Распределенный поиск компонент сильной связности в адаптивной модели

И. В. Молдованов[1]

В работе исследуется сложностная оценка алгоритма поиска компонент сильной связности в Adaptive Massively Parallel Computations(AMPC) модели. Данная модель, в отличие от иных более ограниченных распределенных формализаций, позволяет в рамках одного шага алгоритма строить дерево запросов в распределенную память. Получен вероятностный алгоритм, реализующий поиск компонент сильной связности за полилогарифмическое или сублинейное время, в зависимости от объема доступной локальной памяти. Объем требуемой локальной памяти является сублинейной величиной по отношению к числу вершин в графе.

Ключевые слова: распределенные алгоритмы, вероятностные алгоритмы, компоненты сильной связности.

1. Введение

В современном мире вычислительные мощности растут значительно быстрее чем доступные ресурсы памяти, что в существенной мере ограничивает возможность обрабатывать большие объемы данных. В данном ключе особый интерес получают распределенные системы, позволяющие более эффективным способом использовать значительное число вычислительных единиц (ядер, процессоров и т.д.). Чтобы иметь возможность разрабатывать более эффективные алгоритмы для подобных систем, был предложен ряд математических формализаций: CONGEST, CONGESTED-CLIQUE, MPC (Massively Parallel Computations) и, наконец, AMPC (Adaptive Massively Parallel Computations). Последняя модель была предложена сравнительно недавно в статье [3], и обладает наибольшими выразительными возможностями, позволяя эмулировать все предыдущие формализации. В рамках данной работы будет представлен алгоритм поиска компонент сильной связности направленного графа, выраженный в AMPC модели, и представлена оценка его сложности.

[1] *Молдованов Илья Владимирович* — студент каф. математической теории интеллектуальных систем мех.-мат. ф-та МГУ, e-mail: ilmoldovanov@mail.ru.

Moldovanov Ilya Vladimirovich — student, Lomonosov Moscow State University, Faculty of Mechanics and Mathematics, Chair of Mathematical Theory of Intellectual Systems.

1.1. Распределенные вычисления

Ранее уже была обозначена причина бурного развития интереса к распределенным вычислениям. Остановимся теперь более подробно на описанных выше формализациях. Первой значимой вехой в развитии данных систем была CONGEST модель. Ее первичное предназначение - описание графовых алгоритмов, что, в свою очередь, привело к следующей структуре: каждая вершина исходного графа представляет собой отдельный вычислительный узел, обладающий бесконечной памятью. Коммуникация между узлами осуществляется через ребра исходного графа, и ее размер ограничен $O(\log n)$ битами, где n - число вершин в графе. В качестве меры сложности в данной модели принимается число раундов подобных коммуникаций. Несмотря на кажущееся неудобство подобного подхода с точки зрения практики, данная формализация позволяет достаточно хорошо формулировать сложностные оценки для алгоритмов в терминах Pregel описания[1], которое оказало значительное влияние на сферу проектирования распределенных алгоритмов на графах. В свою очередь, более общая CONGESTED-CLIQUE модель сохраняет все ограничения CONGEST формализации, однако позволяет проводить коммуникация между всеми парами вершин, независимо от наличия соответствующих ребер в исходном графе.

Следующим этапом развития распределенных алгоритмов можно считать появление MPC модели, впервые описанной в [2]. Данная модель подразумевает использование вычислительных узлов с распределенной памятью уже в более каноничном, по сравнению с предыдущими формализациями, ключе и обычно характеризуется тремя параметрами: P - количество доступных машин, N - размер входных данных и S - объем памяти на каждой машине, а так же максимально число отправляемых сообщений (более точно: каждая машина может хранить в памяти и отправить за один раунд не более $O(S \log n)$ бит). Наибольший интерес, очевидно, представляют значения параметра $S << N$, таким образом в большинстве работ рассматривается строго сублинейный случай, то есть $S = N^{\epsilon}$, для некоторого $0 < \epsilon < 1$. В качестве основной меры сложности для оценки MPC алгоритмов принимается количество раундов, в рамках которых происходит обмен сообщениями между машинами и произвольные локальные вычисления.

Описанные в данном разделе модели, несмотря на значимые структурные отличия, обладают ключевым общим свойством - сложность алгоритмов в данных формализациях определяется объемом коммуникаций между вычислительными машинами, а не количеством арифметических операций или обращений к памяти. Данная особенность в вышеизложенном описании формализаций отражена в том факте, что, вообще говоря,

локальные вычисления в рамках раунда допускаются в произвольном объеме. Подобный подход к моделированию распределенных вычислений объясняется спецификой соответствующих программных реализаций - при наличии большого количества вычислительных узлов(каждый из которых может обладать несколькими потоками) именно коммуникация между данными узлами становится основным узким местом большинства задач. Тем не менее чтобы сохранить связь с приложениями, в литературе, посвященной распределенным алгоритмам в той или иной формализации, хорошим тоном считается отмечать сложность локальных вычислений, если она может быть высоко-полиномиальной или даже экспоненциальной.

Замечание В данной работе сложность локальных вычислений не превосходит сублинейной по отношению к числу вершин в графе.

Таким образом, сложность распределенных алгоритмов не является понятием тождественным сложности алгоритмов в ее классическом понимании, однако при экстремальных значениях параметров модели (например, если система состоит только из одной машины, которой разрешено провести только одну операцию в раунд) данные понятия можно считать достаточно близкими (особенно это касается MPC и AMPC моделей). Дополнительно подобную аналогию можно провести так же и с параллельными парадигмами вычислений, например PRAM, так в работе [2] приводятся симуляции некоторых PRAM алгоритмов в MPC формализации.

Таким образом, мы косвенным путем получаем верхнюю оценку сложности распределенных алгоритмов, по крайней мере для MPC и AMPC моделей, которые являются предметом фокуса данной работы.

1.2. AMPC модель

В данной работе мы будем рассматриват AMPC модель, отличительные особенности которой представлены ниже. Впервые описанная в [3], данная формализация обобщает MPC подход, позволяя без увеличения сложности в терминах раундов эмулировать любые реализуемые в MPC алгоритмы.

Данная модель характеризуется следующими параметрами: N - размер входных данных, P - количество машин, которые будут обрабатывать данные, S - объем памяти на каждой машине выражаемый, в количестве идентификаторов, размера $\log N$ бит, которые могут храниться в памяти машины. Обозначим также через T общий объем памяти всех машин, то есть $T = SP$.

В данной работе мы рассмотрим наиболее интересный случай $S = O(N^\epsilon)$, $0 < \epsilon < 1$. Подобное соотношение параметров характеризует ситуацию,

когда все данные не могут быть целиком размещены в памяти одной машины, то есть начинают оказывать влияние естественные для распределенных алгоритмов ограничения, связанные с необходимостью синхронизации результатов вычислений, произведенных каждой машиной, а так же отсутствием доступа ко всем данным. Более того, в рамках данной работы будет представлен алгоритм, позволяющий решать исходную задачу в ситуации, когда $S = O(n^\epsilon)$, $0 < \epsilon < 1$, где n - число вершин исследуемого графа.

Также в рамках AMPC модели имеется набор распределенных неограниченных по памяти хранилищ данных, которые мы обозначим через $D_0, D_1, D_2, ...$ Для описания удобно предположить, что все хранилища предоставляют семантику доступа ключ-значение, то есть, что они хранят набор пар ключ-значение и по запросу, содержащему заданный ключ, возвращают соответствующий идентификатор. При этом важно отметить, что размеры ключа и индентификатора составляют $O(\log N)$ бит. В свою очередь входные данные хранятся в D_0 и используют набор ключей известный всем машинам (например, последовательные целые числа). Дополнительно при наличии $k > 1$ пар ключ-значение, имеющих один и тот же ключ x, отдельные значения могут быть получены, по запросам $(x, 1), ..., (x, k)$. Стоит отметить, что индексы от 1 до k присваиваются произвольно. Наконец, запрос ключа, которого нет в распределенном хранилище, приводит к пустому ответу.

Алгоритм в данной модели состоит из раундов. В i-ом раунде каждая машина может считать данные из D_{i-1} записать в D_i. Таким образом, в течение раунда каждая машина может выполнить до $O(S)$ операций чтения (далее называемых запросами), произвести $O(S)$ записей, а так же выполнить произвольные вычисления. Каждый запрос и запись представляют собой обращение к хранилищу для чтения или записи одной пары ключ-значение. Ключевым свойством модели по сравнению с MPC формализацией является то, что запросы, которые машина делает в течение раунда, могут зависеть от результатов предыдущих запросов, сделанных в том же раунде.

1.3. Пример алгоритма в AMPC модели

Приведем теперь наглядный пример функционирования описанной выше модели.

Рассмотрим граф G, имеющий n вершин и m ребер. Задача состоит в том, чтобы вычислить степень каждой вершины в данном графе.

Зафиксируем параметры модели: размер задачи $N = n + m$, число машин $P = n$ и объем локальной памяти $S = n^\epsilon$, для некоторого $0 < \epsilon < 1$. В качестве начального состояния памяти D_0 выберем следующее пред-

ставление графа: граф G записан в виде набора ребер, то есть пар вида $(Source, Destination)$. В данном случае ключ $Source$, и соответствующая запись $Destination$ представляют собой целые числа, не превышающие n, а значит удовлетворяют ограничению на длину записи $O(\log N)$
Отметим, что представление в виде списка смежности было бы некорректно в данной модели, так как размер записи для каждого ключа составлял бы $O(n \log N)$ бит.
Рассмотрим теперь возможный алгоритм решения поставленной задачи. Пронумеруем каждую машину от 1 до n и установим соответствие между машинами и вершинами с совпадающими номерами. Для подсчета степеней вершин каждой машине необходимо считать все ребра с соответствующим значением ключа, увеличивая внутренний счетчик. Таких ребер может быть не более чем $n - 1$, а значит за $O(n^{1-\epsilon})$ раундов каждая машина сможет прочесть все ассоциированные с данной вершиной ребра. В качестве ответа каждая машина запишет пару $(Source, Degree)$ в конечное состояние общей памяти.

1.4. Мотивация применения AMPC модели

Дополняя представленный выше пример, опишем более подробно применимость AMPC модели для решения практических задач.
В мире распределенных вычислений значительную роль играет понятие "MapReduce". Данный фреймворк позволяет реализовать параллельные вычисления над большими объемами данных с использованием набора вычислительных узлов, формирующих кластер. Подобные кластера могут содержать значительное количество компьютеров, связанных беспроводным соединением. Тем не менее память каждой отдельной машины ограничена размером оперативной памяти или диска, что позволяет единовременно хранить лишь фрагмент общих данных. Таким образом, описанная ранее MPC модель была впервые представлена как формализация подобного подхода к распределенным вычислениям, позволяющая оценить эффективность различных алгоритмических решений.
В свою очередь, AMPC модель была представлена как развитие основополагающих идей предыдущей формализации. Данная модель была дополнена распределенной базой данных, позволяющей в некотором роде устранить недостаток, связанный с ограничением локальной памяти на машинах, за счет относительно быстрых запросов в распределенное хранилище.

1.5. Постановка задачи

Подводя итог представленному выше описанию AMPC модели, обозначим описанные в работах [3] и [4] результаты.

Авторам данных исследований удалось в AMPC модели получить эффективные реализации большого количества фундаментальных задач, присущих ненаправленным графам. В числе описанных ими алгоритмов присутствуют Связность, Связность леса, Нахождение минимального покрывающего дерева, Нахождение максимального независимого множества, Поиск связных компонент. Важно отметить, что сложность полученных реализаций превосходит(в плане эффективности) ранее известную сложность в MPC формализации.
Таким образом, в качестве потенциального направления для дальнейшего исследования можно выделить рассмотрение аналогичных задач в направленных графах. В данном ключе естественным кандидатом является фундаментальная задача поиска компонент сильной связности, которой уделяется значительное внимание в классической литературе. Еще более интересным данное направление исследования делает тот факт, что на данный момент не удалось получить эффективную MPC реализацию подобного алгоритма(по сложности превосходящую параллельные аналоги). Таким образом, целью данной работы была поставлена следующая задача: используя превосходящие выразительные возможности AMPC модели, получить распределенный алгоритм поиска компонент сильной связности, имеющий, как максимум, сублинейную сложность.

1.6. Компоненты сильной связности

Сначала напомним ряд определений.
Ориентированный граф $G(V, E)$ называется **сильно связным**, если для любых двух вершин $s, t \in V$ существуют направленные пути из s в t и из t в s.
Компонентами сильной связности ориентированного графа называются его максимальные по включению сильно связные подграфы.
Простейший алгоритм для решения данной задачи состоит в следующем: для каждой пары вершин в графе необходимо определить взаимную достижимость и соединить пары, удовлетворяющие данному свойству, ненаправленным ребром. Далее алгоритм поиска компонент связности на построенном ненаправленном графе позволяет найти искомый ответ.
Среди более совершенных подходов можно выделить линейные(в сложностных терминах классической последовательной парадигмы) алгоритмы Тарьяна [5] и Косарайю [6].
Тем не менее, несмотря на свою вычислительную эффективность, данные алгоритмы плохо подходят для параллельной реализации, так как являются последовательными по своей сути. Таким образом, первый параллельный алгоритм поиска компонент сильной связности был предложен в [7]. В качестве основного узкого места данного подхода можно вы-

делить линейную по числу вершин глубину рекурсии, в худшем случае. Тем не менее данная проблема была успешно решена в [8]. Авторам удалось представить алгоритм, позволяющий реализовать поиск компонент сильной связности в CRCW PRAM модели со сложностью $O(R\log^2 n)$, где R - сложность алгоритма нахождения всех вершин в графе, достижимых из заданной.

1.7. Структура работы

В заключение вводной части приведем план дальнейшего изложения результатов, представленных в данной работе.
Определение сложности полученного в работе алгоритма, а так же доказательство его корректности во многом базируются на сформулированных и доказанных в разделе **2 Выборка вершин** технических утверждениях.
Основная содержательная часть работы изложена в разделе **3 Достижимость из одной вершины**. В рамках данного раздела приведен распределенный алгоритм, позволяющий определить множество достижимости для фиксированной вершины в графе. В подразделе **3.2** представлена версия алгоритма, позволяющая решить поставленную задачу только для графа с фиксированными показателями входящих и исходящих степеней, однако в подразделах **3.3** и **3.4** описаны способы обобщить предыдущий результат на произвольный граф.
Наконец, заключительный результат представлен в виде описанного в разделе **4 Компоненты сильной связности** подхода к поиску компонент сильной связности в AMPC модели.

2. Выборка вершин

В данной секции мы рассмотрим вероятностный подход к выборке вершин в графе, который позволяет, практически не увеличивая общую сложность алгоритма, сконструировать подмножество вершин, обладающих необходимыми нам полезными свойствами. Данная техника эффективно применяется в ряде вероятностных графовых алгоритмов, при этом особенный интерес в контексте поставленной задачи для нас представляют работы [9] и [10].
Приведем сперва два ключевых результата, которые легли в основу всех утверждений данной секции:

Теорема 1. ***Неравенство Чернова*** *Пусть* $X = \sum_{i=1}^{k} X_i$, *где все* X_i *независимые Бернуллиевские случайные величины, принимающие значение*

1 с вероятностью p_i, и значение 0 с вероятностью $1-p_i$. Через μ обозначим $E[X] = \sum_{i=1}^{k} E[X_i]$, тогда:

1) $P(X \geq (1+\delta)\mu) \leq e^{-\frac{\delta^2\mu}{2+\delta}}$ *для всех* $\delta > 0$

2) $P(X \leq (1-\delta)\mu) \leq e^{-\frac{\delta^2\mu}{2}}$ *для всех* $0 < \delta < 1$

Следствие 1. *Пусть X, X_i и μ соответствуют условиям теоремы, тогда:*

$$P(|X-\mu| \geq \delta\mu) \leq 2e^{-\frac{\delta^2\mu}{3}} \text{ для всех } 0 < \delta < 1$$

Более подробно данные утверждения изложены в [11].

Лемма 1. *Рассмотрим V - множество натуральных чисел от 1 до n включительно и константы $\gamma \leq n, \delta \in (0,1)$. Пусть так же имеется L - проиндексированный набор из n подмножеств V, таких что размер каждого не меньше чем γ, то есть $\forall L_v \in L$ имеем $L_v \subseteq V, |L_v| \geq \gamma$. Далее рассмотрим набор независимых одинаково распределенных случайных величин S_v, проиндексированных $1...n$ и имеющих распределение Бернулли с параметрами*

$$S_v = \begin{cases} 1, & p \\ 0, & 1-p \end{cases}$$

Тогда если $p \geq \min(\frac{12\log(2n/\delta)}{\gamma}, 1)$, то с вероятностью как минимум $1-\delta$

1) $\forall L_v \in L\ \exists u \in L_v :\ S_u = 1$.

2) $\sum_{v \in V} S_v \in [\frac{1}{2}pn, \frac{3}{2}pn]$

Доказательство. 1) Для фиксированного индекса v рассмотрим $Pr(\sum_{u \in L_v} E[S_u] - \sum_{u \in L_v} S_u \geq \frac{1}{2}\sum_{u \in L_v} E[S(u)])$ Поскольку S_v - независимые, одинаково распределенные Бернуллиевские случайные величины, то воспользовавшись оценкой Чернова получим следующее неравенство:

$$Pr(|\sum_{u \in L_v} E[S_u] - \sum_{u \in L_v} S_u| \geq \frac{1}{2}\sum_{u \in L_v} E[S_u]) \leq \tag{1}$$

$$2\exp\left(-\frac{\sum_{u \in L_v} E[S_u]}{12}\right) = 2\exp\left(-\frac{p|L_v|}{12}\right) \leq \tag{2}$$

$$2\exp\left(-\frac{p\gamma}{12}\right) \leq 2\exp\left(-\log 2n/\delta\right) \leq \frac{\delta}{n} \tag{3}$$

Поскольку $\frac{1}{2}\sum\limits_{u\in L_v} E[S(u)] \geq 1$, то получаем, что с вероятностью как минимум $1-\frac{\delta}{n}$, $\sum\limits_{u\in L_v} S_u \geq 1$.

Наконец, объединив вероятности для всех $L_v \in L$, пункт (1) выполняется с вероятностью как минимум $1-\delta$

2) Рассмотрим выражение $Pr(|\sum\limits_{v\in V} E[S_v] - \sum\limits_{v\in V} S_v| \geq \frac{1}{2}\sum\limits_{v\in V} E[S(v)])$ Поскольку S_v - независимые, одинаково распределенные Бернуллиевские случайные величины, то воспользовавшись оценкой Чернова получим следующее неравенство:

$$Pr(|\sum_{v\in V} E[S_v] - \sum_{v\in V} S_v| \geq \frac{1}{2}\sum_{v\in V} E[S_v]) \leq \tag{4}$$

$$2\exp\left(-\frac{\sum\limits_{v\in V} E[S_v]}{12}\right) = 2\exp\left(-\frac{p|V|}{12}\right) \leq \tag{5}$$

$$2\exp\left(-\frac{p\gamma}{12}\right) \leq 2\exp\left(-\log 2n/\delta\right) \leq \frac{\delta}{n} \leq \delta \tag{6}$$

□

Рассмотрим направленный граф $G = G(V, E)$. Для каждой вершины $v \in V$ обозначим $D_{out}(v)$ - множество ее соседей по исходящим ребрам, а $D_{in}(v)$ - по входящим, в дополнение введем $Deg_{out}(v) = |D_{out}(v)|$ и $Deg_{in}(v) = |D_{in}(v)|$.
В свою очередь исходящей степенью графа $Deg_{out}(G)$ назовем $\min\limits_{v\in V} Deg_{out}(v)$. Аналогично определим $Deg_{in}(G)$.

Лемма 2. *Рассмотрим направленный граф $G = G(V, E), |V| = n$, и параметры $d \in [\log n, n-1]$, $\delta \in (0, 1)$. Если $Deg_{out}(G) \geq d$, то с вероятностью как минимум $1-\delta$ $\exists SV \subset V$, такое что $\forall v \in V\, \exists u \in D_{out}(v)\cup\{v\}$: $u \in SV$ и $|SV| \in O(\frac{n\log n}{d})$*

Доказательство. Для доказательства рассмотрим **Утверждение 1**. Множество натуральных чисел V представляет собой номера вершин в графе, а в качестве семейства подмножеств L выберем номера соседей, составляющих исходящую окрестность каждой вершины. Затем, в качестве p_d возьмем число $\frac{c\log n}{d}$, где c - наименьшее такое целое число, что $\frac{c\log n}{d} \geq \frac{12\log(2n/\delta)}{d}$. Наконец во множество SV добавим только такие вершины, индексы которых соответствуют Бернуллиевским случайным величинам S_v, принявшим значение "1"(вероятность этого события оставляет p_d). Тогда:

- По пункту 1 **Утверждения 1** получим, что каждое множество из L с вероятностью как минимум $1-\delta$ содержит индекс, который соответствует Бернуллиевской случайно величине, принявшей значение "1". Таким образом, каждая исходящая окрестность в графе с вероятностью как минимум $1-\delta$ содержит вершину из SV.
- По пункту 2 **Утверждения 1** $|SV| \in O(p_d d)$ с вероятностью как минимум $1-\delta$, а так как $p_d = \frac{c \log n}{d} \in O(\frac{logn}{d})$, то $|SV| \in O(n\frac{\log n}{d})$ с вероятностью как минимум $1-\delta$

□

Теперь нам необходимо доказать более жесткое свойство. В данном случае мы хотим показать, что для некоторого d мы можем таким образом выбрать подмножество вершин, что если из заданной вершины v в любую другую вершину в графе существует хотя бы один направленный путь, превышающий по длине d, то так же из v в данную вершину с большой вероятностью существует и такой путь, что любой его подотрезок, длина которого превышает $2d$, содержит хотя бы одну вершину из случайно выбранного подмножества. На основании вышеизложенных рассуждений приведем следующее утверждение. Стоит отметить, что фактически, в плане доказательства, оно мало отличается от описанного ранее, так как обозначенное увеличение количества множеств повлияет лишь на константу в выражении для вероятности.

Лемма 3. *Пусть имеется V - набор индексов от $1...n$ и константы $\gamma \leq n, \delta \in (0,1)$. Пусть так же имеется L - проиндексированный набор из n^2 подмножеств V, таких что размер каждого не меньше чем γ, то есть $\forall L_{v,i} \in L$, $1 \leq v \leq n$, $1 \leq i \leq n$ имеем $L_{v,i} \subseteq V, |L_{v,i}| \geq \gamma$. Далее рассмотрим набор независимых одинаково распределенных случайных величин S_v проиндексированных $1...n$ и имеющих распределение Бернулли с параметрами*

$$S_v = \begin{cases} 1, & p \\ 0, & 1-p \end{cases}$$

Тогда если $p \geq \min(\frac{12 \log(2n^2/\delta)}{\gamma}, 1)$, то с вероятностью как минимум $1-\delta$

1) $\forall L_v \in L \; \exists u \in L_v : \; S_u = 1$.

2) $\sum\limits_{v \in V} S_v \in O(pn)$

Доказательство. 1) Для фиксированных индексов v, i рассмотрим $Pr(\sum\limits_{u\in L_{v,i}} E[S_u] - \sum\limits_{u\in L_{v,i}} S_u \geq \frac{1}{2}\sum\limits_{u\in L_{v,i}} E[S(u)])$ Поскольку S_v - независимые, одинаково распределенные Бернуллиевские случайные величины, то воспользовавшись оценкой Чернова получим следующее неравенство:

$$Pr(|\sum_{u\in L_{v,i}} E[S_u] - \sum_{u\in L_{v,i}} S_u| \geq \frac{1}{2}\sum_{u\in L_{v,i}} E[S_u]) \leq \tag{7}$$

$$2\exp\left(-\frac{\sum\limits_{u\in L_{v,i}} E[S_u]}{12}\right) = 2\exp\left(-\frac{p|L_{v,i}|}{12}\right) \leq \tag{8}$$

$$2\exp\left(-\frac{p\gamma}{12}\right) \leq 2\exp\left(-\log 2n^2/\delta\right) \leq \frac{\delta}{n^2} \tag{9}$$

Поскольку $\frac{1}{2}\sum\limits_{u\in L_{v,i}} E[S(u)]) \geq 1$, то получаем, что с вероятностью как минимум $1 - \frac{\delta}{n^2}$, $\sum\limits_{u\in L_{v,i}} S_u \geq 1$.

Наконец, объединив вероятности для всех $L_{v,i} \in L$, пункт (1) выполняется с вероятностью как минимум $1 - \delta$

2) Рассмотрим выражение $Pr(|\sum\limits_{v\in V} E[S_v] - \sum\limits_{v\in V} S_v| \geq \frac{1}{2}\sum\limits_{v\in V} E[S(v)])$ Поскольку S_v - независимые, одинаково распределенные Бернуллиевские случайные величины, то воспользовавшись оценкой Чернова получим следующее неравенство:

$$Pr(|\sum_{v\in V} E[S_v] - \sum_{v\in V} S_v| \geq \frac{1}{2}\sum_{v\in V} E[S_v]) \leq \tag{10}$$

$$2\exp\left(-\frac{\sum\limits_{v\in V} E[S_v]}{12}\right) = 2\exp\left(-\frac{p|L_v|}{12}\right) \leq \tag{11}$$

$$2\exp\left(-\frac{p\gamma}{12}\right) \leq 2\exp\left(-\log 2n^2/\delta\right) \leq \frac{\delta}{n^2} \leq \delta \tag{12}$$

□

Лемма 4. *Рассмотрим направленный граф $G = G(V, E), |V| = n$ и вершину $v \in V$, а так же параметры $d \in [\log(n), n], \delta \in (0, 1)$. Тогда $\exists SV \subset V$, такое, что если в графе существует ацикличный направленный путь $P(v, u) : |P| \geq d$, то с вероятностью как минимум*

$1-\delta$ *существует такой путь* $P'(v,u)$, $|P'| \geq d$, *что любой подучасток данного пути длины как минимум* $2d$ *содержит вершину из* SV *и* $|SV| \in O(\frac{n \log n}{d})$.

Доказательство. Для доказательства рассмотрим **Утверждение 3**. В качестве семейства подмножеств L выберем последовательные подотрезки длины d на путях из v во все остальные вершины. Затем, в качестве p_d возьмем число $\frac{c \log n}{d}$, где c - наименьшее такое целое число, что $\frac{c \log n}{d} \geq 12 \log(2n^2/\delta)$. Тогда:

- Как было сказано выше, множество L представляет собой последовательные подотрезки длины d на путях из v во все остальные вершины. Если же пути соответствующей длины из вершины v в некоторую другую вершину нет, то без ограничения общности заполним данные $n-1$ множество произвольными вершинами.
  Итак, $|L| \leq n^2$, а значит, применив пункт 1 **Утверждения 3**, получим, что для некоторого пути из v в любую другую вершину, его последовательные подотрезки длины d содержат как минимум одну вершину из SV. Таким образом, расстояние между двумя вершинами из SV на данном пути не превышает $2d$.

- По пункту 2 **Утверждения 3** $|SV| \in O(p_d n)$,а так как $p = \frac{c \log n}{d} \in O(\frac{\log n}{d})$, то $|SV| \in O(\frac{n \log n}{d})$

□

3. Достижимость из одной вершины

В данной секции мы опишем вероятностный алгоритм в AMPC модели, позволяющий для графа $G = G(V,E)$ найти подмножество вершин, достижимое из фиксированной вершины $v \in V$.

3.1. Обзор структуры алгоритма

Сперва мы идейно представим ход алгоритма поиска всех вершин, достижимых из фиксированной вершины v, а так же кратко опишем идеи, лежащие в его основе.
Будем считать далее, что каждая вершина в графе имеет фиксированные входящую и исходящую степень, равные $\lceil n^{\frac{\epsilon}{2}} \rceil$(в последующих секциях будет так же приведен обобщенный алгоритм, тем не менее сохраняющий все основные идеи и принципы описываемого подхода).
Опираясь на материал, изложенный в предыдущей секции, за $O(1)$

раундов в AMPC модели мы можем построить такое подмножество $SV(n^{1-\epsilon}\log n)$, что если между двумя вершинами в графе есть путь, по длине превышающий $2\lceil n^{1-\epsilon}\log n\rceil$, то расстояние на этом пути между последней вершиной из $SV(n^{1-\epsilon}\log n)$ и конечной вершиной с большой вероятностью не превышает $2\lceil n^{1-\epsilon}\log n\rceil$. Таким образом, задача разобьется на два этапа:

- для каждой вершины в графе найти все вершины из $SV(n^{1-\epsilon}\log n)$, находящиеся на расстоянии не превышающем $2\lceil n^{1-\epsilon}\log n\rceil$, из которых она достижима
- для каждой вершины из $SV(n^{1-\epsilon}\log n)$ сказать, достижима ли она из исходной вершины v

В таком случае, по транзитивности мы сможем для каждой вершины в графе с вероятностью как минимум $1-\delta$, где $\delta \in O(1)$, сказать, достижима ли она из вершины v.
Рассмотрим сперва решение первой подзадачи. По **Утверждению 4** размер множества $SV(n^{1-\epsilon}\log n)$ с большой вероятностью не превышает $O(n^{\epsilon})$. То есть при параллельном поиске в ширину сразу из всех вершин множества $SV(n^{1-\epsilon}\log n)$, все уникальные идентификаторы могут быть укомплектованы в память или запрос одной машины. Выберем теперь подмножество $SN(n^{\frac{\epsilon}{2}})$, такое, что для любой вершины в графе, либо она сама, либо ее входящий сосед будут принадлежать данному подмножеству с вероятностью как минимум $1-\delta$. По **Утверждению 2** данную операцию мы так же можем произвести за $O(1)$ раундов в AMPC модели. Вершины из $SN(n^{\frac{\epsilon}{2}})$ мы назовем опорными, и каждую вершину в графе мы свяжем с минимальной по номеру опорной вершиной в ее окрестности(мы уже гарантировали, что объект для установления связи найдется с большой вероятностью). Каждую опорную вершину и все связанные с ней простые вершины мы назовем гипервершиной, далее соединим направленными ребрами все такие гипервершины, составные вершины которых имели хотя бы одно направленное ребро. По **Утверждению 2** количество вершин в получившимся гиперграфе (механизм отождествления вершин и обхода гиперграфа мы опишем подробно в соответствующем разделе, однако отметим, что данная процедура основывается на том факте, что даже в худшем случае подграф из $\lceil n^{\frac{\epsilon}{2}}\rceil$ вершин может быть помещен в памяти одной машины со всеми своими ребрами) с большой вероятностью сократиться в $\lceil n^{\frac{\epsilon}{2}}\rceil$ раз, а значит поиск в ширину из всех вершин из $SV(n^{1-\epsilon}\log n)$ нужно будет осуществить на глубину $\lceil \max(n^{1-\frac{3}{2}\epsilon}, 1)\log n\rceil$, что в свою очередь привносит доминирующий вклад в сложностную оценку всего алгоритма.
Обратимся теперь ко второй подзадаче. Чтобы реализовать данный этап,

рассмотрим подграф $G^\epsilon(SV, E^\epsilon)$, где для любых двух вершин $r, t \in SV(n^{1-\epsilon}\log n)$ множество E^ϵ содержит ребро (r, t) тогда и только тогда, когда в ходе предыдущего этапа было установлено, что t достижима из r. На данном подграфе алгоритм может быть запущен рекурсивно:

- Выбрать подмножество $SV^\epsilon(n^{\frac{\epsilon}{2}}\log n)$, и провести параллельный поиск в ширину из его вершин. (В рамках данного поиска, как было отмечено в предыдущем абзаце, путем отождествления вершин подграфа с опорными, внутри подгафа строится гиперграф, позволяющий оптимизировать процедуру поиска)
- Загрузить подграф, образованный из вершин из $SV^\epsilon(n^{\frac{\epsilon}{2}}\log n)$, в память одной машины и провести расчеты локально за один раунд (это возможно, так как даже в худшем случае число ребер в данном графе не превышает $O(n^\epsilon)$)

Замечание: получившийся подграф $G^\epsilon(SV, E^\epsilon)$ может не обладать заданными ограничениями на степень, однако данная проблема может быть легко решена посредством обобщающих подходов, которые будут описаны далее.

3.2. Графы фиксированной степени

Чтобы описать общий подход для решения задачи достижимости, рассмотрим сперва случай, когда каждая вершина графа имеет фиксированные входящую и исходящую степени $\lceil d = n^{\frac{\epsilon}{2}} \rceil$.
В первую очередь мы представим алгоритм для вычислительно наиболее сложной части, а именно - параллельного поиска в ширину из семейства вершин $SV \subset V$. Данный фрагмент может быть изложен обособленно, так как алгоритм не подразумевает какой-либо особой структуры для множества SV, кроме ограничения на количество элементов ($|SV| \in O(n^\epsilon)$).
В качестве параметров алгоритм принимает граф, объем памяти на одной машине, подмножество вершин SV, а так же глубину, на которую необходимо провести поиск. В результате работы алгоритма с большой вероятностью, каждая вершина в графе будет знать все вершины из SV, для которых она находится на расстоянии, не превышающем L.
Замечание: для каждого алгоритма в данной работе мы будем указывать два поля ограничений: ограничения на входные данные и ограничения на параметры AMPC модели.
Покажем теперь корректность **Алгоритма 3.1**, а так же оценим его сложность.

Algorithm 3.1 Поиск в ширину из семейства вершин (G, ϵ, SV, L)

Require: $N = n + m$, $S = n^{\epsilon}$, $\epsilon \in (0, 1)$, $P = \lceil n^{1+\frac{1}{2}\epsilon} \rceil$

Require: $G = G(V, E)$, $\epsilon \in (0, 1)$, $|SV| \in O(n^{\epsilon})$, $\forall v \in V$ $Deg_{out}(v) = Deg_{in}(v) = \lceil n^{\frac{\epsilon}{2}} \rceil$

1: Каждой машине присвоить произвольную вершину, в соответствии с ее номером ▷ **Комментарий:** С каждой вершиной будет ассоциировано $\lceil n^{\frac{\epsilon}{2}} \rceil$ машин

2: Выбрать подмножество $B(n^{\frac{\epsilon}{2}})$

3: Каждую вершину из $V \setminus B(n^{\frac{\epsilon}{2}})$ в графе связать с минимальной по номеру вершиной из $B(n^{\frac{\epsilon}{2}})$ среди ее входящих соседей ▷ Назовем такую вершину $B(v)$

4: В память каждой машины, ассоциированной с конкретной вершиной v, загрузить подграф, образованный всеми вершинами u, такими что $B(u) = B(v)$, и ребрами между ними, а так же всех входящих и исходящих соседей v ▷ **Комментарий:** Все вершины u, такие что $B(u) = B(v)$, обозначим $C(v)$

5: Для каждой вершины инициализируем в распределенном хранилище пустой список R, где будут храниться вершины из SV, из которых v достижима и две таблицы TC и TN с размерностями $(\lceil n^{\frac{\epsilon}{2}} \rceil, \lceil n^{\epsilon} \rceil)$, соответствующие $C(v)$ и множеству входящих соседей v ▷ **Комментарий:** Слова "таблица"и "список"в данном контексте применены условно и используются для облегчения восприятия, на самом деле мы находимся в рамках описанной в начале семантики ключ-значение и используем соответствующие двух- и трех- местные ключи

6: Проинициализировать R для всех вершин из SV самими собой

7: **for** $l = 0$, $l < \lceil \max(\frac{L}{n^{\frac{\epsilon}{2}}}, 1) \log n \rceil$ **do**

8: Для каждой вершины v соответствующие ей $\lceil n^{\frac{\epsilon}{2}} \rceil$ машин копируют $R(v)$ в соответствующие строчки таблиц для всех вершин из $C(v)$

9: Для каждой вершины v соответствующие ей $\lceil \frac{n^{\frac{\epsilon}{2}}}{2} \rceil$ машин строят объединение всех строк из $TC(v)$ и дополняют $R(v)$

10: Для каждой вершины v соответствующие ей $\lceil n^{\frac{\epsilon}{2}} \rceil$ машин копируют $R(v)$ в соответствующие строчки таблиц для всех исходящих соседей v

11: Для каждой вершины v соответствующие ей $\lceil \frac{n^{\frac{\epsilon}{2}}}{2} \rceil$ машин строят объединение всех строк из $TN(v)$ и дополняют $R(v)$

12: **end for**

Ensure: $\forall u \in SV$ and $\forall k \in V$: $dist(u, k) \leq L$ верно: k знает, что достижима из u

Лемма 5. *В результате работы* ***Алгоритма 3.1*** $\forall u \in SV$ *and* $\forall k \in V : dist(u,k) \leq L$ *с большой вероятностью верно:* k *получит отметку, что достижима из* u

Доказательство.

- **Шаг 3:** Так как $|B(n^{\frac{\epsilon}{2}})| \in O(n^{\frac{\epsilon}{2}} \log n)$ и входящая степень каждой вершины в графе равна $\lceil n^{\frac{\epsilon}{2}} \rceil$, то по **Утверждению 2**, с большой вероятностью или сама вершина или ее входящий сосед принадлежат $B(n^{\frac{\epsilon}{2}})$, то есть данный шаг алгоритма корректен и каждая вершина в графе попадет в ту или иную группу

- **Шаг 7:** Рассмотрим следующий граф: все вершины, попавшие в одну группу в ходе шага три, отождествим с одной гипервершиной, и соединим направленными ребрами все такие гипервершины, составляющие вершины которых связаны хотя бы одним направленным ребром в исходном графе.
  Число гипервершин в таком графе будет равно числу элементов в $B(n^{\frac{\epsilon}{2}} \log n)$, то есть в $\frac{n^{\frac{\epsilon}{2}}}{\log n}$ раз меньше, чем в исходном графе. Таким образом поиск достаточно провести на глубину $\lceil \frac{L \log n}{n^{\frac{\epsilon}{2}}} \rceil$, что и происходит внутри цикла, на **Шаге 7** (сперва инициализируется вся гипервершина, а затем происходит взаимодействие по ребрам между гипервершинами)

□

Лемма 6. *В результате работы* ***Алгоритма 3.1*** *каждая машина делает не более чем* $O(n^{\epsilon})$ *запросов за раунд и использует не более чем* $O(n^{\epsilon})$ *памяти.*

Доказательство.

- **Шаг 3:** Так как входящая степень каждой вершины в графе равна $\lceil n^{\frac{\epsilon}{2}} \rceil$, то просмотреть всех входящих соседей возможно за $O(n^{\frac{\epsilon}{2}})$ запросов

- **Шаг 4:** Так как входящая степень каждой вершины в графе равна $\lceil n^{\frac{\epsilon}{2}} \rceil$, то число вершин, которые будут отождествлены с данной вершиной из $B(n^{\frac{\epsilon}{2}} \log n)$ не превосходит $O(n^{\frac{\epsilon}{2}})$, а значит число ребер между ними принадлежит $O(n^{\epsilon})$

- **Шаг 5:**
  - По условию число вершин в SV не превосходит $O(n^{\epsilon})$
  - Каждую строчку в "таблице"будет заполнять одна машина, а длина строки, в свою очередь, не превосходит n^{ϵ}

- **Шаги 7-12:** Как было отмечено в предыдущем пункте каждая машина обрабатывает одну строку, длина которой ограничена размерами множества SV, то есть не превосходит $O(n^\epsilon)$

□

Лемма 7. ***Алгоритм 3.1*** *завершается через* $O(\max(\frac{L}{n^{\frac{\epsilon}{2}}}, 1)\log^2 n)$ *раундов.*

Доказательство.

- **Шаг 2:** По доказанному ранее сконструировать такое подмножество мы можем за $O(1)$ раундов
- **Шаг 3-6:** Требуют суммарно $O(1)$ раундов
- **Шаг 7:**
  - Шаги 8 и 10 требуют $O(1)$ раундов, так как объем информации для записи (размер множества SV) умещается в один запрос
  - Шаги 9 и 11 требуют $O(\log n)$ раундов каждый, так как число сравниваемых строк на каждом шаге уменьшается в два раза (на первом шаге каждая машина сравнивает две строки и записывает их объединение, таким образом, на втором шаге останется обработать половину от исходного количества строк и т.д)

Наконец, общая сложность цикла равна $O(\max(\frac{L}{n^{\frac{\epsilon}{2}}}, 1)\log^2 n)$

□

Объединим теперь все вышеизложенное в одном утверждении:

Теорема 2. *Пусть дан направленный граф* $G = (V, E)$, $|V| = n$, $|E| = m$. *Пусть так же даны* $\epsilon \in (0, 1)$, $L < n$ *и подмножество* $SV \subset V$, *такое что* $|SV| < n^\epsilon$, *а так же выполняется ограничение на степени:* $\forall v \in V\ Deg_{out}(v) = Deg_{in}(v) = \lceil n^{\frac{\epsilon}{2}} \rceil$. *В таком случае существует вероятностный AMPC алгоритм, который решает задачу поиска в ширину из множества вершин* SV *на глубину* L *за* $O(\max(1, \frac{L}{n^{\frac{\epsilon}{2}}})\log^2 n)$ *раундов, используя объем памяти* n^ϵ *и* $\lceil n^{1+\frac{1}{2}\epsilon} \rceil$ *машин.*

Далее мы приведем сам алгоритм поиска всех вершин, достижимых из конкретной вершины $v \in V$.

Algorithm 3.2 Достижимость из одной вершины (G, v, ϵ)

Require: $N = n + m,\ S = n^{\epsilon},\ \epsilon \in (0, 1),\ P = \lceil n^{1+\frac{1}{2}\epsilon} \rceil$
Require: $G = G(V, E),\ \epsilon \in (0, 1),\ \forall v \in V\ Deg_{out}(v) = Deg_{in}(v) = \lceil n^{\frac{\epsilon}{2}} \rceil$
1: Каждой машине присвоить произвольную вершину, в соответствии с ее номером
2: Выбрать подмножество $SV(n^{1-\epsilon} \log n)$ и включить туда вершину v
3: Запустить **Алгоритм 3.1** $(G, \epsilon, SV(n^{1-\epsilon} \log n), 2\lceil n^{1-\epsilon} \log n \rceil)$
4: Создать пустой граф $G' = G'(SV, NULL)$
5: **for** $v \in SV(n^{1-\epsilon} \log n)$ **do**
6: $\quad \forall u : dist(u, v) \leq \lceil n^{\frac{\epsilon}{2}} \rceil$ записать входящее ребро (u, v) to G'
7: **end for**
8: В графе G' выбрать подмножество $SV'(n^{\frac{\epsilon}{2}})$ и включить туда вершину v
9: Запустить **Алгоритм 3.1** $(G', \epsilon, SV'(n^{\frac{\epsilon}{2}}), 2\lceil n^{\frac{\epsilon}{2}} \rceil)$
10: Локально установить достижимость из v для вершин из $SV'(n^{\frac{\epsilon}{2}})$
Ensure: $\forall u \in V : dist(v, u) \leq \infty$ верно: u знает, что достижима из v

Лемма 8. *В результате работы* ***Алгоритма 3.2*** *с большой вероятностью* $\forall u \in V : dist_(v, u) \leq \infty$ *верно: и знает, что достижима из* v

Доказательство.
- Для вершин из $SV'(n^{\frac{\epsilon}{2}})$ достижимость устанавливается в ходе локальной обработки графа G'
- Для вершин из $SV(n^{1-\epsilon} \log n) \setminus SV'(n^{\frac{\epsilon}{2}})$: По **Утверждению 4** для любой вершины $u \in V$ верно, что если существует путь $P(v, u) : |P| \geq n^{\frac{\epsilon}{2}}$, то с большой вероятностью существует вершина $k \in SV'(n^{\frac{\epsilon}{2}}) : k \in P$ и $dist_P(k, u) \leq 2\lceil n^{\frac{\epsilon}{2}} \rceil$. Таким образом для любой вершины заключение о ее достижимости из вершины v можно сделать, проверив на достижимость из v все такие вершины $k \in SV'(n^{\frac{\epsilon}{2}}) : k \in P$ и $dist_P(k, u) \leq 2\lceil n^{\frac{\epsilon}{2}} \lceil$
- Для вершин из $V \setminus SV(n^{1-\epsilon} \log n)$: Аналогично предыдущему пункту, но поиск нужно провести на глубину $\lceil n^{1-\epsilon} \log n \rceil$

□

Лемма 9. *В результате работы* ***Алгоритма 3.1*** *каждая машина делает не более чем* $O(n^{\epsilon})$ *запросов за раунд и использует не более чем* $O(n^{\epsilon})$ *памяти.*

Доказательство.
- **Шаги 3 и 9:** По **Лемме 6** ограничения на память и число запросов выполняются

- **Шаг 5:** Каждая машина будет добавлять ребра, соответствующие одной вершине, и таких ребер будет не больше, чем элементов в $SV(n^{1-\epsilon}\log n)$, то есть $O(n^{\epsilon})$
- **Шаг 10:** Число ребер в графе с $\lceil n^{\frac{\epsilon}{2}} \rceil$ вершинами не превышает $O(n^{\epsilon})$

□

Лемма 10. ***Алгоритм 3.2*** *завершается через* $O(n^{1-\frac{3}{2}\epsilon}\log^2 n)$ *раундов.*

Доказательство.
- **Шаг 2:** По доказанному ранее сконструировать такое подмножество мы можем за $O(1)$ раундов путем моделирования случайной величины
- **Шаг 3:** По **Утверждению 7** поиск требует $O(n^{1-\epsilon}\log n \frac{\log n}{n^{\frac{\epsilon}{2}}}) = O(n^{1-\frac{3}{2}\epsilon}\log^2 n)$ раундов
- **Шаг 5:** Может быть завершен за один раунд, так как каждая машина добавляет ребра, соответствующие одной вершине, и их объем не превышает размера одного запроса
- **Шаг 8:** По доказанному ранее сконструировать такое подмножество мы можем за $O(1)$ раундов путем моделирования случайной величины
- **Шаг 9:** По **Утверждению 7** поиск требует $O(\log n)$ раундов
- **Шаг 10:** Может быть завершен за один раунд

□

Теорема 3. *Пусть дан направленный граф* $G = (V, E)$, $|V| = n$, $|E| = m$. *Пусть так же даны* $\epsilon \in (0, 1)$ *и вершина* $v \in V$, *а так же выполняется ограничение на степени:* $\forall v \in V\ Deg_{out}(v)(v) = Deg_{in}(v)(v) = \lceil n^{\frac{\epsilon}{2}} \rceil$. *В таком случае существует вероятностный AMPC алгоритм, который решает задачу поиска всех вершин, достижимых из вершины* v, *за* $O(\max(n^{1-\frac{3}{2}\epsilon}, 1)\log^2 n)$ *раундов, используя объем памяти* n^{ϵ} *и* $\lceil n^{1+\frac{1}{2}\epsilon} \rceil$ *машин.*

3.3. Обобщение для графов степени меньшей фиксированной

В данной секции мы опишем модернизированную версию алгоритма, позволяющую обрабатывать графы, степень которых меньше $\lceil n^{\frac{\epsilon}{2}} \rceil$.
Основная идея подхода заключается в следующем: в худшем случае (если множества соседей сильно пересекаются) используя $O(n^{\epsilon})$ запросов

можно обнаружить как минимум $O(n^{\frac{\epsilon}{2}})$ новых вершин, начав поиск из фиксированной вершины. Таким образом, за $O(1)$ раундов можно увеличить степень каждой вершины до $\lceil n^{\frac{\epsilon}{2}} \rceil$ (аналогично для входящих и исходящих степеней).
Если для какой-то вершины не получилось найти $\lceil n^{\frac{\epsilon}{2}} \rceil$ уникальных вершин, то данная вершина и весь найденный подграф представляют собой набор компонент сильной связности, которые не содержат более никаких вершин извне и могут быть обработаны за один раунд локально. Данный факт следует из наблюдения, что компонента сильной связности представляет собой транзитивное замыкание множеств достижимостей вершин. Однако, полностью подобные вершины из графа выбросить мы не можем, так как это повлечет за собой уменьшение степеней всех остальных вершин. Чтобы устранить данное противоречие мы введем виртуальный конгломерат из $\lceil n^{\frac{\epsilon}{2}} \rceil$ вершин, и связанными с ними ребрами дополним недостающие степени.
Замечание для входящих и исходящих степеней процедура происходит аналогично, за исключением направления ребер.
Дополнительно отметим, что в ходе данной процедуры количество вершин вырастет на $\lceil n^{\frac{\epsilon}{2}} \rceil$, что в свою очередь не меняет доминирующего слагаемого в асимптотике в терминах O-большого. Количество ребер в свою очередь вырастает более значительно, однако в представленных ранее рассуждениях данный параметр фигурировал опосредованно. В данной работе мы не делаем каких либо различий для реберной классификации графов(на плотные или разреженные).

Теорема 4. *Пусть дан направленный граф $G = (V, E)$, $|V| = n$, $|E| = m$. Пусть так же даны $\epsilon \in (0, 1)$ и вершина $v \in V$, а так же выполняется ограничение на степени: $\forall v \in V\ Deg_{out}(v) \leq \lceil n^{\frac{\epsilon}{2}} \rceil$, $Deg_{in}(v) \leq \lceil n^{\frac{\epsilon}{2}} \rceil$. В таком случае существует вероятностный AMPC алгоритм, который решает задачу поиска всех вершин, достижимых из вершины v, за $O(\max(n^{1-\frac{3}{2}\epsilon}, 1) \log^2 n)$ раундов, используя объем памяти n^ϵ и $\lceil n^{1+\frac{1}{2}\epsilon} + n^\epsilon \rceil$ машин.*

Algorithm 3.3 Увеличение степени (G, d)

Require: $G = G(V, E) : \forall v \in V\ Deg_{out}(v) \geq 1,\ Deg_{in}(v) \geq 1$
Ensure: $G' = G'(V', E') : \forall v \in V'\ Deg_{out}(v) \geq \lceil n^{\frac{\epsilon}{2}} \rceil, Deg_{in}(v) \geq \lceil n^{\frac{\epsilon}{2}} \rceil$

1: Каждой машине присвоим произвольную вершину, в соответствии с ее номером
2: **for** $v \in V$ **do**
3: $cIn = Deg_{in}(v)$
4: Начать поиск в ширину из v, увеличивая счетчик cIn при обнаружении новой вершины, прекратить поиск если число запросов превысило $O(n^{\epsilon})$
5: $cOut = Deg_{out}(v)$
6: Начать поиск в ширину из v, увеличивая счетчик $cOut$ при обнаружении новой вершины, прекратить поиск если число запросов превысило $O(n^{\epsilon})$
7: **if** $cOut \leq d$ **then**
8: Добавить недостающее число виртуальных ребер
9: **end if**
10: **if** $cIn \leq d$ **then**
11: Добавить недостающее число виртуальных ребер
12: **end if**
13: **end for**

3.4. Обобщение для произвольного графа

Опишем теперь подход, который позволит перейти к рассмотрению графов со степенями, превышающими $\lceil n^{\frac{\epsilon}{2}} \rceil$.

Каждую вершину, степень которой превышает заданный параметр, мы можем превратить в сбалансированное дерево, степень каждой вершины в котором равна нужной нам величине. Данный метод сделает количество вершин в графе асимптотически близким к числу ребер, однако замечательным фактом является то, что даже при замене всех вершин внутри некоторого пути на подобные деревья, его длина вырастет лишь в константу раз. То есть, таким образом, общая сложность алгоритма поиска не вырастет в терминах О-большого. Тем не менее, важно отметить, что количество машин, которые будут использоваться в данном случае, станет пропорционально числу ребер в рассматриваемом графе. Аналогично сложность такой перестройки не будет превышать $O(1)$ раундов (по количеству уровней в дереве).

Замечание для того, чтобы сохранить условие на минимальные степени вершин в получившемся дереве, всех прямых потомков одного родителя нужно соединить промежуточными ребрами - это увеличит максимальную степень до $O(n^{\frac{\epsilon}{2}})$, что все еще соответствует требованиям, описан-

ным в предыдущих алгоритмах.

Algorithm 3.4 Уменьшение степени (G, d)

Require: $G = G(V, E) : \forall v \in V \; Deg_{out}(v) = Deg_{in}(v) \geq \lceil n^{\frac{\epsilon}{2}} \rceil$
Ensure: $G' = G'(V', E') : \forall v \in V' Deg_{out}(v) = Deg_{in}(v) = \lceil n^{\frac{\epsilon}{2}} \rceil$ или $2\lceil n^{\frac{\epsilon}{2}} \rceil$
  Каждой машине присвоим произвольную вершину, в соответствии с ее номером ▷ С каждой вершиной будет ассоциировано $\lceil n^{1-\epsilon} \rceil$ машин
  for $v \in V$ **do**
    if $Deg_{out}(v) \geq \lceil n^{\frac{\epsilon}{2}} \rceil$ **then**
      $\lceil n^{1-e} \rceil$ машин строят дерево по уровням
    end if
    if $Deg_{in}(v) \leq \lceil n^{\frac{\epsilon}{2}} \rceil$ **then**
      $\lceil n^{1-e} \rceil$ машин строят дерево по уровням
    end if
  end for

Теорема 5. *Пусть дан направленный граф $G = (V, E)$, $|V| = n$, $|E| = m$. Пусть так же даны $\epsilon \in (0, 1)$, $L < n$ и вершина $v \in V$. В таком случае, существует AMPC алгоритм, который решает задачу поиска всех вершин, достижимых из вершины v, за $O(\max(n^{1-\frac{3}{2}\epsilon}, 1) \log^2 n)$ раундов, используя объем памяти n^ϵ и $\lceil m^{1+\frac{1}{2}\epsilon} \rceil$ машин.*

4. Компоненты сильной связности

В заключение, приведем алгоритм поиска компонент сильной связности, который будет в значительной степени опираться на описанную ранее процедуру поиска достижимых вершин. Данный алгоритм был описан в [8] и позволяет вычислить компоненты сильной связности за $O(\log^2 n)$ запросов о достижимости из отдельной вершины.
Ключевая идея алгоритма состоит в разбиении графа на подграфы, которые точно не пересекаются ни по одной из сильносвязных компонент, в свою очередь обозначенные подграфы обрабатываются рекурсивно. Такой подход позволяет более эффективно реализовать параллельное решение задачи. Разбиение на подграфы осуществляется за счет следующего наблюдения:

Теорема 6. *[8] Пусть V - множество всех вершин в графе и $N \subset V$ - некоторое его подмножество, пусть так же вершина v не принадлежит N. Обозначим:*

- *A = вершины, достигнутые из набора вершин N*

- B = *вершины, достигнутые из вершины* v
- C = *вершины, которые достигают* v *и достигаются из* v.

Тогда для следующих подмножеств вершин: $V \setminus (A \cup B), A \setminus B, B \setminus (A \cup C)$ *и* $(A \cap B) \setminus C$ *верно, что никакие вершины из разных подмножеств не лежат в одной компоненте сильной связности.*

Далее в своей работе авторы приводят утверждение, что при определенном выборе вершины v и множества C на каждом рекурсивном шаге, суммарное количество запросов о достижимости не превысит $O(\log^2 n)$. Обобщим теперь все вышесказанное, сформулировав финальное утверждение о сложности поиска компонент сильной связности в AMPC модели. В предыдущих разделах, учитывая рассуждения об общении на графы произвольной степени, мы показали, что существует AMPC алгоритм, который с большой вероятностью решает задачу достижимости из одной вершины за $O(\max(n^{1-\frac{3}{2}\epsilon}, 1)\log^2 n)$ раундов, используя объем памяти n^ϵ и $\lceil m^{1+\frac{1}{2}\epsilon} \rceil$ машин.
Таким образом, подставив данную оценку, в приведенный в [8] алгоритм, получим следующее утверждение:

Теорема 7. *Пусть дан направленный граф* $G = (V, E)$, $|V| = n$, $|E| = m$. *Пусть так же дано* $\epsilon \in (0, 1)$. *В таком случае существует вероятностный AMPC алгоритм, который решает задачу поиска компонент сильной связности за* $O(\max(n^{1-\frac{3}{2}\epsilon}, 1)\log^4 n)$ *раундов, используя* $\lceil m^{1+\frac{1}{2}\epsilon} \rceil$ *машин и объем памяти* n^ϵ.

5. Заключение

В рамках данной работы был получен вероятностный алгоритм поиска компонент сильной связности в графе в AMPC модели. Для описанного подхода были приведены доказательства корректности и сложности, а так же представлены оценки на минимально необходимый объем вычислительных ресурсов(памяти и числа машин). В качестве ключевых особенностей данной реализации можно выделить следующие аспекты: полилогарифмическая или сублинейная сложность при относительно больших объемах локальной памяти, а так же сублинейные по числу вершин(а не ребер) в графе требования к памяти и количеству запросов(что зачастую является наиболее сложным и интересным сценарием).

Distributed strongly connected components search in an adaptive model
Moldovanov I.V.

To study the efficiency of distributed algorithms, a number of mathematical formalizations with new concepts of algorithm complexity are introduced. In this paper complexity of finding strongly connected components in the AMPC model is investigated. AMPC model, unlike other more limited distributed formalizations, allows to build a query tree within one step of the algorithm. A probabilistic algorithm is obtained that implements strongly connected components search in polylogarithmic or sublinear time (depending on the amount of available local memory). The amount of required local memory is sublinear with respect to the number of vertices in the graph.

Keywords: distributed algorithms, probabilistic algorithms, strongly connected components.

References

[1] Grzegorz M.M., Austern Aart J.C Bik James C. Dehnert Ilan Horn Naty Leiser Grzegorz Czajkowski, "Pregel: a system for large-scale graph processing", *Proceedings of the 2010 international conference on Management of data, ACM*, 2010, 135–146.

[2] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii, "A model of computation for mapreduce.", *In Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms, SIAM*, 2010, 938—948.

[3] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, "Massively Parallel Computation via Remote Memory Access.", *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, 2019.

[4] Soheil Behnezhad, Laxman Dhulipala, Hossein Esfandiari, "Parallel graph algorithms in constant adaptive rounds: theory meets practice.", *Proceedings of the VLDB Endowment*, **13** (2020).

[5] Tarjan R. E., "Depth-first search and linear graph algorithms.", *SIAM Journal on Computing.*, **1** (1972), 146—160.

[6] Micha Sharir., "A strong-connectivity algorithm and its applications to data flow analysis.", *Computers and Mathematics with Applications.*, **7** (1981), 67—72.

[7] Fleischer, Lisa K.; Hendrickson, Bruce; Pınar, Ali, "On Identifying Strongly Connected Components in Parallel.", *Parallel and Distributed Processing, Lecture Notes in Computer Science.*, **1800** (2000), 505—511.

[8] Warren Schudy, "Finding strongly connected components in parallel using $O(\log^2 n)$ reachability queries.", *SPAA '08: Proceedings of*

the twentieth annual symposium on Parallelism in algorithms and architectures., 2008, 146-–151.

[9] J. D. Ullman and M. Yannakakis., "High probability parallel transitive-closure algorithms.", *SIAM J.Comput.*, **20(1)** (1991), 100—125.

[10] Alexandr Andoni, Zhao Song, Clifford Stein, "Parallel graph connectivity in log diameter rounds.", *59th IEEE Annual Symposiumon Foundations of Computer Science, FOCS.*, 2018, 674—685.

[11] Chernoff, Herman, "A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the sum of Observations.", *The Annals of Mathematical Statistics.*, **23(4)** (1952), 493—507.