Отчет о постановке эксперимента

Система: Ubuntu 18.04
Intel Core i7-7500U 2.70GHz
RAM: 8Gb DDR4

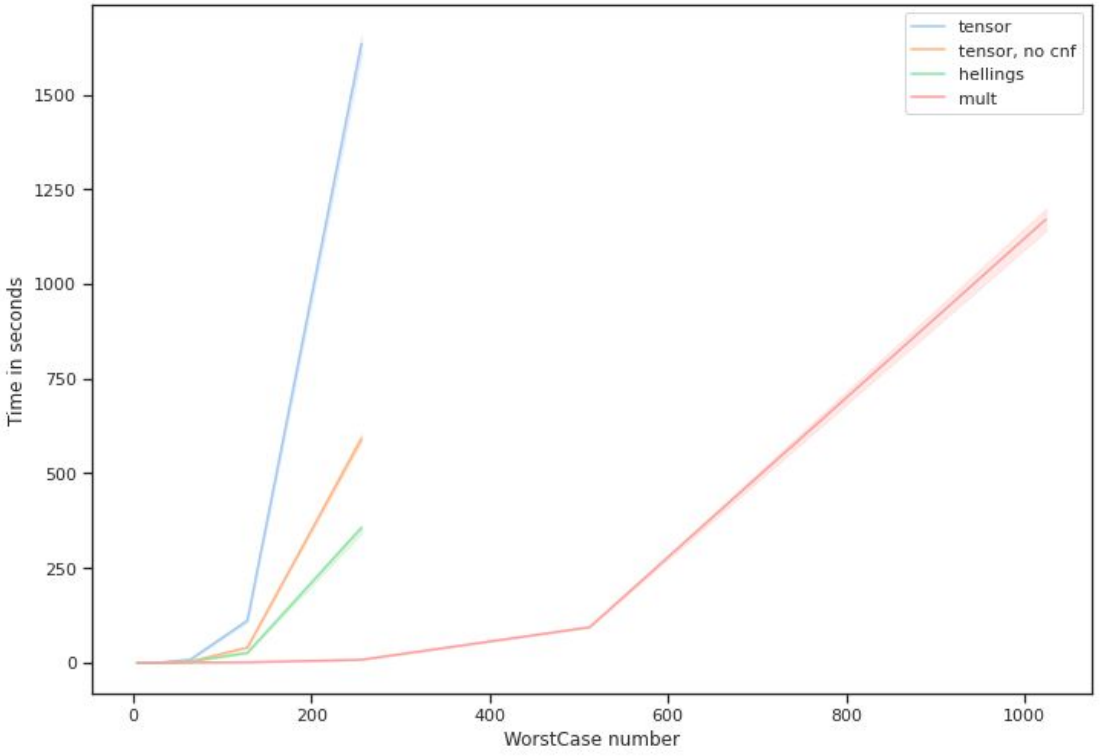
Автор: Ахметьянов Азат, СПбГУ
a.r.akhmetyanov@gmail.com

Экспериментальный анализ

При проведении экспериментов работа алгоритмов вычисления контекстно-свободных запросов на каждой паре граф-запрос повторялась 2-3 итерации, контрольные числа и среднее значение времени исполнения для алгоритма записывались в CSV файл с результатами, затем данные анализировались в Jupyter Notebook-е.

Результаты анализа для набора данных WorstCase

Результаты визуализированы с помощью графика *lineplot* библиотеки *seaborn* (этот вид графика показывает линию, в точках которой средние значения с дополнительно обозначенным доверительным интервалом). Поскольку графы данного набора синтетические и имеют одинаковую структуру, такое представление хорошо подходит для того чтобы сравнить работу алгоритмов в зависимости от размера графа.



Все методы, кроме алгоритма с матричным умножением, начиная с графа WorstCase_512 превысили установленное разумное время работы одной итерации в 30 минут и их работа была предварительно завершена скриптом для анализа (поэтому их графики прерываются). Но, на графике видно, что даже на WorstCase_1024 время выполнения алгоритма с перемножением матриц сравнимо с временем других алгоритмов на WorstCase_256.

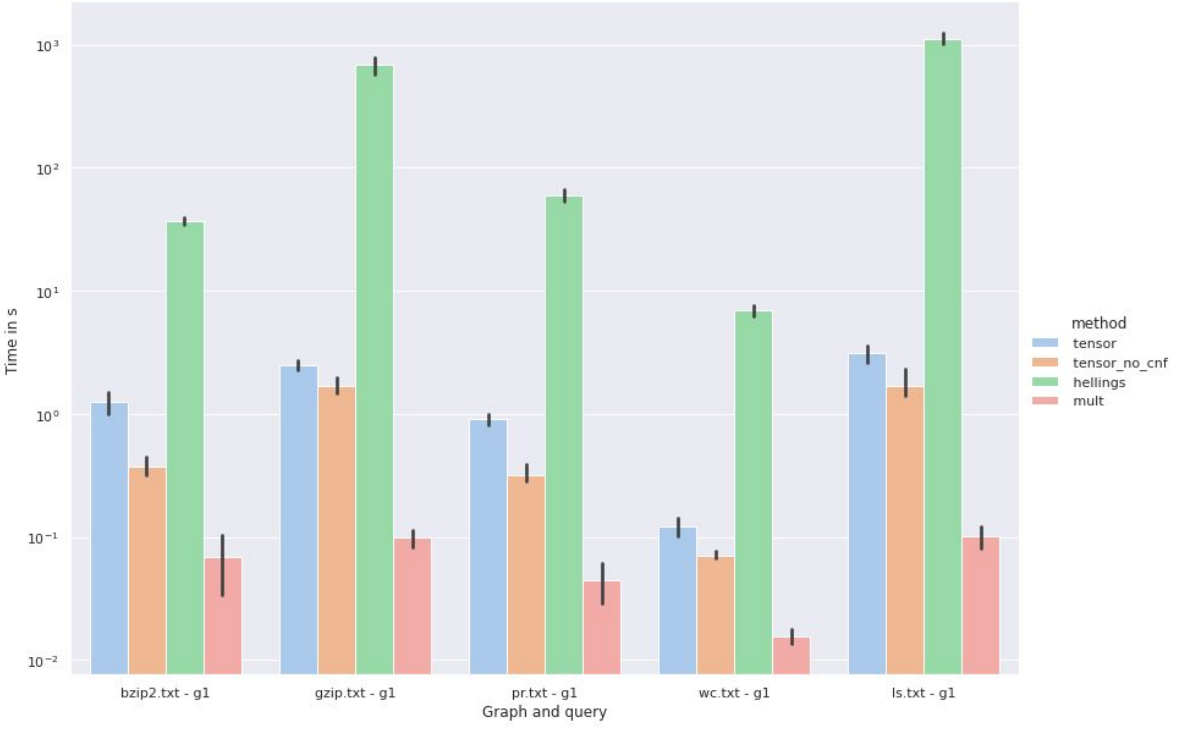
Алгоритм, использующий тензорное произведение, сработал заметно быстрее с грамматикой не в ОНФХ, что обусловлено меньшим числом состояний в рекурсивном автомате, и, как следствие, меньшим размером матрицы.

Дальнейшие эксперименты с этим набором не проводились, так как графы обладают однородной структурой и поведение алгоритмов относительно друг друга на таких структурах можно хорошо оценить по уже проведенным экспериментам.

Результаты анализа для набора данных MemoryAliases

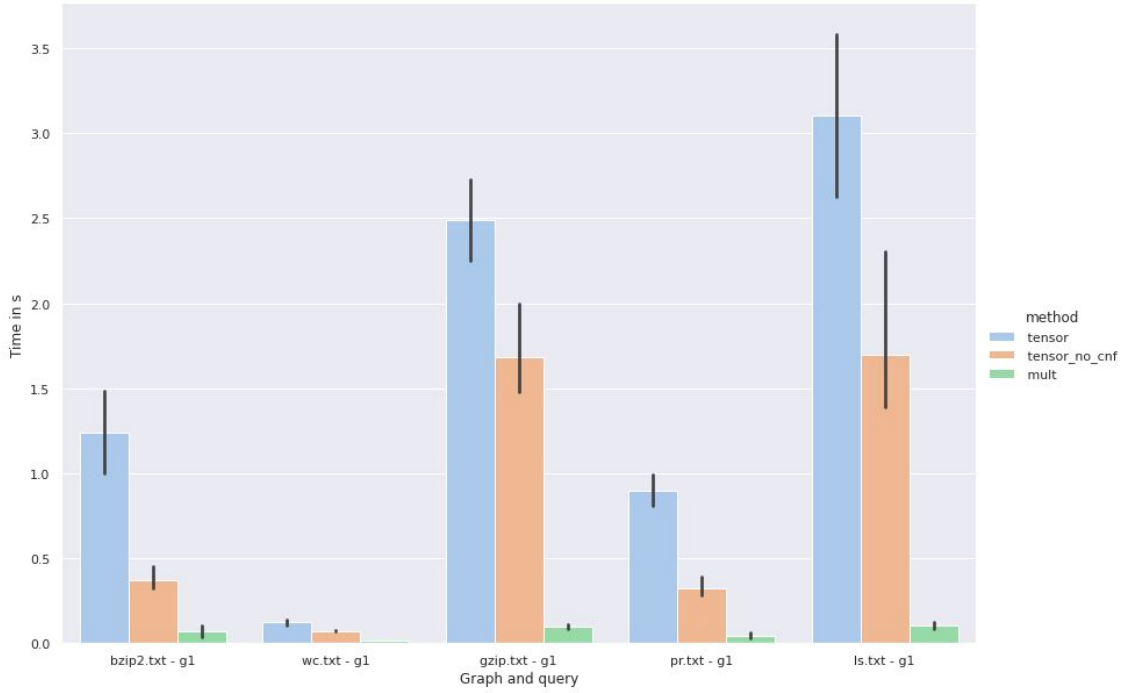
Этот набор интересен во-первых тем, что содержит графы, построенные на основе кода реальных программ, а во-вторых наличием в нем запроса с регулярным выражением в теле. Поскольку графы неоднородной структуры, наглядно использовать линейный график не получилось бы, и был применен *catplot()* библиотеки seaborn для пар граф-запрос. Он отражает средние значения по категориям с доверительным интервалом.

График для грамматики g1 со всеми методами
(на данном графике шкала времени сделана логарифмической, т.к. иначе значения времени у алгоритма Хеллингса, многократно превышающие другие, не оставляли возможности рассмотреть хоть какую-то разницу во времени других алгоритмов)



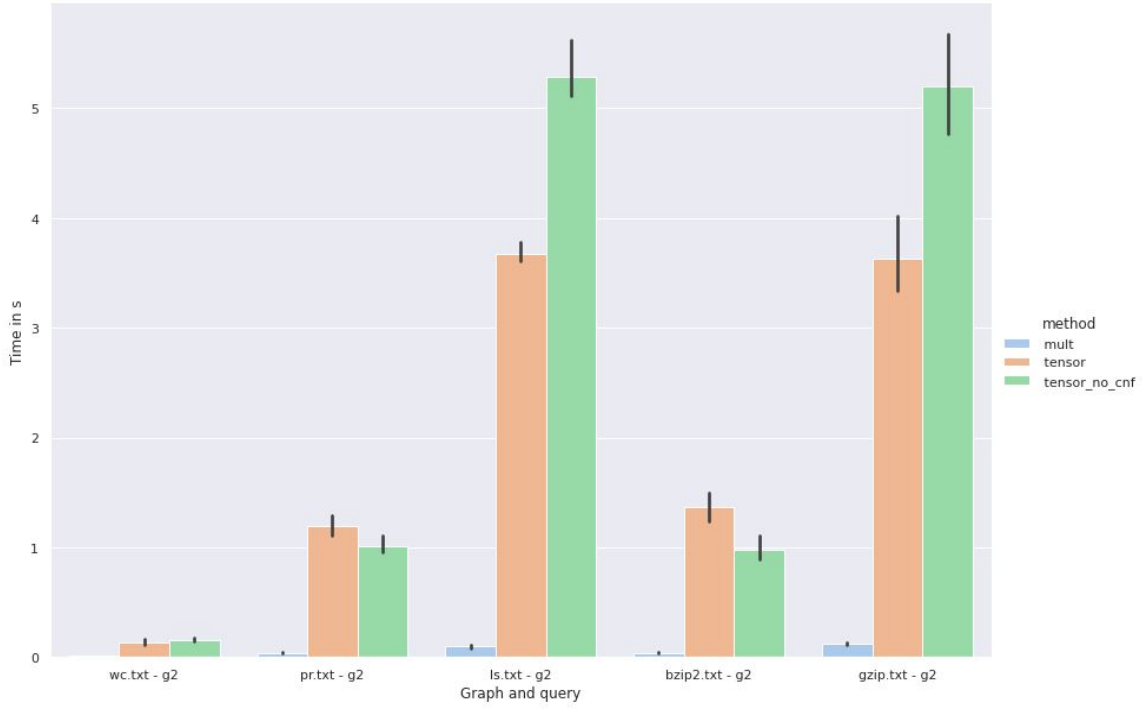
Из данного графика видно, что время работы алгоритма Хеллингса на этом наборе в среднем превышает время других алгоритмов на два порядка и достигает 15-16 минут на одну итерацию.

График для грамматики g1, все методы кроме алгоритма Хеллингса
(шкала времени обычная)



На графике тремя методами заметно, что для *g1* тензорный алгоритм работает быстрее с запросом без преобразования в ОНФХ (также обосновано меньшим числом состояний).

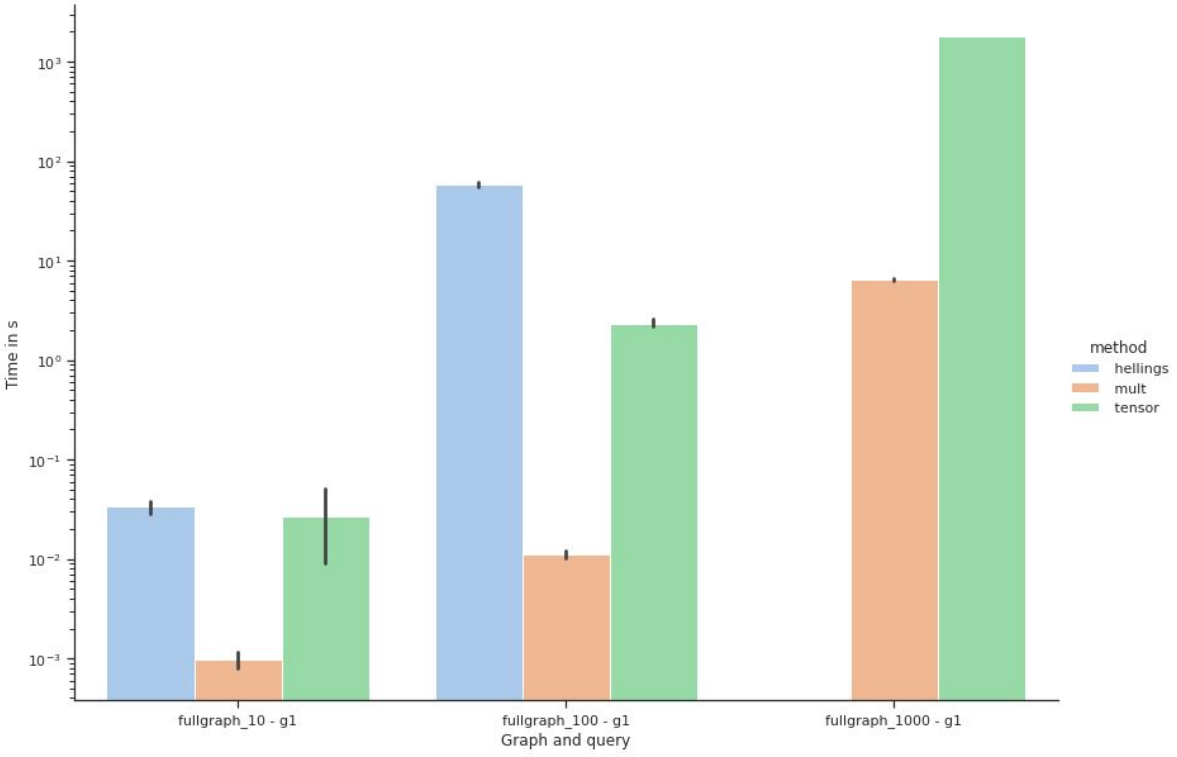
График для грамматики g2, все методы кроме алгоритма Хеллингса
(шкала времени обычная)



Для запроса *g2* (с регулярным выражением) нет очевидного улучшения во времени работы при подаче грамматики без преобразования в ОНФХ на вход тензорного алгоритма, а на некоторых графах метод работал даже незначительно дольше без этого преобразования. Причина этого - особенность реализации преобразования регулярных выражений в правых частях к КС грамматике (построение новых продукций грамматики по конечному автомату) приводит к образованию продукций такого вида: S → [sym]S. Даже несмотря на минимизацию автомата средствами *pyformlang*, грамматика разрастается и преимущества тензорного алгоритма, строящего рекурсивные автоматы, нивелируются. В будущем это можно было бы исправить, подавая на вход тензорному алгоритму уже построенный по входу (без промежуточного преобразования в контекстно-свободную грамматику) рекурсивный автомат.

Результаты на FullGraph

(шкала логарифмическая)



На этих данных алгоритм с перемножением матриц работал в среднем на два порядка быстрее двух других алгоритмов, поэтому и выбрана логарифмическая шкала. При этом алгоритм Хеллингса превысил ограничение в 30 минут на итерацию уже на *fullgraph_1000*, а тензорный алгоритм сработал немного лучше (около 15 минут на итерацию).

Выводы

1. Интересно, что алгоритм с произведением матриц оставался лидером на протяжении всего анализа, как на синтетических данных для худших случаев, так и на *MemoryAliases*, хоть тензорный алгоритм и алгоритм Хеллингса вели себя по-разному на разных наборах данных. Такой отрыв может быть обоснован тем, что

матричное произведение библиотеки *GraphBlas* реализовано очень эффективно для работы на современных ПК.

2. Также результаты показали, что подача на вход тензорному алгоритму грамматики без преобразования в ОНФХ уменьшает время работы алгоритма, но не настолько чтобы “догнать” алгоритм с произведением матриц. Анализ подтвердил недостатки в текущей реализации преобразования КС грамматик с регулярными выражениями в теле и показал перспективность использования прямого преобразования таких грамматик в рекурсивные автоматы.