет важное значение в социологии, биологии и компьютерных науках, поскольку объекты, изучаемые в данных прикладных областях, часто естественно моделируются при помощи графов [5].

В работе [9] впервые сформулирована задача поиска подграфов (паттернов) типа «черная дыра» и «вулкан» в ориентированных графах. В соответствии с этой работой, черная дыра – слабо связное множество вершин графа, у которого отсутствуют исходящие ребра. Вулкан – это, наоборот, слабо связное множество вершин графа, у которого имеются только исходящие ребра.

Паттерны черная дыра и вулкан встречаются в прикладных задачах. Например, в торговой сети, паттерн черная дыра может представлять собой группу торговцев, которые манипулируют рынком. Также черная дыра и вулкан могут описывать собой схемы отмывания денег [10]. Поиск черных дыр и вулканов в реальном времени может своевременно обнаружить пагубные явления, такие как природные катаклизмы, катастрофы, происшествия. Разработка алгоритмов поиска таких паттернов позволяет сохранять общественную безопасность [6].

Основной вклад в исследование задачи поиска паттерна черная дыра внесла группа авторов из университета Нью-Джерси. Они сформулировали задачу и предложили алгоритм поиска черных дыр в случае невзвешенного ориентированного графа [9]. Два года спустя эти же авторы опубликовали алгоритм для приближенного поиска черных дыр в случае взвешенного графа [8, 7]. Hong [6] рассматривает задачу поиска черных дыр в реальном времени для нужд современного города. В этой работе предлагается оригинальный алгоритм, приближенно решающий задачу для динамических графов. Другие работы, посвященные поиску паттерна «черная дыра» в графе, автору данной работы неизвестны.

В данной работе рассматривается упрощенная постановка задачи поиска черных дыр для случая ориентированного графа без весов.

2 Постановка задачи

Рассмотрим граф $G = G(V, E)$, где V – множество вершин, а E – множество ребер.

Определение 1. Подмножество вершин $B \subseteq V$ называется черной дырой, если одновременно выполняются следующие условия: 1) подграф $G'(B, E')$ слабо связан и 2) нет такой пары вершин (u, v) что $u \in B,\ v \in V \setminus B;\ (u, v) \in E$.

Задача состоит в том, чтобы в ориентированном графе без весов найти как можно больше черных дыр (или рассмотреть максимальное количество кандидатов) за ограниченное время.

Для решения данной задачи потребуется проделать следующие шаги:

- Изучить существующие решения;
- Разработать новый конкурентноспособный алгоритм для поиска черных дыр;
- Разработать параллельные реализации всех представленных алгоритмов;
- Провести сравнительное оценочное тестирование представленных подходов на больших, по сравнению с рассмотренными другими исследователями, графах.

3 Существующие алгоритмы решения задачи поиска черных дыр в графе

3.1 Определения

Введем несколько дополнительных определений.

Определение 2. Пусть дан ориентированный граф $G(V, E)$. Последовательность $v_0v_1v_2...v_k$, состоящая из вершин $v_i \in V,\ 0 \leq i \leq k$ образует ориентированный путь из v_0 в v_k, если существуют ребра $(v_{i-1}, v_i) \in E,\ 1 \leq i \leq k$ и $v_i \neq v_j$ для всех $0 \leq i,\ j \leq k,\ i \neq j$. Длина такого ориентированного пути равна k.

Определение 3. Пусть дан ориентированный граф $G(V, E)$. Вершина $v \in V$ достижима из $u \in V$, если существует ориентированный путь, который начинается в u и заканчивается в v.

Определение 4. Пусть дан ориентированный граф $G(V, E)$. Если вершина $v \in V$ достижима из $u \in V$ тогда u является ориентированным предком v, а v является ориентированным потомком u. Если существует ребро из u в v, то u является непосредственным предком v , а v является непосредственным потомком u.

Определение 5. Подмножество вершин $B \subseteq G$ называется компонентой сильной связности (КСС), если все вершины B попарно взаимно достижимы.

Определение 6. Пусть дан ориентированный граф $G(V, E)$. Рассмотрим вершину $v \in V$. Замыкание вершины $Closure(v) = \{u \mid u \text{ достижима из } v\} \cup v$. Другими словами, замыкание вершины v – это множество всех вершин, достижимых из v, включая ее саму.

Следующие утверждения были ранее доказаны в статье [9].

Утверждение 1. *Если вершина $v \in B$, причем $B \subseteq V$ – черная дыра, то все потомки v лежат в B.*

Утверждение 2. *Если вершина $v \in B$, причем $B \subseteq V$ – черная дыра, то замыкание v полностью содержится в B.*

Утверждение 3. *Замыкание любой вершины образует черную дыру.*

3.2 Алгоритм BruteForce

Алгоритм 1: BruteForce

Input : $G(V, E)$ – ориентированный граф, V – множество вершин, E – множество ребер,
n – максимальное число вершин, которое может содержать черная дыра

Output: *Blackholes* – множество всех черных дыр графа размером от 1 до n

```
Blackholes = ∅, C_0 = ∅
for i = 1 to n do
    foreach B ∈ V(i) do
        if G(B) – слабо связан then
            if d_out(B) == 0 then
                Blackholes = Blackholes ∪ B
            end
        end
    end
end
return Blackholes
```

Наиболее простым способом решения задачи является алгоритм 1 BruteForce. Он заключается в последовательном переборе наборов вершин размером от 1 до N. Каждый набор проверяется на соответствие определению черной дыры (см. Опред. 1). Проверяется наличие свойства слабой связности, а затем и отсутствие исходящих ребер. Такой подход никак не учитывает особенности внутреннего устройства черных дыр при переборе, и поэтому рассматривает большое количество избыточных вариантов. В частности, необходимости рассматривать наборы, которые заведомо имеют исходящие ребра, можно избежать, что будет продемонстрировано позже.

3.3 Алгоритм iBlackhole

В 2010 году в статье [9] был предложен алгоритм iBlackhole.

Алгоритм 2: iBlackhole

Input : $G(V, E)$ – ориентированный граф, V – множество вершин, E – множество ребер,
n – максимальное число вершин, которое может содержать черная дыра
Output: *Blackholes* – множество всех черных дыр графа размером от 1 до n

```
Blackholes = ∅, C_0 = ∅
for i = 1 to n do
    P_i = {v | d_out(v) < i}                    // d_out(v) количество исходящих из v ребер
    foreach v ∈ P_i do
        if v ∉ C_{i-1} then
            if как минимум один из непосредственных потомков v не
              принадлежит P_i then
                удалить v из P_i
                удалить всех предков v из P_i
            end
        end
    end
    C_i = P_i                                   // C_i - список кандидатов
    foreach v ∈ C_i do
        if |Closure(v)| == i then
            Blackholes = Blackholes ∪ Closure(v)
        end
        if |Closure(v)| >= i then
            удалить v из _i
            удалить всех предков v из _i
        end
    end
    /* Применить алгоритм полного перебора к F_i(i) - множеству всех
       подмножеств F_i, содержащих i вершин                          */
    foreach B ∈ F_i(i) do
        if G(B) – слабо связан then
            if d_out(B) == 0 then
                Blackholes = Blackholes ∪ B
            end
        end
    end
end
return Blackholes
```

Алгоритм 2 создан для поиска черных дыр заданного размера, но также позволяет найти в графе черные дыры всех размеров. Идея авторов из университета Нью-Джерси состоит в том, чтобы на каждой итерации исключать из рассмотрения вершины, которые заведомо не могут образовывать черные дыры заданного размера, тем самым сокращая перебор. К вершинам, которые прошли фильтр на данной итерации, применяется алгоритм BruteForce, описанный выше. Итерации алгоритма идут по возрастанию размера черных дыр. Это позволяет постепенно наращивать размер множества

кандидатов.

Алгоритм 3: iBlackholeDC

Input : $G(v, E)$ – ориентированный граф, V – множество вершин, E – множество ребер,
n – максимальное число вершин, которое может содержать черная дыра
Output: *Blackholes* – множество всех черных дыр графа размером от 1 до n
$Blackholes = \emptyset, C_0 = \emptyset$
for $i = 1$ *to* n **do**
  $P_i = \{v | d_{out}(v) < i\}$
  убрать лишние вершины из P_i и сформировать список кандидатов C_i
  убрать лишние вершины из C_i и сформировать список кандидатов F_i
  foreach $WCC \in G(F_i)$ **do**
    /* Применить алгоритм полного перебора к $WCC_i(i)$ - множеству всех подмножеств WCC_i, содержащих i вершин */
    foreach $B \in WCC_i(i)$ **do**
      if $G(B)$ – *слабо связан* **then**
        if $d_{out}(B) == 0$ **then**
          $Blackholes = Blackholes \cup B$
        end
      end
    end
  end
end
return *Blackholes*

В той же работе авторы предлагают модификацию алгоритма: iBlackholeDC. DC здесь означает «Divide and Conquer», т.е. «Разделяй и Властвуй». На стадии полного перебора авторы предлагают рассматривать только наборы вершин полностью содержащиеся внутри компонент слабой связности графа, индуцированного множеством кандидатов. Это, само по себе, не гарантирует, что такой набор будет удовлетворять требованию слабой связности, но избавляет от необходимости рассматривать заведомо несвязные наборы. Например, если рассмотреть две вершины, находящиеся в разных компонентах слабой связности, то между ними не найдется направленного пути.

Алгоритм iBlackholeDC хорошо работает для поиска небольших черных дыр в любых графах. Если же граф не содержит черных дыр малого размера, или он содержит крупные компоненты сильной связности, то алгоритм iBlackholeDC долгое время будет безрезультатно перебирать наборы вершин меньшего размера. Так как алгоритм BruteForce используется как составная часть, то можно говорить о наследовании проблем алгоритма BruteForce алгоритмом iBlackholeDC.

3.4 Известные проблемы

В данном разделе описаны примеры, демонстрирующие слабые стороны алгоритма iBlackholeDC [9].

Введем дополнительное утверждение, чтобы описать проблемы существующего подхода.

Утверждение 4. *Все вершины КСС принадлежат одной черной дыре.*

Доказательство. В КСС все вершины попарно взаимно достижимы по определению. Это значит, что замыкание произвольной вершины КСС содержит всю КСС как подмножество. Опираясь на Лемму 3, получаем, что КСС принадлежит той же черной дыре, что и вершина v. □

Рассмотрим следующий пример. Граф состоит из 10^6 вершин, которые образуют единую КСС. Это значит, что в данном графе существует только одна черная дыра – это весь граф. Если о структуре графа ничего не известно, а также нет достаточно больших вычислительных ресурсов, возникает необходимость перебирать слишком большое число размеров черных дыр. В худшем случае нам придется перебрать все возможные размеры множеств кандидатов. В реальности, при использовании iBlackholeDC придется ждать слишком долго, чтобы в итоге получить тривиальный ответ.

Другой пример – это графы малого мира (SmallWorld) [12]. Большая часть вершин такого графа содержится в единственной большой КСС. Также имеется несколько вершин и корней. Нетрудно проверить, что при просмотре от меньших размеров черной дыры к большим, алгоритм iBlackholeDC быстро обнаружит все небольшие черные дыры. Далее он будет очень долго пытаться найти черные дыры внутри КСС, где их попросту нет, пока не доберется до размеров больше размера КСС.

Алгоритм iBlackholeDC:

- Неэффективно работает со SmallWorld графами;
- Каждую итерацию завершает перебором, часто просматривая заведомо неверные варианты;
- Не был испытан авторами на графах размером более 1500 вершин.

4 Разработка алгоритмов поиска черных дыр в графе

В данной работе предлагается рассмотреть решение задачи поиска черных дыр при помощи комбинации двух этапов. Первый этап состоит в предобработке графа, задача которой упростить структуру графа. Уменьшение размера графа значительно сокращает размер задачи. Это важно в силу комбинаторной природы задачи. Второй этап – это непосредственно поиск черных дыр. Здесь не получится избежать перебора в том или ином виде, однако этот перебор можно построить по-разному. Целью данного этапа является обнаружение как можно большего числа черных дыр при заданном ограничении по времени.

4.1 Предобработка графа

Как было сказано выше, большие КСС требуют значительных вычислительных ресурсов для обработки и при этом не образуют дополнительных черных дыр. Это может быть критично при обработке графов малого мира, потому что одна КСС может включать до 100% вершин такого графа. При работе с подобными графами будет удобно рассмотреть КСС как единую вершину, которая объединяет в себе все входящие и исходящие связи периферических вершин КСС. Процесс сжатия каждой КСС до единственной вершины известен под названием конденсация графа.

Определение 7. Если каждую компоненту сильной связности заменить единственной вершиной, то получится ориентированный ациклический граф, который называется конденсацией исходного графа.

В данной работе для поиска компонент сильной связности используется алгоритм Шарира [11].

Конденсированный граф имеет следующие свойства:

- Ориентированный;
- Без циклов;
- Размер меньше, чем исходный граф.

Определение 8. Пусть дан граф $G(V, E)$, где V – множество вершин, E – множество ребер. Масштабом графа называется показатель степени двойки S в выражении $|V| = 2^S$.

Предобработка занимает $O((N + M)log(M))$. При больших значениях N, M процедура может продолжаться несколько минут, что будет показано позднее в разделе 5.1.

4.2 Подход к перебору черных дыр

После предобработки графа происходит переход к этапу поиска черных дыр. Решаемая задача имеет комбинаторную природу, поэтому на данном этапе будет полезно уменьшить число потенциальных кандидатов. Опишем сначала подход к организации полного перебора, который учитывает особенности строения черных дыр.

Определение 9. Корень черной дыры $B \subseteq V$ – это такая вершина $v \in B$, что не найдется ребра $(u, v) \in E$, где $u \in B,\ u \neq v$.

Определение 10. Множество всех корней черной дыры называется базисом черной дыры.

Отметим, что базис черной дыры может состоять из произвольного числа вершин, отличного от нуля.

Алгоритм 4: BasisBruteForce

Input : $G(V, E)$ – ориентированный граф, V – множество вершин, E – множество ребер,
n – максимальное число вершин, которое может содержать черная дыра
Output: *Blackholes* – множество всех черных дыр графа размером от 1 до n

```
Blackholes = ∅
for i = 1 to n do
    foreach C ∈ V(i) do
        B = ⋃_{v∈C} Closure(v)
        if G(B) – слабо связан then
            Blackholes = Blackholes ∪ B
        end
    end
end
return Blackholes
```

Алгоритм 4 BasisBruteForce показывает, как можно организовать перебор, основываясь на предыдущих утверждениях. Во время полного перебора просматриваются все возможные неупорядоченные подмножества вершин графа. Сначала размера 1, потом 2 и так далее. Далее для каждой вершины очередного подмножества строится ее замыкание. Объединение всех таких замыканий является потенциальной черной дырой. Если кандидат является слабо связным подграфом, тогда это черная дыра. На рисунке 1 проиллюстрированы описанные действия.

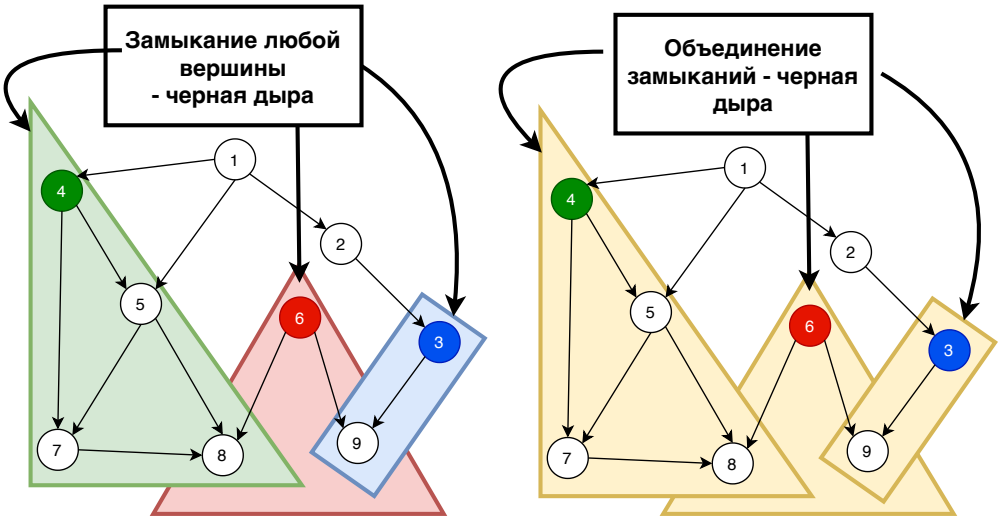


Рис. 1: Построение черной дыры по множеству ее корней. Слева три черных дыры имеющих по одному корню. Справа одна черная дыра с тремя корнями.

По сравнению с BruteForce, описанный вариант перебора не требует проверять число исходящих ребер для периферических вершин. Так как, по определению замыкания, оно уже содержит все достижимые вершины, то точно известно, что больше не найдется ни одного исходящего ребра. К сожалению, такой подход не застрахован от повторного обнаружения уже известной черной дыры. Для уменьшения количества повторных обнаружений уже известных дыр предлагается использовать информацию о топологии графа.

4.3 Сокращение перебора при помощи анализа топологии графа

Согласно определениям, данным выше, если какая-то вершина не является корнем рассматриваемой черной дыры, то она может быть опущена, так как набор корней описывает черную дыру единственным образом. Каждая не корневая вершина в черной дыре достижима из какого-то корня черной дыры. Значит, при наличии матрицы попарной достижимости для данного графа, можно эффективно определять лишние, не являющиеся базисами, комбинации вершин. Конечно же, возникает желание сразу же вычислить такую матрицу достижимости, но это вычислительно сложная задача $O(V^3)$. Кроме того, хранение матрицы достижимости может потребовать $O(|V|^2)$ памяти, что значительно ограничит применимость такого метода. Наконец, отметим, что если дан набор вершин размера K, то в худшем случае потребуется $K^2/2$ обращений к матрице. Таким образом, прямое вычисление матрицы достижимости не выглядит разумным решением. Это может привести к нехватке как памяти, так и времени на предобработку.

Поэтому вводится правило, которое является частным случаем описанного в предыдущем абзаце, чтобы частично сократить поле перебора, не жертвуя при этом временем

ответа. Те дубликаты, которые не могут быть отфильтрованы при помощи топологии, будут проверены напрямую.

Определение 11. Топологоческая сортировка или топологическое упорядочивание ациклического орграфа – такой массив вершин, что для любого ребра $(u, v) \in E$ вершина v будет иметь индекс меньше, чем u.

Определение 12. Пусть дан ориентированный граф $G(V, E)$, корень $r \in Roots(G)$, $topSortOrder_r$ – топологическая сортировка замыкания $Closure(r)$ и $|Closure(v)|$ – размеры замыканий для всех вершин $v \in Closure(r)$, то есть доступных из r. Вершина называется особой вершиной под корнем r, если размер ее замыкания $|Closure(v)|$ равен ее индексу в массиве $topSortOrder_r$ плюс один.

Замечание 1. Заметим, что множество $Special_r$ особых вершин под корнем r полностью содержится в замыкании данного корня, то есть $Special_r \subseteq Closure(r)$

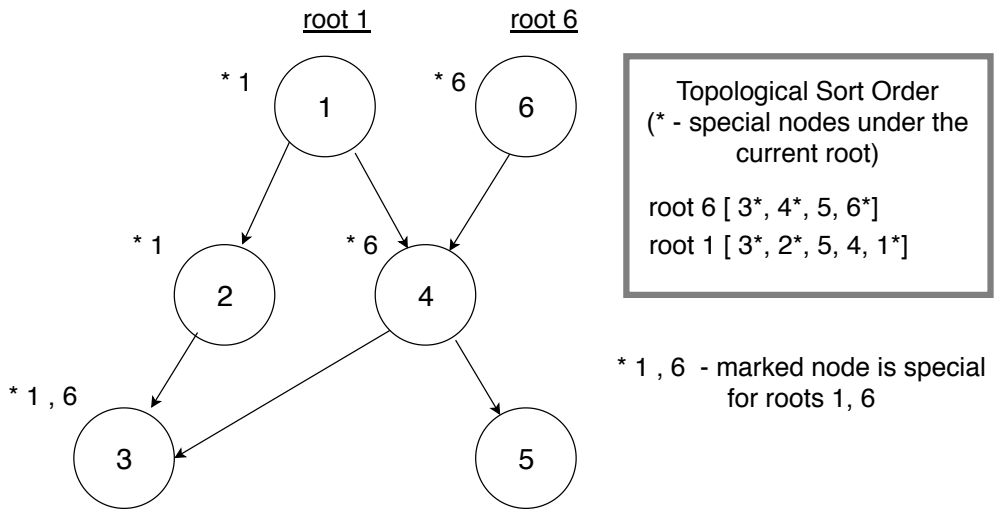


Рис. 2: Граф с двумя корнями (вершины 1 и 6), топологическая сортировка замыканий каждого из корней. Особые вершины под каждым корнем отмечены знаком * и номером соответствующего корня.

Замечание 2. Решение о том, является ли вершина особой, принимается независимо в поддереве каждого корня. Одна и та же вершина может быть особой в поддереве одного корня и не являться таковой в поддереве другого корня. Пример такой ситуации отображен на рисунке 2.

Определение 13. Пусть дан ориентированный граф $G(V, E)$, один из его корней $r \in Roots(G)$, $topSortOrder_r$ – топологическая сортировка замыкания $Closure(r)$, $v \in Special_r$

и $u \in Closure(r)$. Вершина v доминирует над вершиной u под корнем r, если v имеет больший индекс, чем u в массиве $topSortOrder_r$.

Замечание 3. Если дано две особых вершины под одним корнем, то одна из них всегда будет доминировать над другой.

Теорема 1. *Пусть дан ориентированный граф $G(V, E)$, один из его корней $r \in Roots(G)$, $v \in Special_r$, $u \in Closure(r)$. Если v доминирует над u под корнем r, значит u достижима из v.*

Доказательство. По определению особой вершины размер замыкания v равен ее индексу плюс один в $topSortOrder_r$. Рассмотрим все вершины, отличные от v в замыкании $Closure(v)$. Эти вершины в массиве $topSortOrder_r$ имеют индексы меньше, чем вершина v. Отсюда $\forall u \in Closure(v)$ вершина u будет достижима из v. □

Теорема 2. *Пусть дан ориентированный граф $G(V, E)$, один из его корней $r \in Roots(G)$, $v \in Special_r$, $u \in Closure(r)$. Если v доминирует над u под корнем r, значит $Closure(v) \cup Closure(u) = Closure(v)$.*

Доказательство. Согласно предыдущей теореме, если v доминирует над u под корнем r, то вершина u будет достижима из v. Тогда $u \in Closure(v)$, откуда следует $Closure(u) \subseteq Closure(v)$. □

Определение 14. Те вершины графа $G(V, E)$, которые не имеют входящих ребер, называются глобальными корнями.

Рассмотрим две вершины в замыкании глобального корня. Пусть одна из них особая, а другая нет. Чтобы эти две вершины могли образовать базис, особая вершина не должна доминировать над обычной. В противном случае обычная вершина могла бы быть опущена как незначимая. В общем случае, если дан набор вершин, можно убрать из набора все доминируемые вершины и получить корректный базис.

На рисунке 3 приведены примеры наборов из двух вершин, не образующих корректный базис или не удовлетворяющих условию слабой связности.

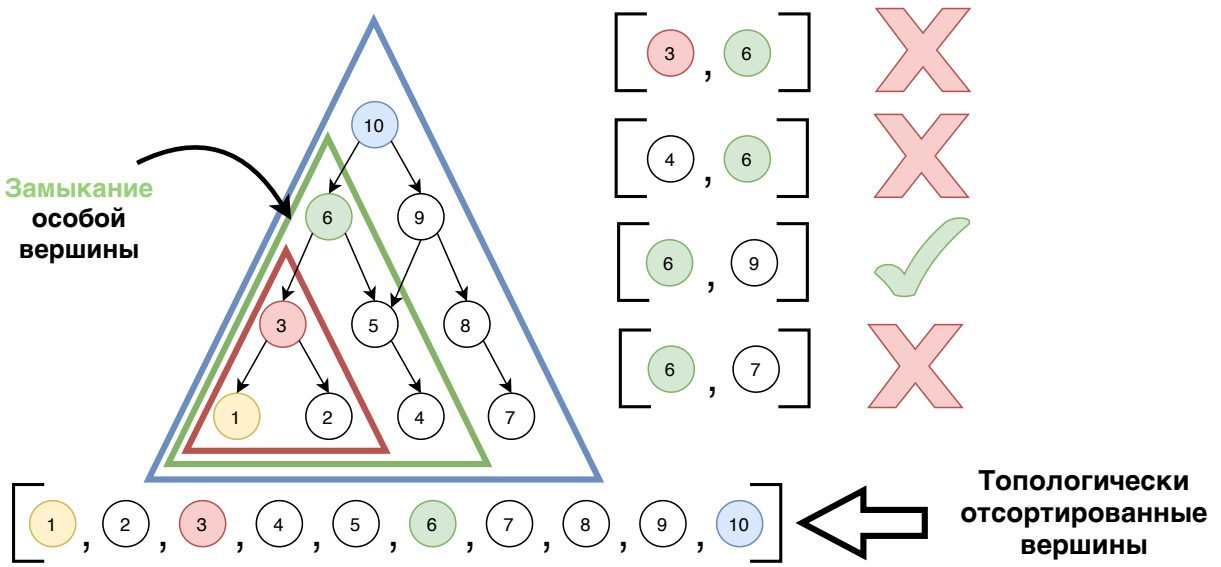


Рис. 3: Граф из 10 вершин с единственным корнем и его топологическая сортировка. Цветом выделены особые вершины. Цветные треугольники обозначают замыкание соответствующих особых вершин. В правой части приведены примеры наборов кандидатов размера 2.

4.4 Алгоритм IsBasis

Алгоритм 5 (IsBasis) проверяет, является ли набор вершин базисом. Этот алгоритм принимает набор вершин и возвращает True, если данный набор образует базис, и False в противном случае. Он используется как составная часть алгоритма TopSortBH, который будет описан позднее.

IsBasis действует следующим образом.

В первую очередь нужно узнать, какие вершины являются особыми. Этот шаг может быть посчитан заранее. Чтобы определить особые вершины, построим топологическую сортировку замыкания для каждого глобального корня. Далее по определению 12 отметим особые вершины.

Вторым шагом, если хотя бы одна из вершин набора доминируема, то данный набор вершин пропускается.

На третьем шаге закончились дешевые методы принятия решения. Проверяем попарную достижимость вершин в наборе. Если найдется хоть одна такая пара, опускаем данный набор вершин.

Если ни одна из предыдущих проверок не ответила False (Пропустить), то алгоритм IsBasis возвращает True. Это значение возвращается в алгоритм TopSort и означает, что текущий набор вершины является корректным базисом черной дыры. Здесь есть небольшая оговорка. Проверки, которые проводит IsBasis никак не учитывают слабую связность графа, индуцированного рассматриваемым набором вершин.

Алгоритм 5: IsBasis

Input : $G(V, E)$ – ориентированный граф без циклов (граф конденсация)
$Cand \subset V$ – множество вершин, потенциальный базис черной дыры

Output: $True$ – если $Cand$ является базисом черной дыры, $False$ в противном случае

```
/* (1) определить особые вершины для каждого корня                    */
```
foreach $r \in Roots(G)$ **do**
  $Special_r = \emptyset$
  build $topSortOrder_r$
  for $i = 0; i < |topSortOrder_r|; i = i + 1$ **do**
    $v = topSortOrder_r[i]$
    $C_v = Closure(v)$
    if $|C_v| == i$ **then**
      $Special_r = Special_r \cup v$
      $maxSpecialIndex_r = max(maxSpecialIndex_r, i)$
    end
  end
end
```
/* (2) проверить, есть ли доминируемые вершины                        */
```
foreach $r \in Roots(G)$ **do**
  $maxSpecialIndex_r = -1$
end
foreach $v \in Cand$ **do**
  foreach $r \in Roots(G)$ **do**
    if $v \in Special_r$ **then**
      $vTopSortIndex_r = arg_v(topSortOrder_r)$
      $maxSpecialIndex_r = max(maxSpecialIndex_r, vTopSortIndex_r)$
    end
  end
end
foreach $v \in Cand$ **do**
  foreach $r \in Roots(G)$ **do**
    if $v \in Closure(r)$ **then**
      $vTopSortIndex_r = arg_v(topSortOrder_r)$
      if $vTopSortIndex_r < maxSpecialIndex_r$ **then**
        return $False$
      end
    end
  end
end
```
/* (3) проверить вершины на взаимную достижимость                     */
```
foreach $(v, u) \in Cand(G) \times Cand(G)$ **do**
  if v *is reachable from* u **then**
    return $False$
  end
end
return $True$

4.5 Алгоритм TopSortBH

Алгоритм 6: TopSortBH

Input : $G(V, E)$ – ориентированный граф
Output: *Blackholes* – множество черных дыр разных размеров
$Blackhole = \emptyset$
Построить граф конденсацию $Cond(V', E')$
for $i = 1$ *to* $|V'|$ **do**
  $cnt_i = 0$
  foreach $B_i \in V'(i)$ **do**
    if $IsBasis(Cond(V', E'), B_i)$ **then**
      if B_i – *слабо связан* **then**
        $Blackholes = Blackholes \cup B_i$
        $cnt_i = cnt_i + 1$
      end
    end
  end
  if $cnt_i == 0$ **then**
    return *Blackholes*
  end
end
return *Blackholes*

Предлагаемый нами алгоритм 6 TopSortBH является комбинацией трех составных частей, которые были описаны ранее.

- Предобработка: конденсация графа;
- Перебор черных дыр при помощи алгоритма 4 BasisBruteForce;
- Сокращение перебора при помощи алгоритма 5 IsBasis на основе информации о топологии графа.

Данный алгоритм действует следующим образом.

Для входного графа строится его конденсация. Далее под графом всегда понимаем именно сконденсированный вариант.

Перебираются наборы вершин размером от 1 до N, и проверяются алгоритмом IsBasis. Если для некоторого набора возвращается $False$, то такой набор сразу же пропускается. Если для некоторого набора возвращается $True$, то для получения черной дыры, необходимо, согласно определению, объединить все замыкания его вершин. Последний шаг заключается в проверке полученного множества на слабую связность. Если данное свойство присутствует, то это будет означать, что найдена черная дыра.

Если не было найдено ни одного корректного базиса размера K, это значит, что не найдется ни одного корректного базиса размером $K + 1$. А значит, в такой ситуации можно остановить перебор.

Отметим основные отличия данного алгоритма от iBlackholeDC:

- Перебираются не наборы вершин, а наборы корней;
- Любой набор, даже тот, что не прошел проверку IsBasis, на самом деле образует черную дыру, если удовлетворяет требованию слабой связности. Для iBlackhole такого условия будет недостаточно.
- Нельзя задать конкретный размер черной дыры для поиска. Для этого потребовалось бы хранить информацию о размерах замыканий вершин и их пересечений;
- Отсутствует связь между итерациями. Для каждого отдельно взятого набора решение принимается независимо;
- Если в графе отсутствуют черные дыры определенных размеров, то это не скажется на работе алгоритма TopSortBH. Он будет перебирать только те черные дыры, которые фактически присутствуют.

4.6 Гибридный алгоритм TopOver

Предложенный выше алгоритм TopSortBH не позволяет ограничить размер рассматриваемых черных дыр. Такая возможность есть у алгоритма iBlackholeDC. Его процесс отбора вершин-кандидатов устроен таким образом, что если провести единственную итерацию при $i = K$, то останется множество вершин, которые могут образовывать черные дыры размером не больше, чем K. Когда такое множество вершин получено, необходимо перебрать наборы вершин и проверить на соответствие определению черной дыры.

Алгоритм 7: TopOver

Input : $G(V, E)$ – ориентированный граф, V – множество вершин, E – множество ребер,
n – максимальное число вершин, которое может содержать черная дыра

Output: *Blackholes* – множество всех черных дыр графа размером от 1 до n

Построить граф конденсацию $GC = Cond(V', E')$
$Blackholes = \emptyset$
$P_n = \{v | d_{out}(v) < n\}$
убрать лишние вершины из P_n и сформировать список кандидатов C_n
убрать лишние вершины из C_n и сформировать список кандидатов F_n
foreach $WCC \in GC(F_n)$ **do**
    `/* Применить алгоритм TopSortBH к` $GC(WCC)$ `*/`
    $Blackholes = Blackholes \cup TopSortBH(GC(WCC))$
end
return *Blackholes*

Воспользуемся отбором кандидатов на основе iBlackholeDC, а в качестве алгоритма перебора используем TopSortBH. Таким образом, получим алгоритм-комбинацию 7 TopSortBH over iBlackholeDC (TopOver). Заметим, что в данном случае конденсация графа производится один раз до начала отбора кандидатов. Так как на вход TopSortBH передается уже сконденсированный граф, то повторная конденсация не требуется. Размеры исходных КСС не учитываются при определении размера черной дыры. При возникновении такой необходимости алгоритм легко модифицировать.

4.7 Аспекты параллелизма в рассмотренных алгоритмах

Все представленные алгоритмы обладают запасом параллелизма. Рассмотрим подробнее алгоритмы 3 iBlackholeDC и 6 TopSortBH. Первый из них обладает параллелизмом на четырех уровнях:

1. Параллелизм итераций. Здесь, как уже упоминалось, есть оговорка. Строго говоря, каждая итерация обращается к результатам предыдущей. Таким образом, в алгоритме удается избежать повторного принятия решений о тех вершинах, которые ранее уже были добавлены во множество кандидатов. Эту проверку можно проигнорировать и начать вычисления с любой итерации. Тогда множество ранее утвержденных кандидатов окажется пусто, и для каждой вершины придется принимать решение. Результат будет верен в любом случае. Значит, каждую итерацию (или последовательность итераций) можно запустить на отдельном вычислителе. Эффективнее распределять итерации блоками, это сократит накладные расходы.

2. Параллелизм рассмотрения кандидатов. Решение о добавлении очередной вершины в множество кандидатов можно принимать независимо от остальных внутри итерации. В этом случае важно, чтобы множество кандидатов с предыдущей итерации было полностью сформировано.

3. Параллелизм компонент слабой связности. Каждую компоненту слабой связности графа, индуцированного набором кандидатов, можно рассматривать независимо.

4. Параллелизм перебора. Работу по перебору подмножеств кандидатов можно распределить между узлами. Для этого достаточно пронумеровать все наборы.

Теперь о параллелизме алгоритма TopSortBH.

1. Параллелизм компонент слабой связности. Исходный граф можно разделить на копоненты слабой связности и каждую обрабатывать независимо.

2. Параллелизм перебора. Здесь можно выделить два подуровня параллелизма.

2a. Параллелизм размеров множеств-кандидатов. Каждый размер можно рассматривать независимо. Между размерами нет связи по данным (в отличие от итераций iBlackholeDC), что дает значительный простор для параллельной обработки. В рамках одного размера удобно применять эвристики для сокращения перебора на основе топологии.

2б. Параллелизм перебора. Для каждого выбранного размера подмножества существует большое количество комбинаций выбора. Для каждой такой комбинации решение принимается независимо.

4.8 Детали реализации

В этом разделе рассмотрим несколько особенностей реализации алгоритма TopSortBH.

Все описанные алгоритмы реализованы на языке C++. Для написания параллельного кода над общей памятью использовалась технология OpenMP.

4.8.1 О работе с графом

Граф хранится в виде списков смежности. То есть для каждой вершины хранится список номеров вершин, в направлении которых есть ориентированные ребра. Все графы считаются ориентированными. В неориентированном графе для каждого ребра существует обратное ему по направлению. Такой подход позволяет использовать одинаковые реализации алгоритмов как для ориентированных, так и для неориентированных графов.

Кроме исходного графа для работы алгоритмов часто требуются граф с инвертированными ребрами и неориентированный граф. Так, например, в iBlackholeDC возникает задача построения замыкания вершины по обратным ребрам. Проверку слабой связности удобно реализовать обходом вершин в неориентированном графе. Поэтому алгоритмы на ранних стадиях работы строят инвертированный и неориентированный графы.

4.8.2 Проверка слабой связности

Для проверки слабой связности используется обход в ширину на неориентированном графе.

Листинг 1: Проверка слабой связности множества корней

```
bool CheckConnectivity(const TGraph& graphUndir,
                       const set<size_t>& bh)
{
    TUsed used;
    used.assign(graphUndir.size(), 1);
    for (size_t v : bh) {
        used[v] = 0;
    }
    size_t v = *bh.begin();
```

```
    TClosure closure = GetClosure(graphUndir, v, used);
    return closure.size() == bh.size();
}
```

Создается массив used, в котором 0 помечаются вершины из проверяемого множества. От произвольной вершины запускается процедура обхода замыкания вершины. Обход посещает только те вершины, которые отмечены 0, а уже обойденные помечает 1. Если размер замыкания совпадает с размером проверяемого множества, значит, присутствует слабая связность. Код приведен в листинге 1.

4.8.3 Проверка попарной достижимости

Проверка попарной достижимости корней встроена в функцию GetBlackhole. Ее код представлен в листинге 2. GetBlackhole для заданного графа *graph*, топологической сортировки *tsOrder* и набора позиций в нем строит множество вершин черной дыры. Если среди корней находится хотя бы одна доминируемая вершина, то будет возвращено пустое множество.

Массив *used* содержит в себе счетчики количества посещений для каждой вершины графа. При обходе очередного замыкания (см. Листинг 3), все вершины, которые были посещены или хотя бы рассмотрены к посещению, увеличивают свои счетчики на 1. Если одна из вершин может быть достигнута различными путями от одного корня, то ее счетчик может быть увеличен только единожды. Если корень может быть достигнут при обходе замыкания другого корня, то его счетчик после обхода всех корней станет больше или равен 2. Если это так, то такая черная дыра могла быть обнаружена при просмотре кандидатов меньшего размера, а значит должна быть пропущена.

Важно заметить, что такой алгоритм проверки будет работать только на сконденсированном графе.

Листинг 2: Построение черной дыры по набору корней

```
set<size_t> GetBlackHole(const TGraph& graph,
                         const vector<size_t>& tsOrder,
                         const vector<size_t>& pos)
{
    set<size_t> blackhole;
    TUsed used;
    used.assign(graph.size(), 0);
    for (size_t p : pos) {
        size_t v = tsOrder[p];
        TClosure closure = GetClosure(graph, v, used);
        for (size_t x : closure) {
```

```
            blackhole.insert(x);
        }
    }
    for (size_t p : pos) {
        size_t x = tsOrder[p];
        if (used[x] >= 2) {
            return {};
        }
    }
    return blackhole;
}
```

Листинг 3: Обход замыкания данной вершины

```
TClosure GetClosure(const TGraph& g,
                    size_t v,
                    TUsed& used)
{
    return ClosureBFS(g, used, v);
}

TClosure ClosureBFS(const TGraph& g,
                    TUsed& used,
                    size_t s)
{
    TUsed localUsed(used.size(), 0);
    queue<size_t> q;
    TClosure closure;
    if (!used[s]) {
        q.push(s);
        closure.insert(s);
    }
    used[s] += 1;
    localUsed[s] += 1;
    while(!q.empty()) {
        size_t v = q.front();
        q.pop();
        for (size_t to : g[v]) {
            if (!localUsed[to]) {
                if (!used[to]) {
```

```
                    q.push(to);
                    closure.insert(to);
                }
                used[to] += 1;
                localUsed[to] += 1;
            }
        }
    }
    return closure;
}
```

4.8.4 Сокращение перебора

Как было описано выше, наличие доминируемых вершин в наборе корней, позволяет пропустить этот набор, так как его можно свести к другому, меньшего размера. Чтобы понять, что в наборе есть хотя бы одна доминируемая вершина, не проверяя достижимости, необходимо убедиться, что не найдется особой вершины, которая была бы топологически выше другой вершины в наборе.

Функция ShouldSkip (Листинг 4) осуществляет такую проверку. Массив *pos*, как и ранее, содержит в себе упорядоченные по возрастанию номера позиций корней черной дыры в массиве tsOrder. Массив tsOrder содержит порядок топологической сортировки графа. Правило, описанное выше, эквивалетно тому, что особая вершина может быть самой низкой в топологии набора и при этом единственной. Таким образом, как только обнаружена особая вершина на любой позиции, начиная со второй, можно делать вывод, что набор не подходит.

Листинг 4: Проверка множества на наличие доминируемых вершин

```
bool ShouldSkip(const vector<size_t>& tsOrder,
                const vector<char>& special,
                const vector<size_t>& pos)
{
    for (size_t i = 1; i < pos.size(); i++) {
        size_t p = pos[i];
        size_t v = tsOrder[p];
        if (special[v]) {
            return true;
        }
    }
    return false;
```

```
}
```

4.9 Модификация SkipPrecise

Листинг 5: Пропуск сразу нескольких наборов кандидатов без ложно-отрицательных результатов

```
void ForceSkipPrecise(const vector<size_t>& tsOrder,
                      const vector<char>& special,
                      vector<size_t>& pos)
{
    size_t skip = 0;
    for (size_t i = 1; i < pos.size(); i++) {
        size_t p = pos[i];
        size_t v = tsOrder[p];
        if (special[v]) {
            skip = i;
            break;
        }
    }
    for (size_t i = skip + 1; i < pos.size(); i++) {
         pos[i] = tsOrder.size() - pos.size() + i;
    }
}
```

Пусть дан массив *pos*, который содержит в себе особую вершину на некоторой позиции $skip \neq 0$. Как было сказано в предыдущем пункте, такой массив *pos* будет иметь в себе избыточные элементы, а потому не может являться корректным базисом. Следовательно, его необходимо пропустить. Если рассмотреть значения на отрезке индексов $[0, skip]$, и рассмотреть массив $[..., skipVal, ...]$, то станет ясно, что любой массив pos' с таким префиксом не будет формировать корректный базис. Таким образом, можно сразу же переходить к набору $[..., skipVal+1, skipVal+2, ..., skipVal+pos.size()-skip]$, тем самым пропуская проверку для всех наборов с рассмотренным префиксом. Отметим, что для удобства реализации, на самом деле, мы перейдем не к следующему корректному набору кандидатов, а к последнему набору из некорректной последовательности. Реализация данного правила представлена в листинге 5.

4.10 Модификация SkipFast

Листинг 6: Пропуск большого числа кандидатов. Допускаются ложно-отрицательные результаты.

```
void ForceSkipFast(const vector<size_t>& tsOrder,
                   const vector<char>& special,
                   vector<size_t>& pos)
{
    size_t skip = 0;
    for (size_t i = 1; i < pos.size(); i++) {
        size_t p = pos[i];
        size_t v = tsOrder[p];
        if (special[v]) {
            skip = i;
            break;
        }
    }
    size_t skipPos = pos[skip];
    pos[0] = skipPos - 1;
    for (size_t i = 1; i < pos.size(); i++) {
        pos[i] = tsOrder.size() - pos.size() + i;
    }
}
```

В силу комбинаторной природы задачи, рассмотрение всех черных дыр некоторого графа может потребовать больших временных ресурсов. В ситуации, когда необходимо быстро обнаружить как можно больше любых черных дыр, можно прибегнуть к более агрессивной политике сокращения перебора. Для такого случая можно применить представленную в листинге 6 эвристику.

Пусть дан массив pos, который содержит в себе особую вершину на некоторой позиции $skip \neq 0$. Попробуем сформировать массив pos' такой, что единственная особая вершина, если такая найдется, будет находиться по индексу равному 0. Мы знаем, что особая вершина находится в позиции $skip$. Перейдем от массива $[..., skipVal, ...]$ к массиву $[skipVal, skipVal + 1, skipVal + 2, ..., skipVal + pos.size() - skip]$. Такой набор по построению будет иметь особую вершину на первой позиции, но не обязательно будет являться корректным базисом.

Данный метод приводит к появлению ложно-отрицательных результатов, но значительно увеличивает число обнаруженных за ограниченное время черных дыр, что отражено в экспериментальной части. В то же время, использование данной эвристики сокращает число просмотренных, но пропущенных кандидатов.

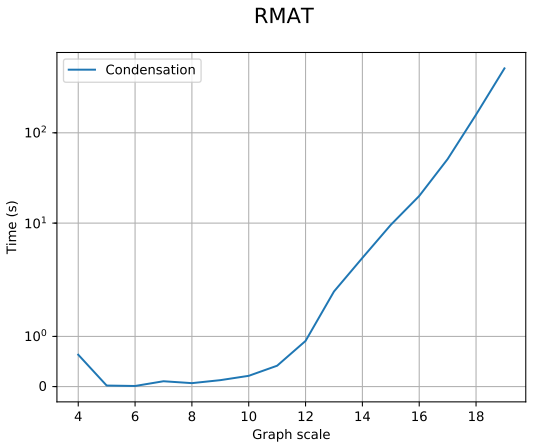
5 Результаты экспериментальных исследований

Для сравнения разработанных алгоритмов был проведен ряд экспериментов. Сравнение производилось на трех типах графов: RMAT [3], SSCA2 [2], и Random Uniform [4]. Все запуски были произведены на системе Polus, состоящей из 5 узлов [1]. Характеристики узла выглядят следующим образом:

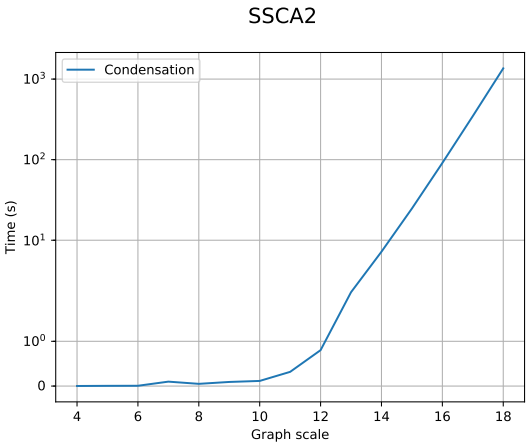
- 2 десятиядерных процессора IBM POWER8 (каждое ядро имеет 8 потоков), всего 160 потоков
- Общая оперативная память 256 Гбайт (в узле 5 оперативная память 1024 Гбайт) с ECC контролем
- 2 x 1 ТБ 2.5” 7K RPM SATA HDD
- 2 x NVIDIA Tesla P100 GPU, 16Gb, NVLink
- 1 порт 100 ГБ/сек
- Производительность кластера (Tflop/s): 55,84 (пиковая), 40,39 (Linpack)

Замечание 4. Здесь и далее: Timeout – время, по прошествии которого программе посылается сигнал завершения.

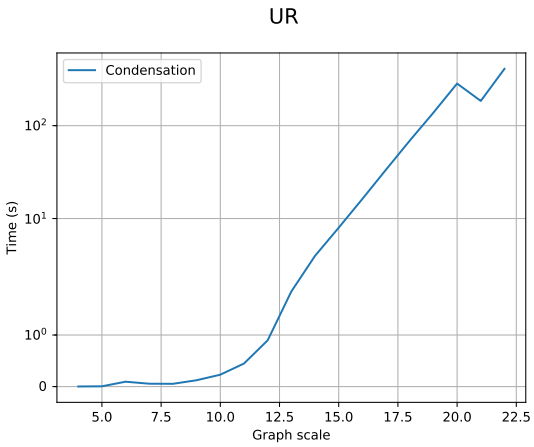
5.1 Конденсация графа



(a) RMAT



(b) SSCA2



(c) UR

Рис. 4: Зависимость времени конденсации от масштаба графа. Timeout: 30 минут.

В первую очередь проведено исследование производительности алгоритма конденсации. На рисунке 4 показано время его работы для различных видов и масштабов графов. Видно, что с увеличением размера графа также значительно возрастает время необходимое на обработку. Это время может достигать десятков минут и более, но данное время мало по сравнению с перебором черных дыр на исходном, не сжатом, графе.

Таблица 1: Влияние конденсации на размер графа.

	До		После	
Граф	Вершин	Ребер	Вершин	Ребер
RMAT	131072	4181810	4837	4092
RMAT	262144	7892410	12547	10360
UR	131072	4193755	1	0
UR	262144	8388074	1	0
SSCA2	131072	1358656	131072	1358656
SSCA2	262144	2724762	262144	2724762

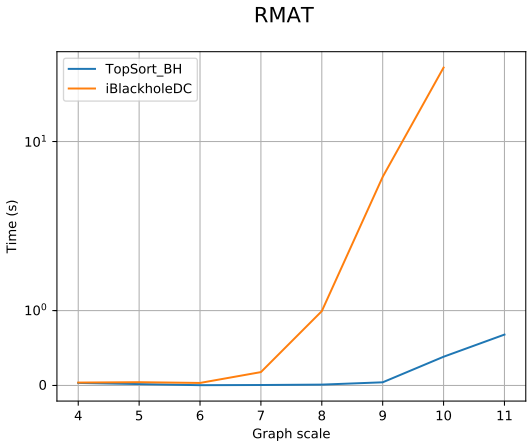
Конденсация дает различные результаты, в зависимости от структуры сжимаемого графа.

Так, Uniform Random графы, как правило состоящие из одной большой КСС, будут сжаты в единственную вершину. SSCA2 графы почти не имеют компонент сильной связности, поэтому их размер после сжатия в большинстве случаев не изменится. RMAT графы уменьшатся в размерах за счет наличия крупных КСС. Сравнение приведено в таблице 1

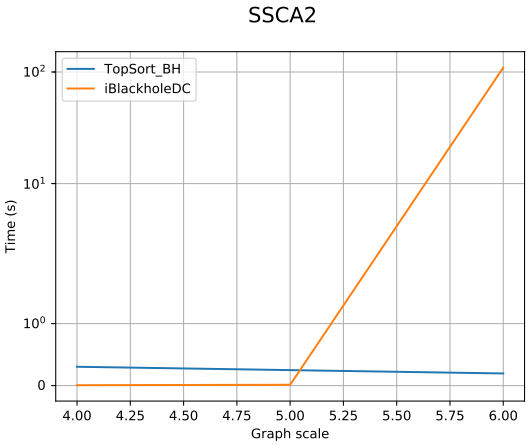
5.2 Сравнение iBlackholeDC и TopSortBH

Было проведено сравнение iBlackholeDC и TopSortBH на графах небольшого размера. В работе [9] алгоритм iBlackholeDC уже испытывался на похожих по размеру графах. Теперь же было проведено его сравнение с представленным в данной работе TopSortBH.

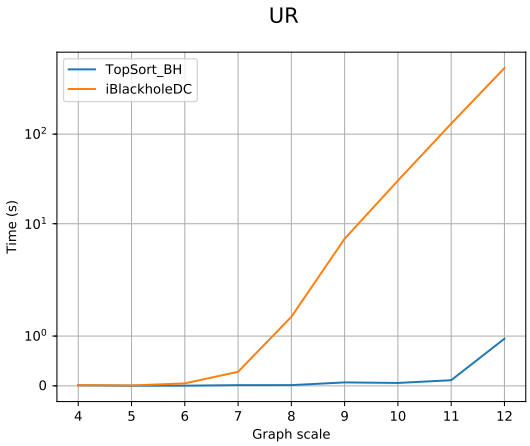
iBlackholeDC запускался в один поток. Алгоритм был реализован по описанию в оригинальной статье 2010 года и больше не изменялся. TopSortBH запускался в 8 потоков. Здесь проводится не равное сравнение, поскольку есть цель подчеркнуть, что представленный в данной работе алгоритм хорошо подходит для параллельной работы, а также показать общее преимущество достигнутое по результатам работы. Оба алгоритма получали на вход граф без предобработки. Таким образом, конденсация графа, которая необходима алгоритму TopSortBH для корректной работы, учитывалась в общем времени его работы.



(a) RMAT



(b) SSCA2



(c) UR

Рис. 5: Алгоритм iBlackholeDC, запущенный в 1 потоке, и алгоритм TopSortBH, запущенный с использованием 8 потоков. Timeout: 30 минут.

На рисунке 5a показано, что iBlackholeDC не уложился в ограничение по времени для графов масштабом больше 10. TopSortBH успел отработать на всех графах, кроме масштаба 12, и время его работы не превышало секунды.

На рисунке 5b видим, что iBlackholeDC показывал преимущество на графах совсем маленьких размеров, но начал проигрывать в последствии. Оба алгоритма переставали укладываться в отведенное время после масштаба 6 SSCA2 графа. Эти графы хуже всего поддаются любым эвристикам и сжатиям, в них ярче всего проявляется комбинаторная природа задачи.

В случае графа UR Рис. 5c оба алгоритма уложились в отведенное время для всех представленных размеров, однако TopSortBH был быстрее во всех случаях. Здесь конденсация графа имеет решающее значение для времени работы алгоритма.

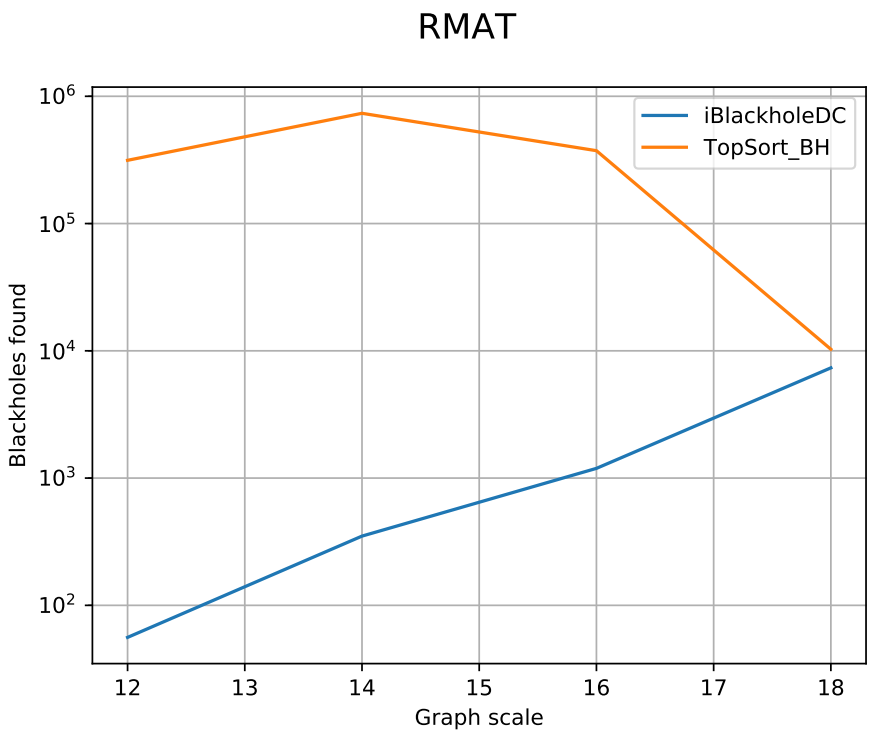
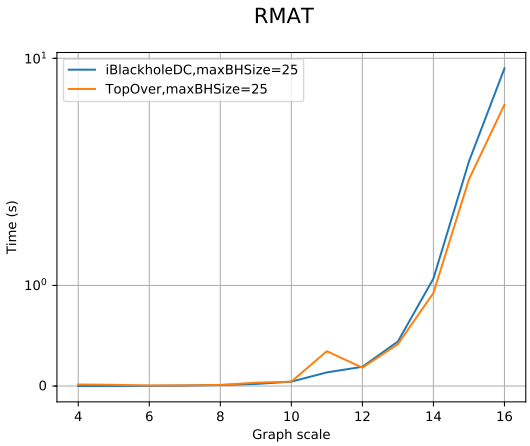


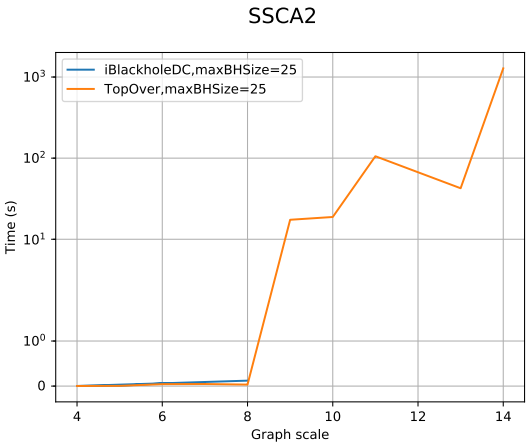
Рис. 6: Алгоритм iBlackholeDC, запущенный в 1 потоке, и алгоритм TopSortBH, запущенный с использованием 8 потоков на графах большого размера. По оси Y отложено найденное количество черных дыр. Timeout: 3 часа.

Также было проведено сравнение производительности для RMAT графов больших размеров. Сложность подобного сравнения заключается в том, что такие графы обрабатываются очень долго и конечный результат может зависеть от структуры графа и заданного ограничения по времени. На рисунке 6 показано количество найденных черных дыр за 3 часа. В правой части графики начинают сходиться. Предположительно, это особенность конкретно взятого графа в сочетании с выбранным временем останова. Последнее может особенно сильно влиять на показатели алгоритмов относительно друг друга.

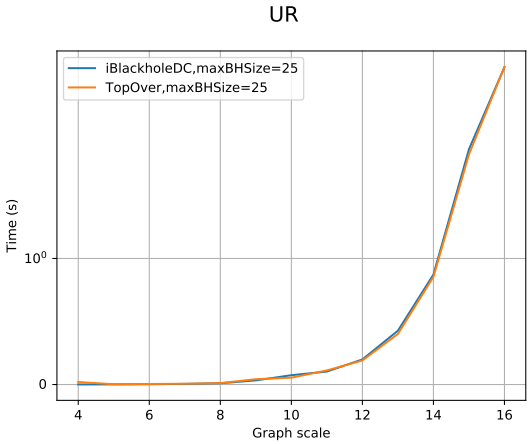
5.3 Сравнение iBlackholeDC и TopOver



(a) RMAT



(b) SSCA2

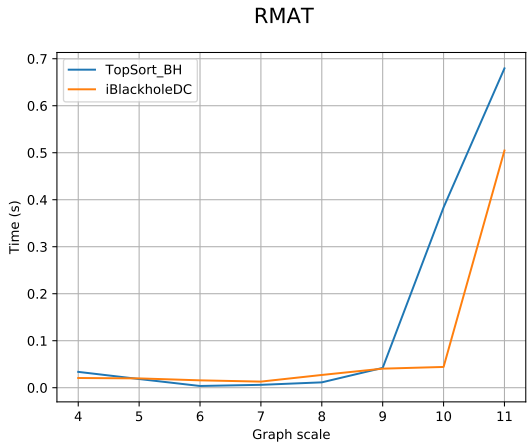


(c) UR

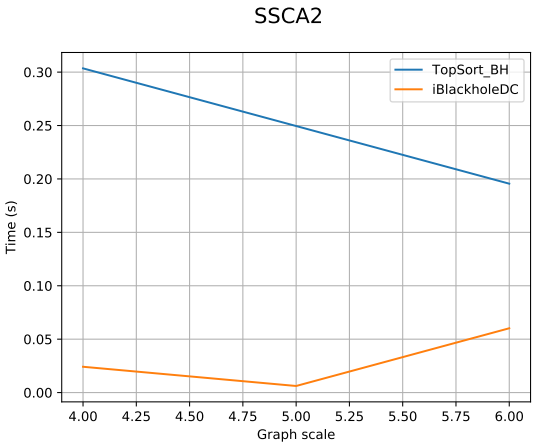
Рис. 7: Алгоритмы iBlackholeDC и TopOver. Оба запущены с использованием 1 потока. Осуществляется поиск черных дыр размером до 25 вершин. Timeout: 30 минут.

Переходим к сравнению алгоритма TopOver и оригинального iBlackholeDC. Эти алгоритмы сравниваем в режиме поиска черных дыр ограниченного размера. В качестве такого размера было выбрано число 25. Запуски производились на графах масштабом до 16. На графах RMAT рис. 7a алгоритмы показывают сравнимую производительность, однако при увеличении размера графа TopOver алгоритм демонстрирует ярко выраженное преимущество. В случае UR рис. 7c графов разница между алгоритмами отсутствует. На графах SSCA2 рис. 7b iBlackhole не успевает отработать для масштабов больше 8, а TopOver укладывается в полчаса на графах масштабом до 14 включительно.

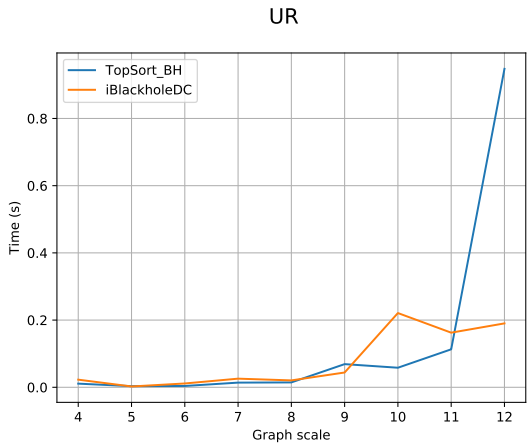
5.4 Алгоритм iBlackholeDC на сконденсированных графах



(a) RMAT



(b) SSCA2

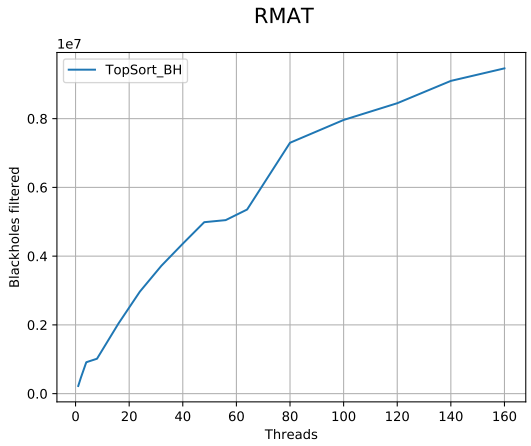


(c) UR

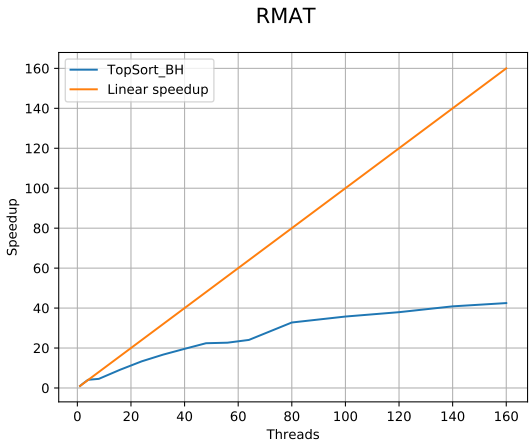
Рис. 8: Алгоритм iBlackholeDC, осуществляющий конденсацию графа перед началом своей работы, запущен с использованием 8 потоков. Алгоритм TopSortBH запущен с использованием 8 потоков, Timeout: 30 минут.

Логичным шагом было добавить стадию конденсации графа в iBlackholeDC и сравнить его многопоточный вариант с TopSortBH. Как видно на рисунке 8, при таких условиях iBlackholeDC выигрывает по времени работы у TopSortBH.

5.5 Исследование параллелизма алгоритма TopSortBH



(a) Отфильтровано кандидатов

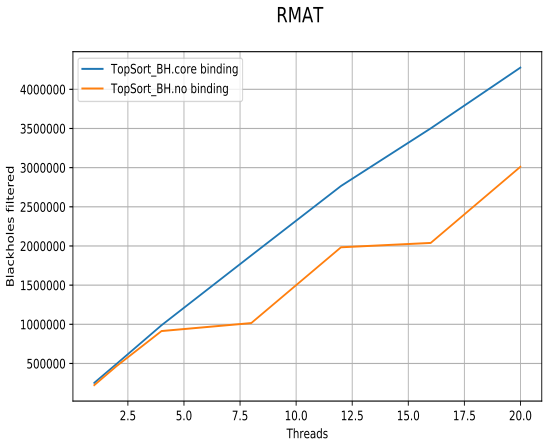


(b) Ускорение

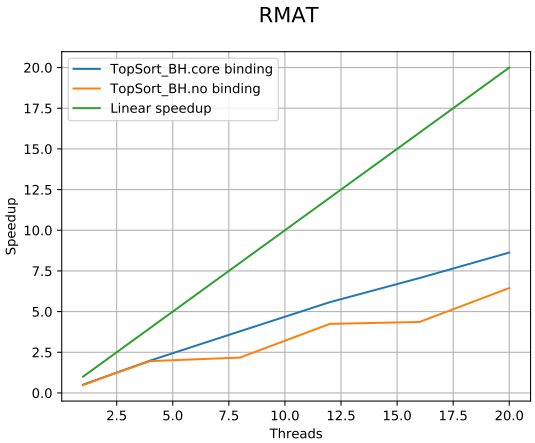
Рис. 9: TopSortBH запущенный с использованием от 1 до 160 потоков. По оси X отложено число потоков, по оси Y – количество просмотренных кандидатов. Timeout: 30 минут.

Замечание 5. Здесь и далее: привязка потоков не использовалась при запуске приложения, если явно не указано иное.

Как уже было сказано, TopSortBH разрабатывался для использования в параллельном режиме. Поэтому был проведен эксперимент с использованием большого количества потоков. Все запуски производились на графе RMAT масштаба 18. Важно отметить, что во всех конфигурациях алгоритм успел обнаружить одинаковое количество черных дыр и завершился через полчаса работы. Поэтому было удобно сравнивать количество рассмотренных кандидатов. Система Polus позволяет запускать до 160 потоков на одном узле, что позволило провести исследование поведения алгоритма с использованием общей памяти. На рисунке 9 показано, что при использовании 80 потоков алгоритм успевает отфильтровать 7.3 млн кандидатов, а при использовании 160 потоков 9.4 млн кандидатов. Т.е. использование дополнительных 80 потоков увеличивает производительность приблизительно на 28%. Также на графике наблюдается заметный излом именно при 80 задействованных потоках.



(a) Отфильтровано кандидатов

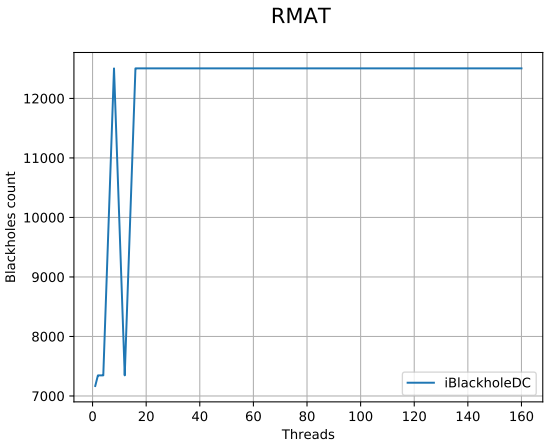


(b) Ускорение

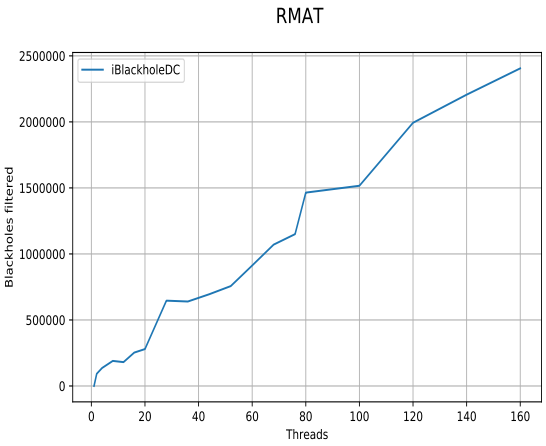
Рис. 10: Алгоритм TopSortBH запущенный с использованием от 1 до 20 потоков. С использованием привязки потоков и без. По оси X отложено количество потоков. По оси Y отложено число отфильтрованных кандидатов. Timeout: 30 минут.

Также алгоритм TopSortBH был протестирован в режиме привязки потоков к ядрам. Polus располагает 20 ядрами на каждый узел, поэтому в режиме привязки было использовано не более 20 потоков. По рисунку 10 видно, что производительность с использованием привязки для 20 потоков превышает результат обычного запуска на треть. Более того 20 потоков в режиме привязки, показывают такую же производительность, как 40 потоков без привязки.

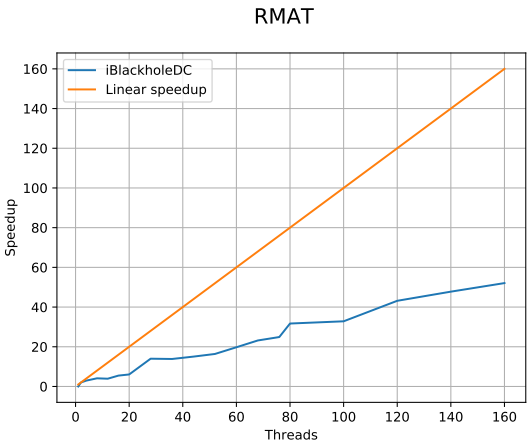
5.6 Исследование параллелизма алгоритма iBlackholeDC



(a) Обнаружено черных дыр



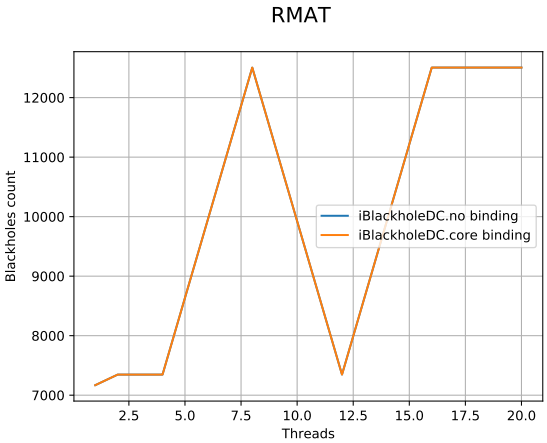
(b) Отфильтровано кандидатов



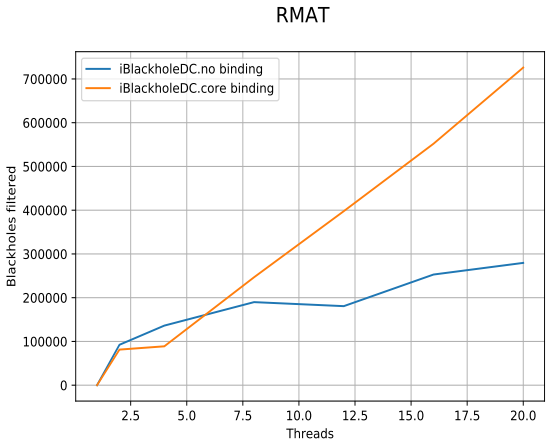
(c) Ускорение

Рис. 11: Алгоритм iBlackholeDC запущенный для количества потоков от 1 до 160. Число потоков отложено по оси X. Timeout: 30 минут.

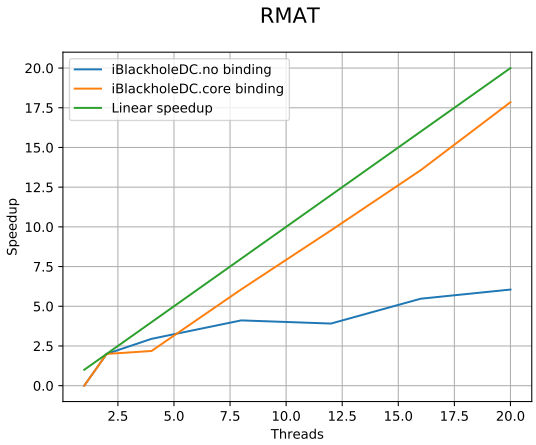
Алгоритм iBlackholeDC был запущен с использованием до 160 потоков. Для всех запусков был установлен порог по времени 30 минут. Во время каждого запуска была собрана информация о количестве найденных черных дыр (Рис. 11a) и количестве кандидатов, которые не прошли проверку 11b. Заметим, что количество черных дыр, найденных за отведенное время, может изменяться в зависимости от числа потоков. Такое поведение связано с тем, что диапазон итераций, обрабатываемых каждым потоком, зависит от номера потока и общего числа потоков. В то же время можно отметить, что алгоритм показывает неплохую масштабируемость, вопреки тому, что для эффективной работы следующей итерации требуется информация о предыдущей.



(a) Обнаружено черных дыр



(b) Отфильтровано кандидатов

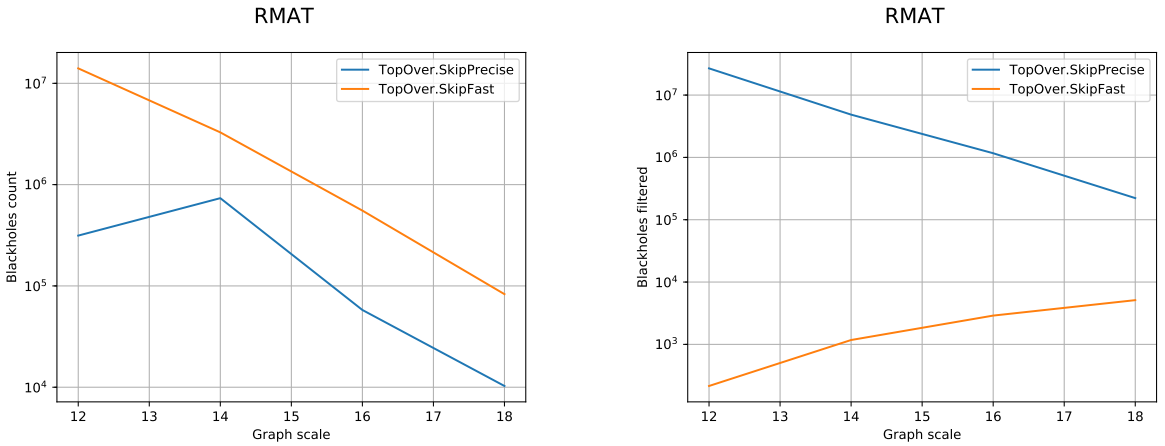


(c) Ускорение

Рис. 12: Алгоритм iBlackholeDC запущенный для количества потоков от 1 до 20 с использованием привязки потоков к ядру. Число потоков отложено по оси X. Timeout: 30 минут.

Также были проведены запуски iBlackholeDC с использованием привязки потока к ядру процессора. Узел системы Polus имеет 2 процессора по 10 ядер, таким образом привязать можно было до 20 потоков. На рисунке 12 приведено сравнение алгоритма с использованием привязки потоков и без. Интересен тот факт, что число обнаруженных черных дыр не меняется по сравнению с запусками без привязки потоков. При этом значительно возрастает количество просмотренных кандидатов.

5.7 Модификация SkipFast



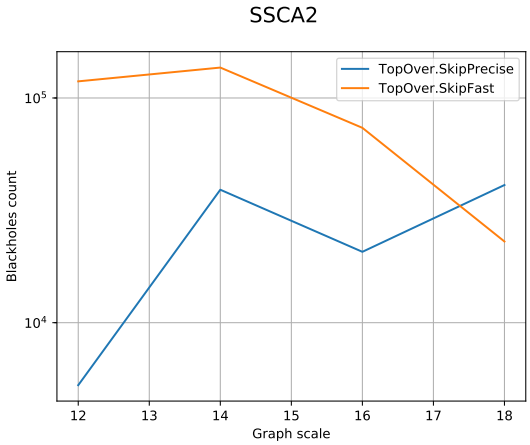
(a) Обнаружено черных дыр

(b) Отфильтровано кандидатов

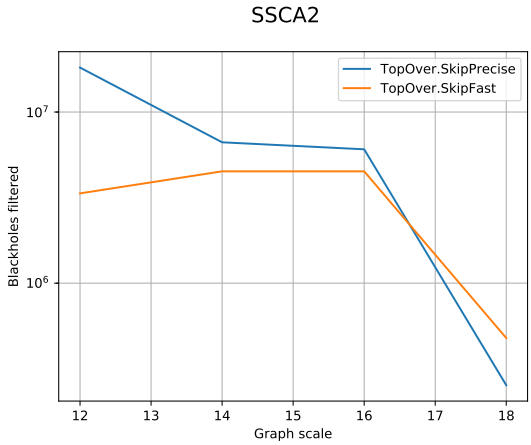
Рис. 13: Алгоритм TopOver запущенный в 1 потоке с использованием модификации SkipFast. По оси X отложен масштаб графа. Timeout: 30 минут.

Описанная в пункте 4.10 модификация позволяет быстрее пропускать большие группы наборов-кандидатов, но допускает появление ложно-отрицательных результатов. В ситуациях, когда время сильно ограничено, такой подход позволяет быстрее находить черные дыры. Данный подход был проверен экспериментально. Алгоритм TopOver был запущен в единственном потоке на графах типа RMAT масштабом от 12 до 18 включительно. Измерялось число обнаруженных черных дыр и число отклоненных кандидатов. При этом, применение правила SkipFast считалось как рассмотрение единственного кандидата, поскольку только при его рассмотрении было затрачено время работы алгоритма.

На рисунке 13 показаны результаты данного эксперимента. Видим, что за отведенные полчаса алгоритм, использующий модификацию SkipFast, устойчиво находил на порядок больше черных дыр, чем SkipPrecise версия. Вместе с тем, количество рассмотренных и отклоненных кандидатов для SkipFast версии значительно уменьшилось. Такой результат можно интерпретировать следующим образом: с большой вероятностью, оценка которой остается за рамками данной работы, применение модификации SkipFast приводит к переходу к следующей группе ранее не рассмотренных черных дыр.



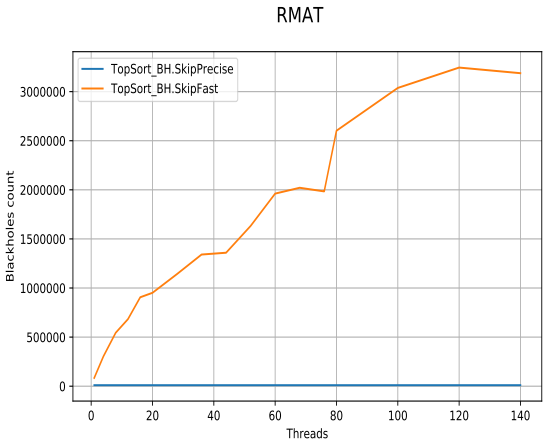
(a) Обнаружено черных дыр



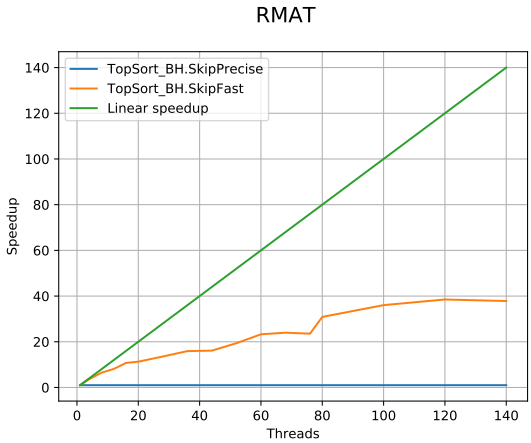
(b) Отфильтровано кандидатов

Рис. 14: Алгоритм TopOver запущенный в 1 потоке с использованием подификации SkipFast. По оси X отложен масштаб графа. Timeout: 30 минут.

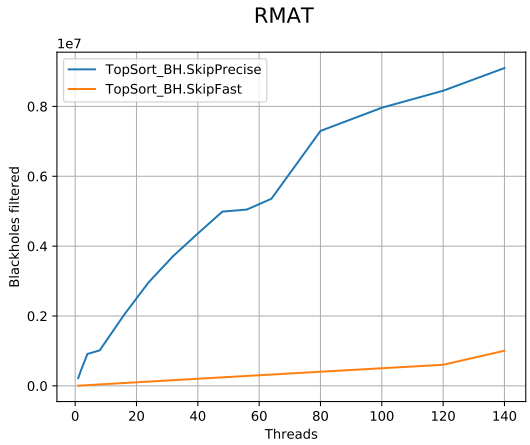
Также было проведено сравнение модификаций перебора SkipFast и SkipPrecise на графах SSCA2. Здесь (см. Рисунок 14) SkipFast хорошо проявляет себя в работе с графами масштабом до 17 включительно, но проигрывает для масштаба 18.



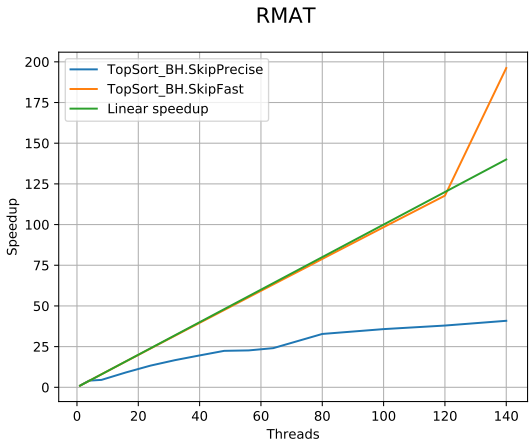
(a) Обнаружено черных дыр



(b) Ускорение



(c) Отфильтровано кандидатов



(d) Ускорение

Рис. 15: Алгоритм TopSort запущенный с использованием модификации SkipFast. Число потоков от 1 до 140. Граф RMAT масштаба 18. По оси X отложено количество потоков. Timeout: 30 минут.

Также было проверено влияние модификации SkipFast на результаты работы алгоритма TopSortBH. Алгоритм TopSortBH был запущен на графе RMAT масштаба 18. Запуски производились с использованием от 1 до 140 потоков. Каждый запуск был ограничен 30 минутами. Отслеживались число найденных черных дыр и число отфильтрованных кандидатов. Для сравнения, на графиках представлены результаты работы алгоритма TopSortBH в режиме SkipPrecise, приведенные на рисунке 9.

Как видно на рисунке 15, TopSortBH с модификацией SkipFast демонстрирует многократное преимущество при любом числе потоков. Причем, использование большего числа потоков пропорционально укрепляет это преимущество. Также здесь наблюдается характерное снижение числа отклоненных кандидатов.

6 Заключение

В данной работе была рассмотрена задача поиска паттерна черная дыра в ориентированных графах без весов.

В рамках решения данной задачи в работе рассмотрен предложенный ранее алгоритм iBlackhole и его модификация iBlackholeDC. Для него впервые разработана параллельная реализация на общей памяти с использованием технологии OpenMP. В работе реализации алгоритмов тестируются на вычислительном узле на основе двух процессоров IBM Power8 (до 160 потоков) в составе системы Polus, установленной на факультете ВМК МГУ. При использовании 160 потоков iBlackholeDC работает в 50 раз быстрее, чем в режиме одного потока. С использованием привязки потоков демонстрирует почти линейную масштабируемость.

Для решения задачи поиска черных дыр разработан и реализован новый алгоритм TopSortBH. Он состоит из трех главных частей: конденсация графа, ранее не использовавшаяся при решении данной задачи, подход к организации перебора, позволяющий избегать наборов вершин, которые заведомо не являются черными дырами, подход к сокращению такого перебора, с целью ускорения обработки графов большого размера. В процессе разработки алгоритма доказано две теоремы об особенностях строения черных дыр.

В дизайне алгоритма TopSortBH заложен потенциал для его использования в массивно-параллельных системах. При использовании 160 OpenMP-потоков, просмотр кандидатов становится до 40 раз быстрее. При использовании привязки потоков к ядрам процессора, 20 потоков демонстрируют 8 кратный прирост производительности.

В диссертации разработана и экспериментально проверена модификация SkipFast для алгоритма TopSortBH. Она позволила быстро обнаруживать значительное количество черных дыр в условиях ограниченного времени за счет более эффективного сокращения числа рассматриваемых кандидатов. При использовании 140 потоков за 30 минут работы данная модификация алгоритма успевает найти более 3 миллионов черных дыр на графе RMAT с 2^{18} вершин и отфильтровать дополнительно 100 тысяч кандидатов. iBlackholeDC при тех же условиях обнаружит лишь 12.5 тысяч черных дыр и отклонит 2.2 миллиона кандидатов.

Рассмотрено и проверено на практике комбинированное использование техник TopSortBH и iBlackholeDC. Такой подход назван TopOver, он демонстрирует преимущество в задаче поиска черных ограниченного размера. В частности, в задаче поиска черных дыр размером до 25 на графах SSCA2 TopOver успевает завершить работу для всех масштабов до 14 включительно (2^{14} вершин), в то время как iBlackholeDC успевает отработать для масштабов 8 и меньше.

Таким образом, в работе получены следующие основные результаты:

- Доказаны теоремы о строении черных дыр на графах;

- Разработан алгоритм TopSortBH, основанный на сокращении размера графа (конденсации), сокращении перебора на основе доказанных теорем, а также модификация алгоритма SkipFast;
- Разработаны многопоточные реализации алгоритма TopSortBH и существовавшего ранее алгоритма iBlackholeDC с использованием технологии OpenMP;
- Разработан гибридный подход TopOver для поиска черных дыр ограниченного размера;
- Проведено исследование производительности:
  - Рассмотрены графы до 2^{18} вершин (по сравнению с 1500 вершин в предыдущих работах);
  - Предложенные новые алгоритмы за меньшее время обрабатывают граф или успевают рассмотреть большее число кандидатов за то же время по сравнению с существующим ранее алгоритмом iBlackholeDC.

Список литературы

[1] *Описание вычислительного комплекса IBM Polus.* Url: `http://hpc.cmc.msu.ru/polus` Дата обращения 17.05.2020.

[2] Bader, D. A., and Madduri, K. Design and implementation of the hpcs graph analysis benchmark on symmetric multiprocessors. In *High Performance Computing – HiPC 2005* (Berlin, Heidelberg, 2005), D. A. Bader, M. Parashar, V. Sridhar, and V. K. Prasanna, Eds., Springer Berlin Heidelberg, pp. 465–476.

[3] Chakrabarti, D., Zhan, Y., and Faloutsos, C. R-MAT: A recursive model for graph mining. In *Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004* (2004), M. W. Berry, U. Dayal, C. Kamath, and D. B. Skillicorn, Eds., SIAM, pp. 442–446.

[4] Erdős, and Renyi. On random graphs. *Publicationes Mathematicae 6* (1959).

[5] Fortunato, S. Community detection in graphs. *Physics reports 486*, 3-5 (2010), 75–174.

[6] Hong, L., Zheng, Y., Yung, D., Shang, J., and Zou, L. Detecting urban black holes based on human mobility data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2015), ACM, p. 35.

[7] Li, Z., and Xiong, H. Mining blackhole and volcano patterns for fraud detection. *Encyclopedia of Social Network Analysis and Mining* (2014), 904–915.

[8] Li, Z., Xiong, H., and Liu, Y. Mining blackhole and volcano patterns in directed graphs: a general approach. *Data Mining and Knowledge Discovery 25*, 3 (2012), 577–602.

[9] Li, Z., Xiong, H., Liu, Y., and Zhou, A. Detecting blackhole and volcano patterns in directed networks. In *2010 IEEE International Conference on Data Mining* (2010), IEEE, pp. 294–303.

[10] Semenov, A., Mazeev, A., Doropheev, D., et al. Survey of common design approaches in aml software development. In *GraphHPC 2017 Conference (GraphHPC). CEUR Workshop Proceedings* (2017), vol. 1981, pp. 1–9.

[11] Sharir, M. A strong-connectivity algorithm and its applications in data flow analysis. *Computers & Mathematics with Applications 7*, 1 (1981), 67–72.

[12] Watts, D. J. Networks, dynamics, and the small-world phenomenon. *American Journal of sociology 105*, 2 (1999), 493–527.