# Отчет об экспериментальном сравнении реализаций

Автор:

Шапошников Алексей, СПБГУ

2 ноября 2020 г

## Экспериментальный анализ производительности

## КС-достижимости

При проведении эксперимента использовался компьютер с Windows 10 -> WSL-Ubuntu 20.04; Intel Core i5-4690 CPU @ 3.50 GHz; 16 gb DDR4 RAM.

Для эксперимента использовались данные из датасета DataForFLCourse: из директорий FullGraph, MemoryAliases, WorstCase выгружены все графы в отдельную директорию, из которой запускались алгоритмы. Каждому графу сопоставлялась одна из 4-х грамматик (g1-g4), среди которых были две простые с эпсилон продукциями, одна с большим количеством продукций и одна с регулярными выражениями в теле продукции.

Отмечу, что идея ожидаемой структуры проведения эксперимента и записи результатов на практике оказалась неподходящей. Будь это отчетом для научно-исследовательской работы, безусловно, пришлось бы переделывать, а в качестве задачи для курса по теории формальных языков эти ошибки дали очень полезный опыт проведения эксперимента, который, безусловно, пригодится в дальнейших замерах. Но и на таких

данных тоже можно сделать определенные выводы. А в частности об ошибках проведения эксперимента:

- Подразумевалось, что файлы будут отсортированы по размеру, чтобы получить как можно больше данных, но метод listdir выдавал их в ином порядке, в результате чего прогоны, которые должны были “крутиться” в течение, например, ночи, зависали над большим графом и не получалось собрать консистентные данные для анализа. В итоге, я делал несколько запусков скрипта для замеров, удаляя из директории графы, на которых алгоритмы работали более 20 минут.
- Структура скрипта замеров подразумевает, что все 4 алгоритма выполнятся, а затем данные будут собраны и представлены. Из-за этого не получилось показать, на каких графах какие-то алгоритмы отработали за адекватное время, а другие - вышли за границы. Безусловно, сильное упущение.
- Данные замеров времени преобразовывались сразу в графики. Это важное замечание, потому что, записав данные в том числе и в отдельные файлы, удалось бы избежать упущений таких, как некорректный выбор способа представления данных; вдобавок, можно было бы произвести сравнения другого вида (как пример, время работа определенного алгоритма на одном графе и разных грамматиках и т.д.)
- Неправильное представление данных в виде графиков с помощью библиотеки matplotlib - использовался метод boxplot для показания разброса по времени для каждого алгоритма по сравнению с другими. Из-за того, что один из алгоритмов

выполнялся долго, вертикальная ось затраченного времени сильно разрасталась, в итоге “ящики с усами” других алгоритмов значительно сжималось, графики становились плохо воспринимаемы для глаза.

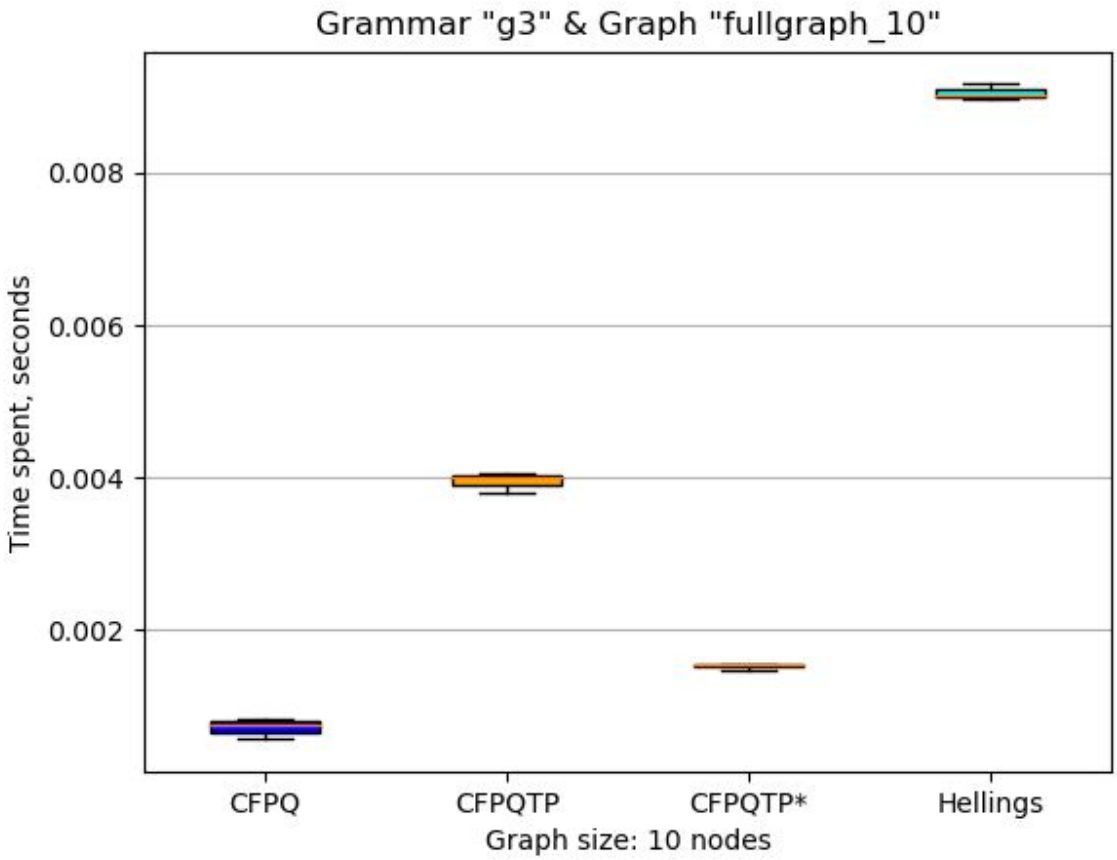
Производилось 5 итераций запуска алгоритмов на каждой паре граф-грамматика (также отмечу, что для диаграммы размаха более показательно было бы сделать больше итераций, но запуск алгоритмов занимал слишком много времени - стоило бы подумать о другом способе отображения данных). Вверху каждого графика подписаны граф и грамматика, на которых исполнялись алгоритмы; снизу - кол-во вершин в графе; вертикальная ось показывает затраченное алгоритмами время в секундах (стоило бы сделать логарифмическую шкалу); горизонтальная шкала - исполняемый алгоритм, а конкретно:

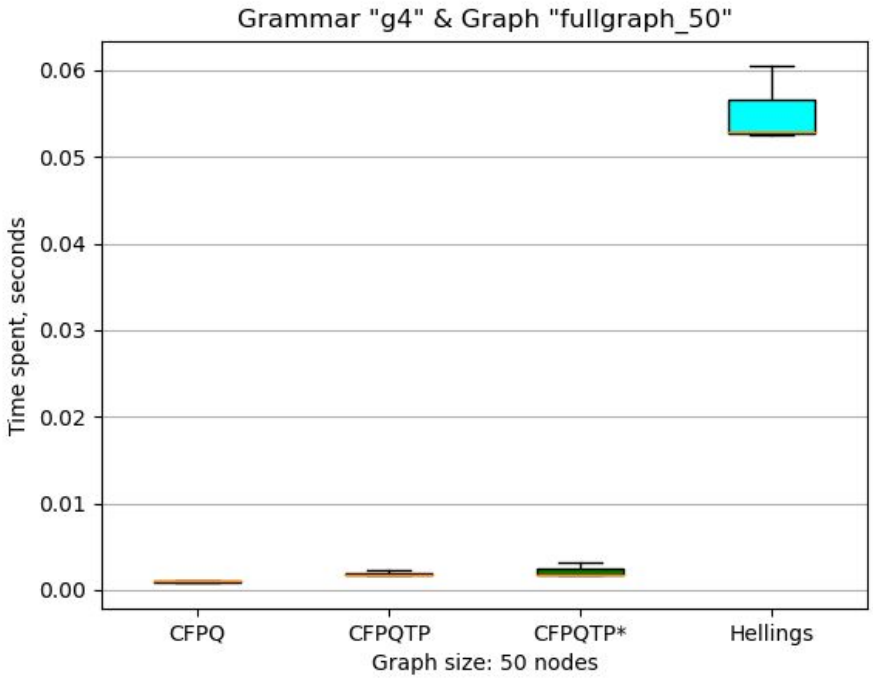
- CFPQ - достижимость через произведения булевых матриц, на вход - грамматика в ОНФХ.
- CFPQTP - достижимость через тензорное произведение матриц (включает преобразование поданной на вход грамматики в рекурсивный автомат) - грамматика в начальном виде.
- CFPQTP* - так же достижимость через тензорное произведение матриц, но на вход - грамматика в ОНФХ.
- Hellings - алгоритм Хеллингса - грамматика в начальном виде

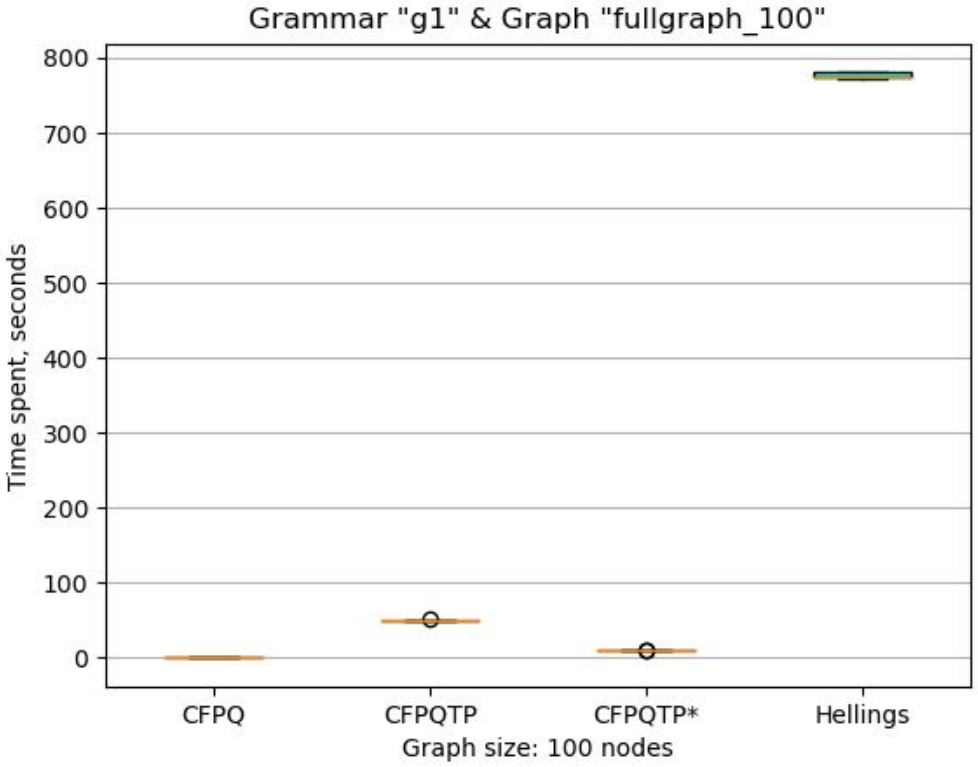
Стоит обратить внимание на то, что приведение грамматики к ОНФХ виду не измерялось, а для алгоритма CFPQTP в затраченное время включается преобразование грамматики в рекурсивный автомат (можно было бы также провести эксперимент по тому, насколько быстрее было бы приведение исходных данных сразу в рекурсивный автомат и подачи его на вход CFPQTP, так как таким образом вышло бы меньше продукций).

Все графики замеров выложены на google drive. Далее приведены некоторые из них:

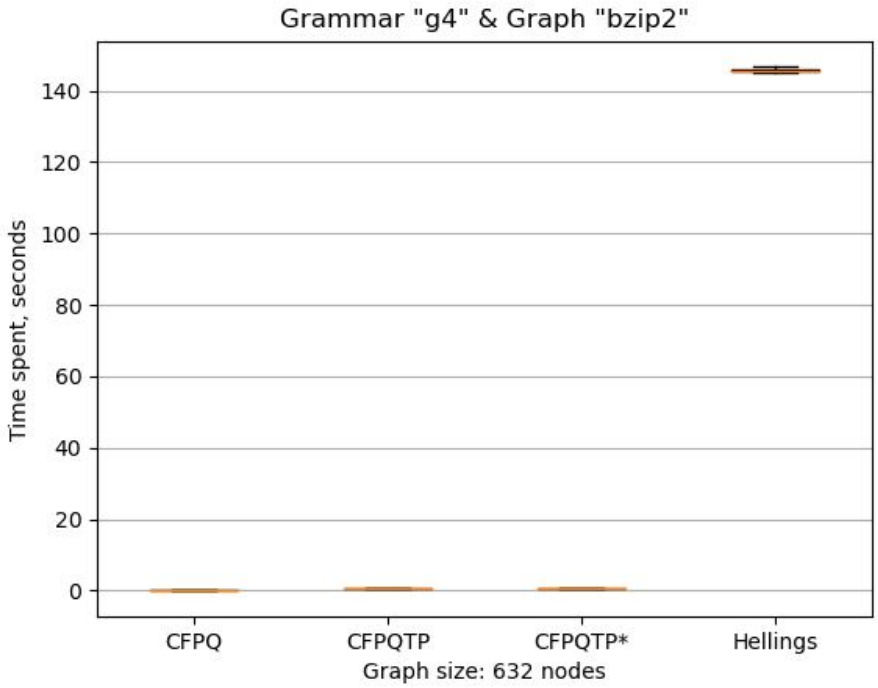
**FullGraph**

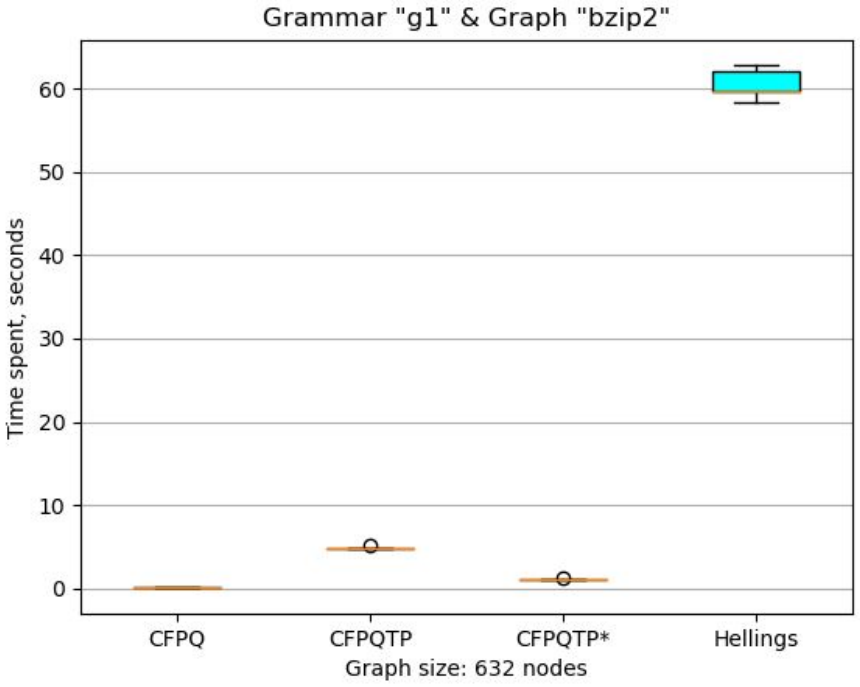


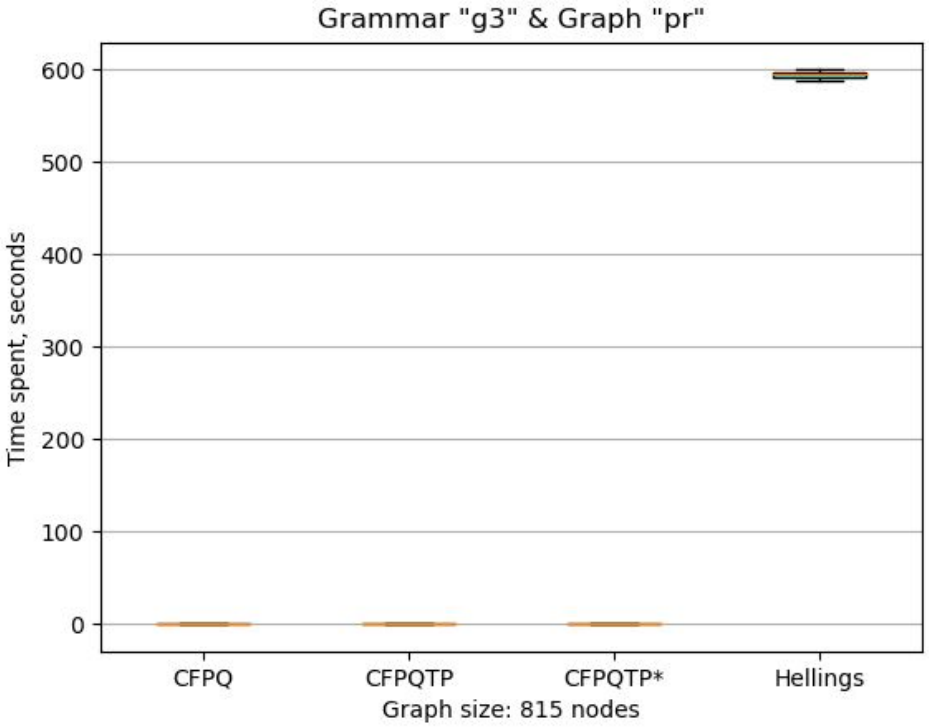

Grammar "g4" & Graph "fullgraph_50"
Time spent, seconds
0.06
0.05
0.04
0.03
0.02
0.01
0.00
CFPQ
CFPQTP
CFPQTP*
Hellings
Graph size: 50 nodes

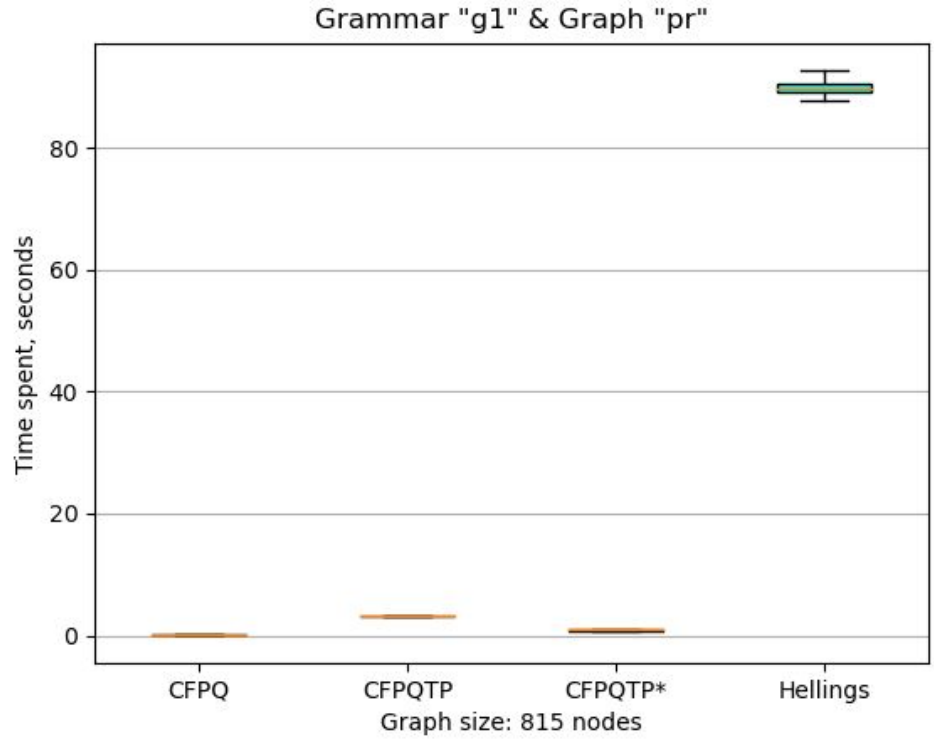

Grammar "g1" & Graph "fullgraph_100"
Time spent, seconds
800
700
600
500
400
300
200
100
0
CFPQ
CFPQTP
CFPQTP*
Hellings
Graph size: 100 nodes

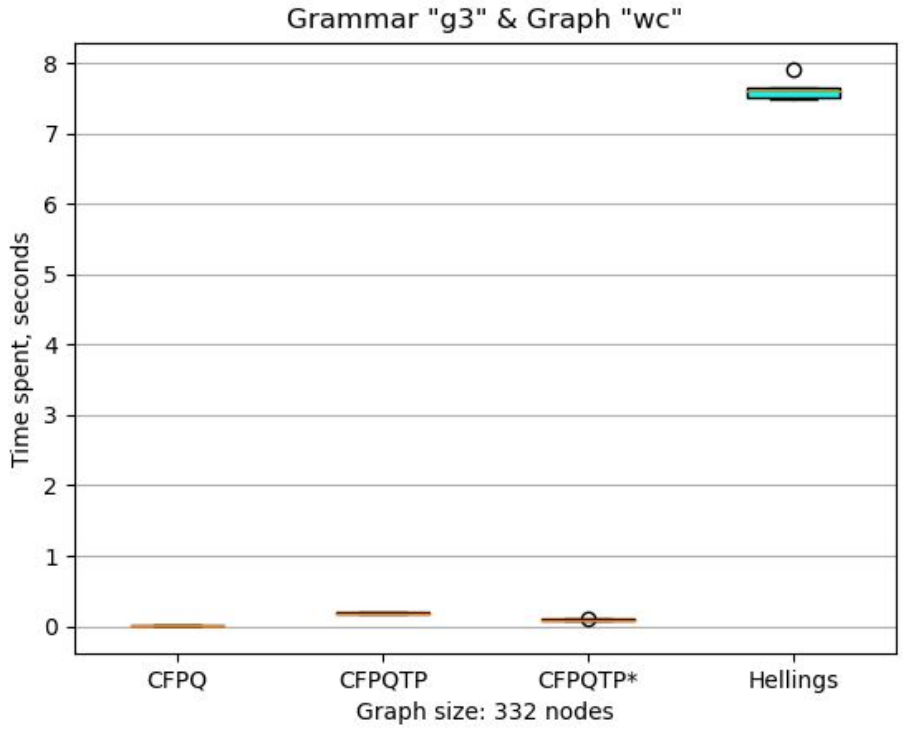
# MemoryAliases

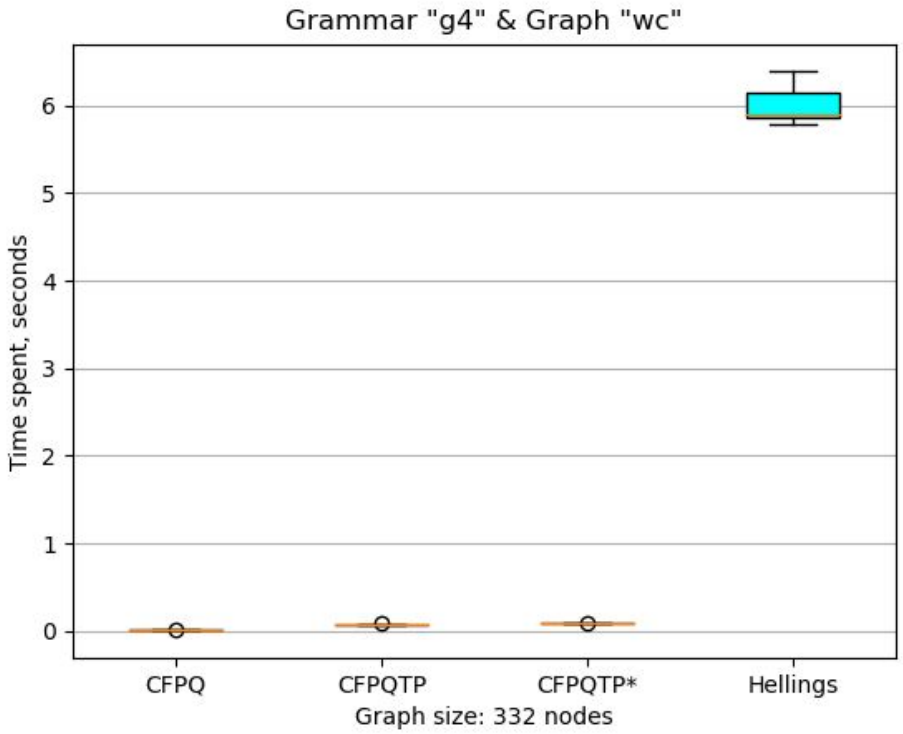


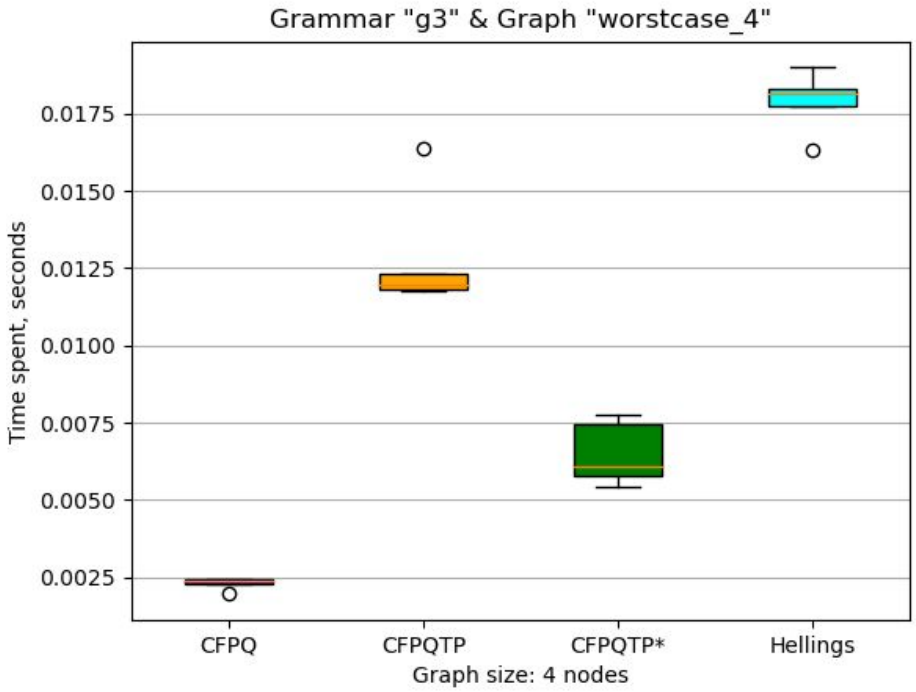

Grammar "g3" & Graph "pr"
Time spent, seconds
600
500
400
300
200
100
0
CFPQ
CFPQTP
CFPQTP*
Hellings
Graph size: 815 nodes

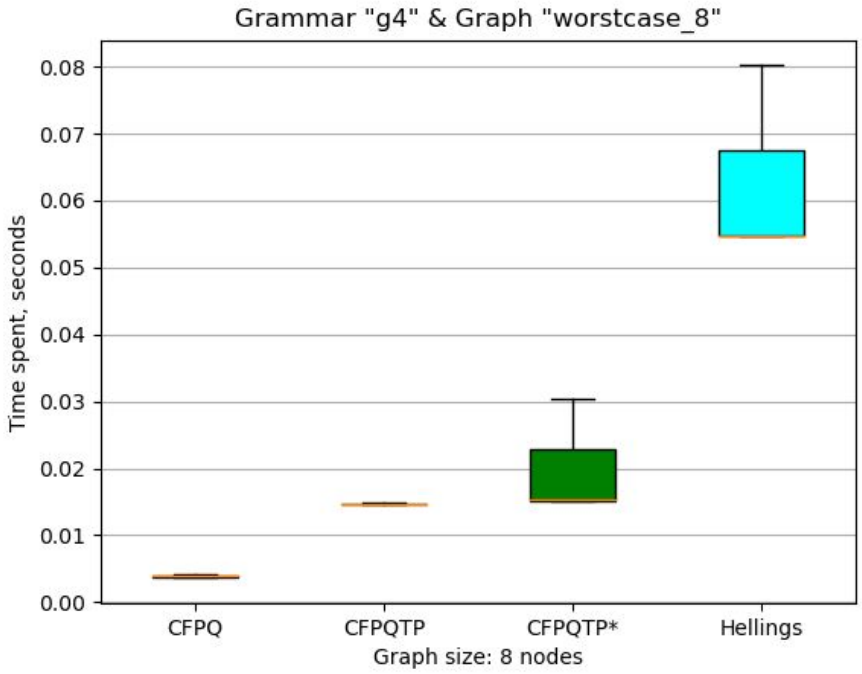

Grammar "g1" & Graph "pr"
Time spent, seconds
80
60
40
20
0
CFPQ
CFPQTP
CFPQTP*
Hellings
Graph size: 815 nodes

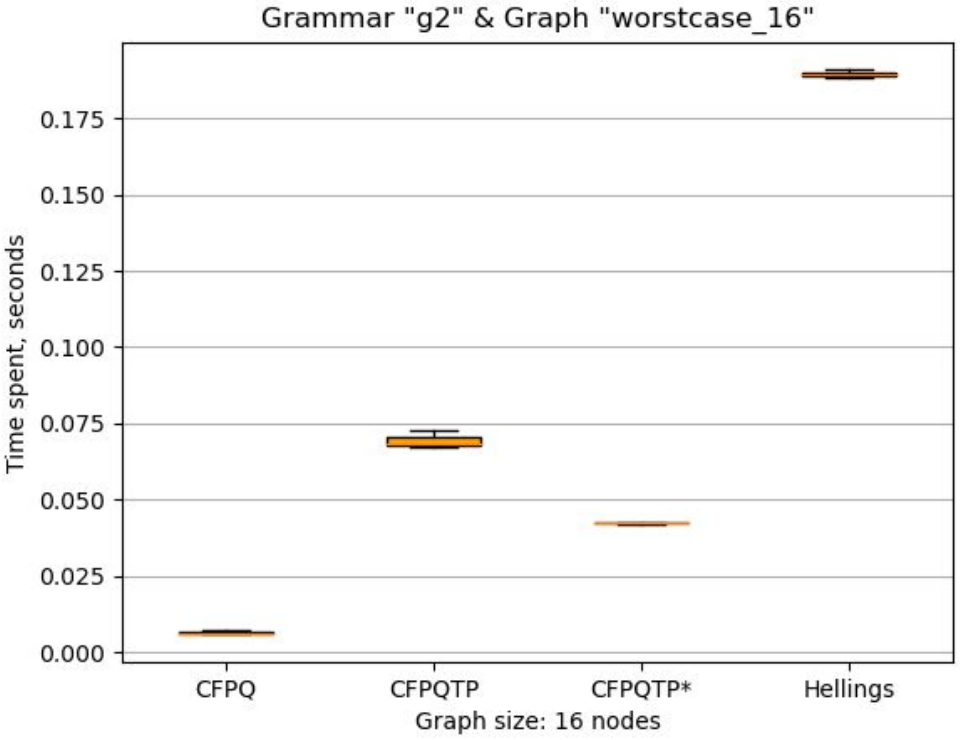

Grammar "g3" & Graph "wc"
Time spent, seconds
0
1
2
3
4
5
6
7
8
CFPQ
CFPQTP
CFPQTP*
Hellings
Graph size: 332 nodes

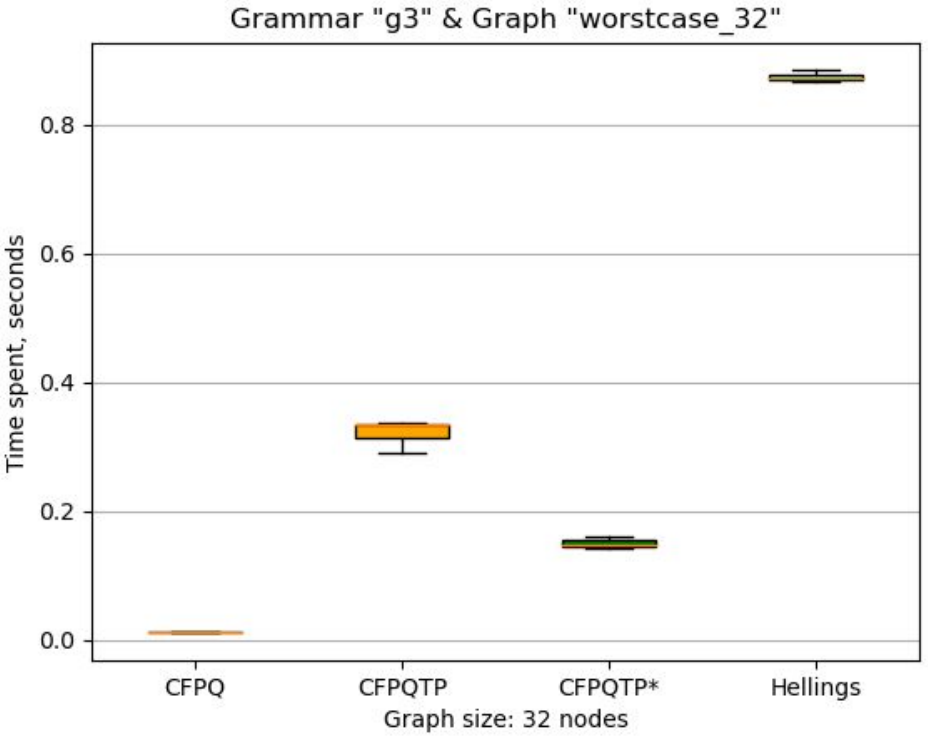

Grammar "g4" & Graph "wc"
Time spent, seconds
0
1
2
3
4
5
6
CFPQ
CFPQTP
CFPQTP*
Hellings
Graph size: 332 nodes

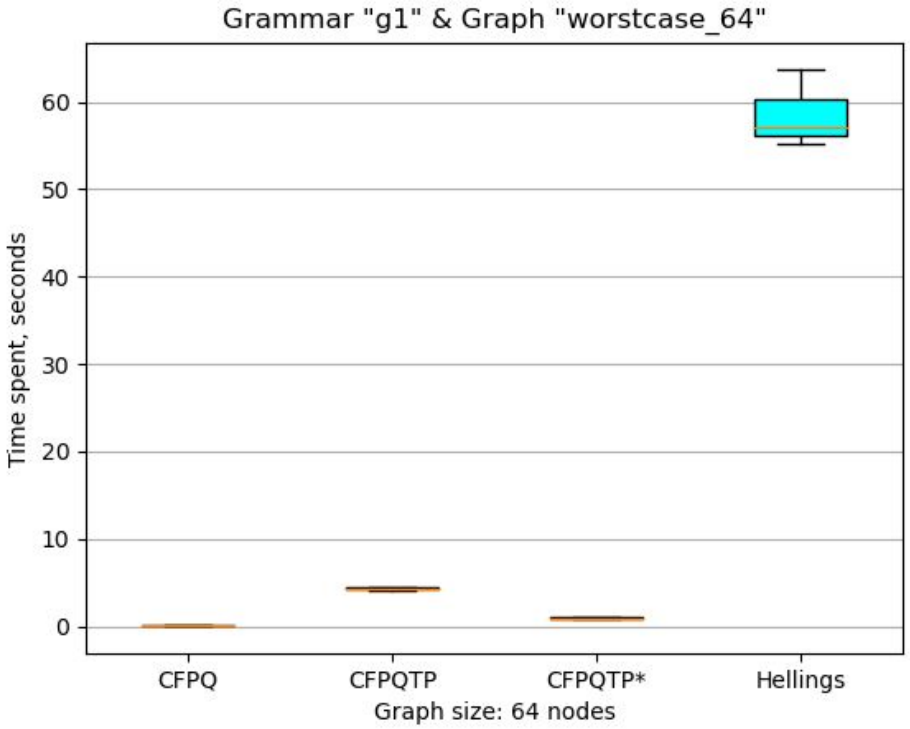
# WorstCase

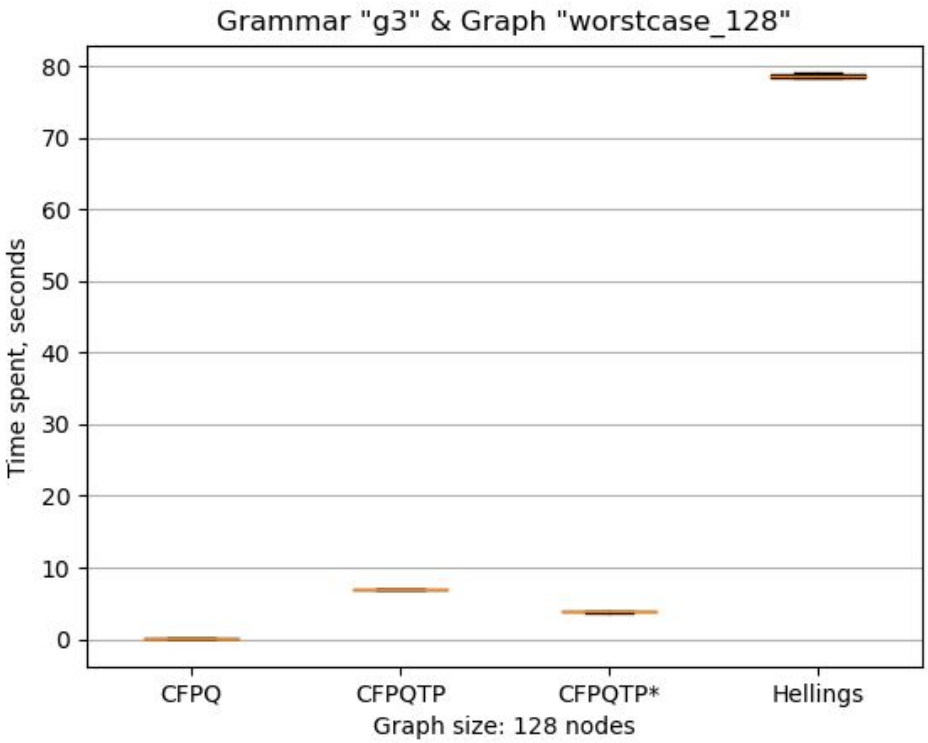

Grammar "g3" & Graph "worstcase_4"
Time spent, seconds
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
CFPQ
CFPQTP
CFPQTP*
Hellings
Graph size: 4 nodes


Grammar "g4" & Graph "worstcase_8"
Time spent, seconds
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
CFPQ
CFPQTP
CFPQTP*
Hellings
Graph size: 8 nodes


Grammar "g2" & Graph "worstcase_16"
Time spent, seconds
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
CFPQ
CFPQTP
CFPQTP*
Hellings
Graph size: 16 nodes


Grammar "g3" & Graph "worstcase_32"
Time spent, seconds
0.8
0.6
0.4
0.2
0.0
CFPQ
CFPQTP
CFPQTP*
Hellings
Graph size: 32 nodes


Grammar "g1" & Graph "worstcase_64"
Time spent, seconds
0
10
20
30
40
50
60
CFPQ
CFPQTP
CFPQTP*
Hellings
Graph size: 64 nodes


Grammar "g3" & Graph "worstcase_128"
Time spent, seconds
0
10
20
30
40
50
60
70
80
CFPQ
CFPQTP
CFPQTP*
Hellings
Graph size: 128 nodes

## Выводы по замерам

1. Первый напрашиваемый вывод - что алгоритм Хеллингса реализован не очень эффективно, хоть и проходит тесты. Он занимал в несколько десятков раз больше времени, чем остальные алгоритмы.
2. Самым эффективным оказался алгоритм, реализованный через произведение булевых матриц, что можно связать с оптимизированностью библиотеки GrapgBlas.
3. CFPQTP отрабатывал дольше вышеупомяного алгоритма в несколько раз, при этом CFPQTP*, который принимал на вход ОНФХ (без учета времени на преобразование!) отработал лучше CFPQTP на 3 из 4 грамматиках (и все равно хуже CFPQ) - кроме g4, в которой использовались регулярные выражения, что можно связать с разрастанием количеством продукций, если учесть последующее преобразование к РА. Опять-таки есть смысл проведения замеров для алгоритма с тензорным произведением, которому на вход подается сразу рекурсивный автомат, считанный из файла (что в теории уменьшит количество продукций и ускорит время выполнения).

## Выводы по проведению эксперимента

1. Стоит записывать данные замеров “на ходу”, а не только когда завершатся все рассматриваемые алгоритмы. Безусловно, стоит сохранять именно данные о времени в файлы (например, csv), чтобы потом подобрать подходящее представление в виде графиков или даже таблиц, в зависимости от разброса, разницы во времени и т.д.

2. Не стоит создавать универсальный скрипт для всех тестовых данных, потому что он может на каком-то этапе остановить замер. Хорошей идеей было бы написать shell-скрипт, который бы запускал замеры определенных тестовых данных, переставал работать по истечению временного лимита, переходил к следующему набору данных. Также хорошо изучить возможность фильтрации (без непосредственного перечисления названия в скрипте) файлов для последовательной подачи графов и грамматик на вычисление.
3. Следовало бы сделать шкалу времени логарифмической, рассмотреть возможность отображения затраченного времени в виде столбцов, а не “ящиков с усами”.
4. При планировании стоит выделить отдельное время на случай, если что-то пойдет не так и придется производить повторные замеры. Так вышло в моем случае, когда из-за неучтенной возможности неправильного эксперимента, данные были получены непосредственно перед дедлайном - и не было возможности повторить эксперимент в другом виде, зато из этого получился анализ над ошибками.
5. В дополнение к последнему пункту, можно было попробовать получить сначала “образец” замеров на нескольких тестовых данных, и, проанализировав их, “подогнать” проведение эксперимента ad hoc.