Санкт-Петербургский государственный университет

Порсев Денис Витальевич

Экспериментальный анализ реализации алгоритмов на графах с использованием операций линейной алгебры

28 мая 2021 г.

Санкт-Петербург
2021

1 Введение

Современные компьютерные архитектуры позволяют легко обрабатывать линейные и иерархические структуры данных, такие как листы, стеки или деревья. Задачи обработки различных графов же зачастую имеют неструктурированный характер. В них отсутствует векторизация, в связи с чем распараллеливание и оптимизация алгоритмов на графах становятся трудными задачами, нерегулярный доступ к памяти вызывает промахи в кэше. В то же время алгоритмы на графах можно преобразовать к последовательности матрично-векторных операций, адаптировав для этого только базовые операции линейной алгебры. Что вместе со стандартизацией модели хранения различных видов графов в памяти в виде разреженной матрицы поможет упростить оптимизацию кода обработки графа.

В данной работе будет проведен анализ производительности алгоритмов на графах с использование операций линейной алгебры. Автором были реализованы следующие алгоритмы: поиск в ширину, подсчет треугольников, поиск кратчайших путей (алгоритм Беллмана-Форда). А именно, будет проведено сравнение реализаций вышеперечисленных алгоритмов с помощью библиотеки pygraphblas, являющейся оберткой написанной на языке python над API спецификацией GraphBlas, предоставляющей набор стандартных операций над матрицами и векторами. Реализации алгоритмов с помощью библиотеки SciPy, предназначенной для решения различных научных и инженерных математических проблем, а также реализации, предоставляемой стандартной библиотекой анализа графов NetworkX, специально предназначенной для работы с графами и другими сетевыми структурами.

2 Проведение эксперимента

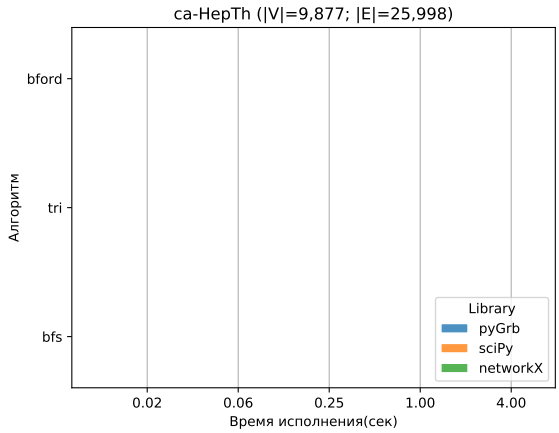
Измерения производились на компьютере со следующими характеристиками: процессор AMD A10-5757M 2.5 GHz, 8 Гб оперативной памяти DDR3, под управлением операционной системы Ubuntu 20.04.2 LTS.

В качестве исходных данных были использованы датасеты SNAP (Stanford Network Analysis Platform[3]) взятые из SuiteSparse Matrix Collection — коллекции разреженных матриц реальных данных[2]. А именно, наборы данных ca-AstroPh[1], ca-CondMat, ca-HepTh описывающие соавторство в научных работах в виде неориентированного графа. Наборы amazon-0302, amazon-0312, amazon-0505, amazon-0601, представляющие собой ориентированные графы, собранные парсингом сайта Amazon с промежутком в несколько месяцев, а также неориентированный граф социальной сети com-Youtube, в котором более миллиона вершин.

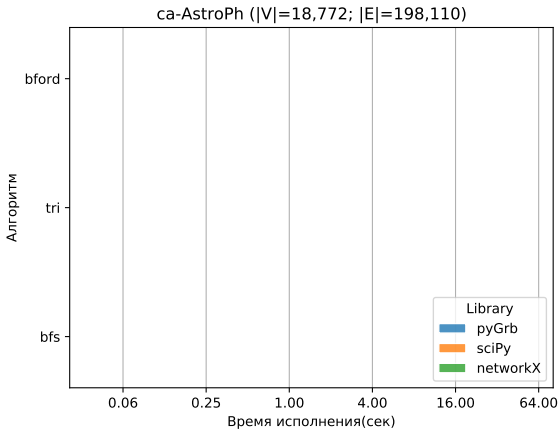
Эксперимент был поставлен следующим образом. В память программы загружался граф из датасета, после чего случайным образом выбиралась начальная вершина для

поиска в ширину и поиска кратчайшего пути (для реализованного алгоритма подсчета треугольников в графе начальная вершина не требуется). Затем к этому графу последовательно применялись упомянутые алгоритмы и измерялось время исполнения каждого. Для измерения времени использовалась библиотека time, значения сохранялись в долях секунды. Для датасетов *ca* выполнялось 10 итераций, для остальных датасетов алгоритмы исполнялись 5 раз ввиду больших размеров графов. Полученные временные значения записывались в файл.

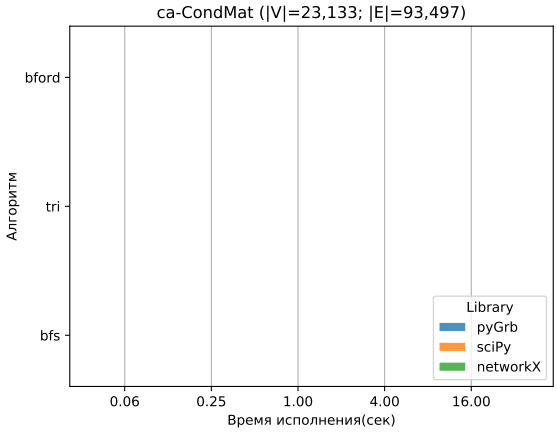
На рисунках 1-4 представлены результаты проведенных измерений.



(a) Набор данных ca-HepTh



(b) Набор данных ca-AstroPh



(c) Набор данных ca-CondMat

Рис. 1: Времена работы алгоритмов на наборах данных *ca*.
Среднее 10 замеров.

Рисунок 1 иллюстрирует среднее время исполнения алгоритмов реализованных с помощью разных библиотек в виде гистограммы. Такое представление удобно использовать в случае, когда данные можно сгруппировать по категориям. На временной оси используется логарифмический масштаб. Это связано с тем, что алгоритм Беллмана-Форда в реализации на SciPy работает существенно медленнее остальных вариантов.

На графе другого типа — amazon-0302, состоящем из большего числа вершин и ребер было проверено, не является ли плохая производительность алгоритма поиска кратчайшего пути на SciPy зависимой от входных данных первого эксперимента. Алгоритм поиска треугольников к графам типа *amazon* на применялся, так как написанные реализации считают треугольники только в неориентированном графе. Результаты представлены на рисунке 2.

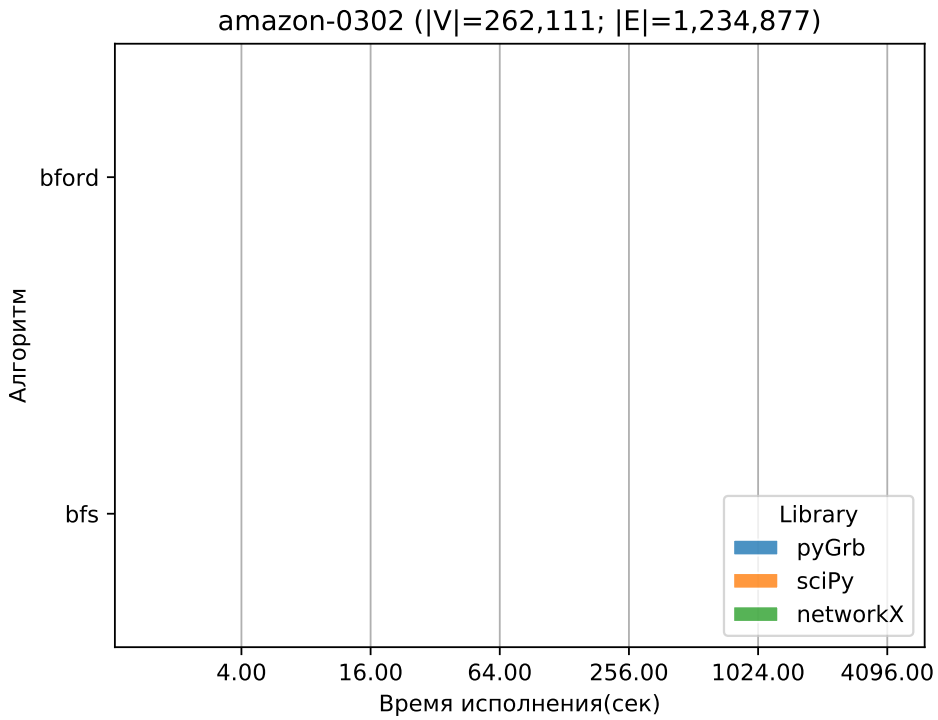
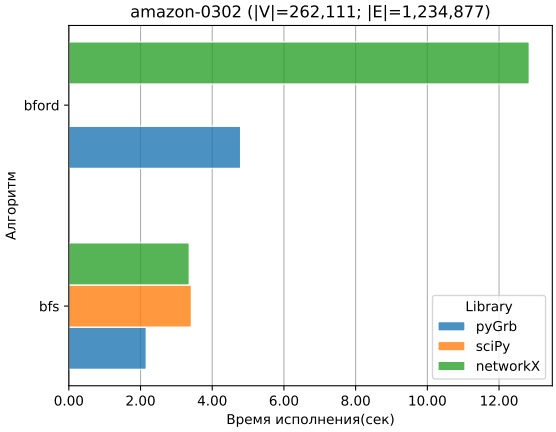


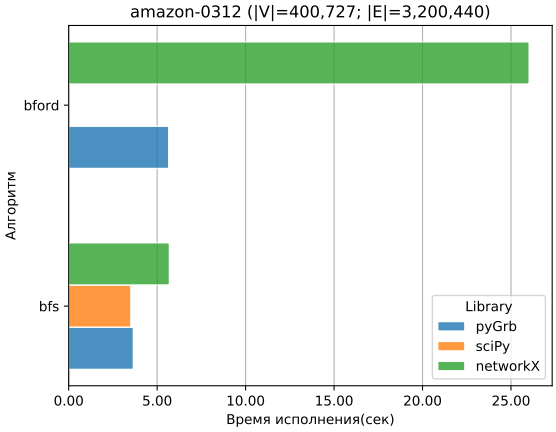
Рис. 2: Результаты работы алгоритмов на графе amazon-0302.
Среднее 5 замеров.

После чего производительность реализаций поиска в ширину и алгоритма Беллмана-Форда были проанализированы на графах amazon-0312, amazon-0505, amazon-0601. Они интересны тем, что были собраны с одной сети с разницей не больше чем в месяц друг от друга, благодаря чему производительность алгоритмов можно оценить на одинаково структурированных начальных данных с разным количеством вершин и ребер (рисунок 3). Время исполнения Беллмана-Форда на SciPy было принято за 0, чтобы на графике линейного масштаба разница в производительности алгоритмов была видна нагляднее.

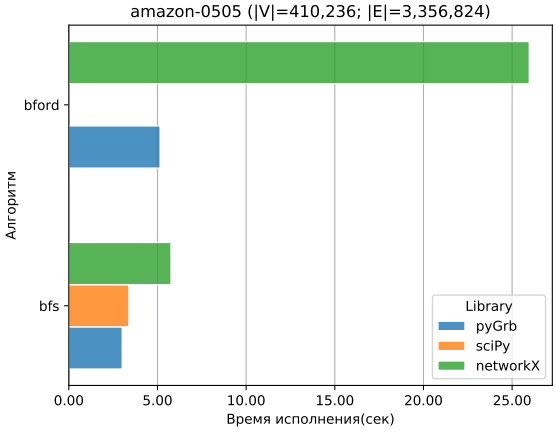
В заключении был проанализирован граф с существенно превосходящим числом вершин и примерно равным числом ребер относительно графов *amazon*. Результаты представлены на рисунке 4.



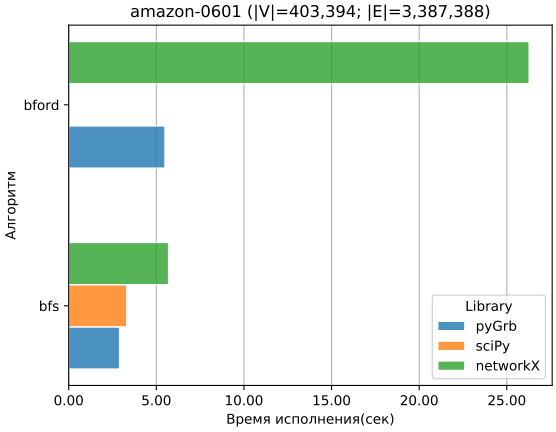
(a) Набор данных amazon-0302



(b) Набор данных amazon-0312



(c) Набор данных amazon-0505



(d) Набор данных amazon-0601

Рис. 3: Времена работы алгоритмов на наборах данных *amazon*.
Среднее 5 замеров.

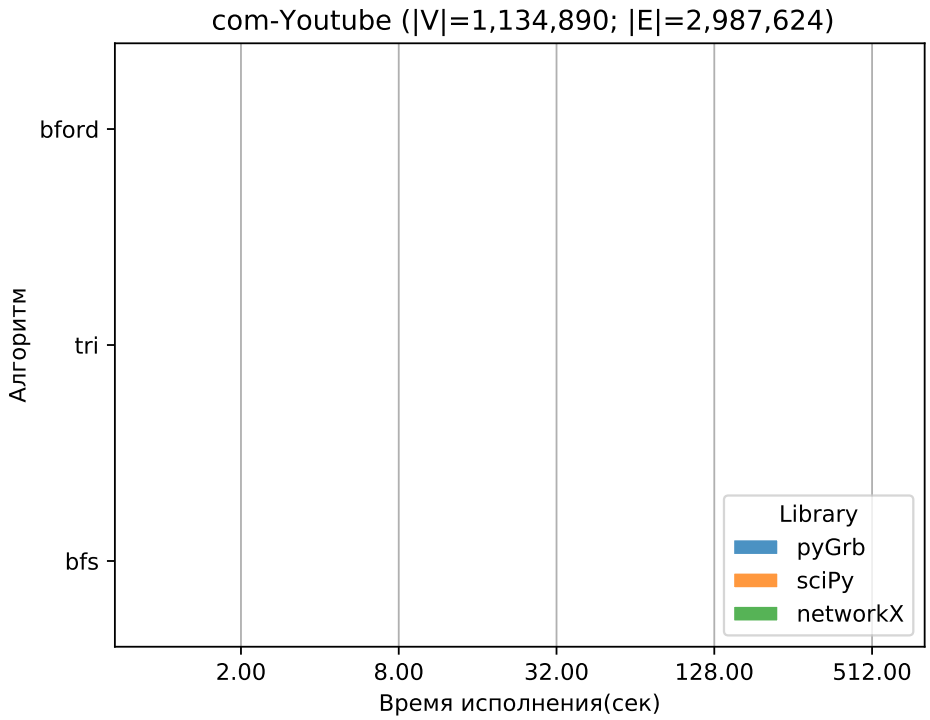


Рис. 4: Результаты работы алгоритмов на графе com-Youtube. Среднее 5 замеров.

3 Заключение

В результате проведенных экспериментов было получено:

1. Реализация алгоритмов с помощью операций линейной алгебры на pygraphblas оказалось самой эффективной. Наибольшая разница обнаружилась в алгоритме подсчета треугольников. Это можно объяснить тем, что его реализация в большей степени основана на перемножении матриц, которое в pygraphblas максимально оптимизировано. В поиске в ширину были отмечены наименьшие различия. Это можно обосновать использованием меньшего числа операций линейной алгебры в реализации.

2. Хочется отметить, что в реализации алгоритмов поиска в ширину и подсчета треугольников на SciPy были использованы операции линейной алгебры, из-за чего код реализации этих алгоритмов на pygraphblas и SciPy получился практически идентичным. Из этого можно сделать вывод о том, что эти операции не так эффективно адаптированы в SciPy по сравнению с pygraphblas. Тем не менее, с увеличением числа вершин графа реализации с помощью SciPy все заметнее опережали стандартные решения, используемые NetworkX.

3. Беллман-Форд на SciPy использовал одноименную функцию из библиотеки[4], что может объяснить такой непропорционально большой отрыв во времени исполнения в сравнении с другими алгоритмами. По всей видимости, проверки на отрицательные циклы повлияли на время исполнения алгоритма на больших графах. Однако результат оказался гораздо медленнее ожидаемого, даже с учетом проверок.

4. По полученным графикам на наборах данных *amazon* можно судить о пропорциональной зависимости размеров графов ко времени исполнения алгоритмов. Однако окончательный анализ о существовании такой зависимости стоит провести на более разнородных данных. Возможно, стоит использовать датасеты графов с большей разницей в размерах, при этом имеющих одинаковую структуру.

Список литературы

[1] G. A. Davis and Y. Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(Article 1 (December 2011)):25, 2011.

[2] S. P. Kolodziej, M. Aznaveh, M. Bullock, J. David, T. A. Davis, M. Henderson, Y. Hu, and R. Sandstrom. The suitesparse matrix collection website interface. *Journal of Open Source Software*, 4(35 (March 2019)):1244–1248, 2019.

[3] J. Leskovec. Stanford large network dataset collection.

[4] scipy.org. scipy.sparse.csgraph.bellman_ford.