Отчёт о результатах измерения производительности программы

Цель работы

Провести замеры производительности реализаций различных алгоритмов подсчёта количества путей в данной графовой базе данных, отвечающих данной контекстно-свободной грамматике.

Основные сведения

Исходные данные: набор баз данных и запросов DataForFLCourse.

Оборудование:

- процессор: Intel(R) Core(TM) i3-8130U CPU @ 2.20GHz 2.21HGz
- оперативная память: 4793 МБ
- операционная система: Ubuntu 18.04.3 x64
- утилита для тестирования: pytest
    - основные сведения:
        - Python 3.8.2
        - pytest-6.0.1
        - py-1.9.0
        - pluggy-0.13.1
    - дополнительные плагины:
        - timeout-1.4.2
        - cov-2.10.0

Тестируемые алгоритмы:

- CFPQ (алгоритм Хеллингса)
- CFPQ_BM (алгоритм на булевых матрицах)
- CFPQ_TP (алгоритм на тензорном произведении матриц)
- CFPQ_TP_CNF (CFPQ_TP на грамматике в НФХ)

Результаты тестирования

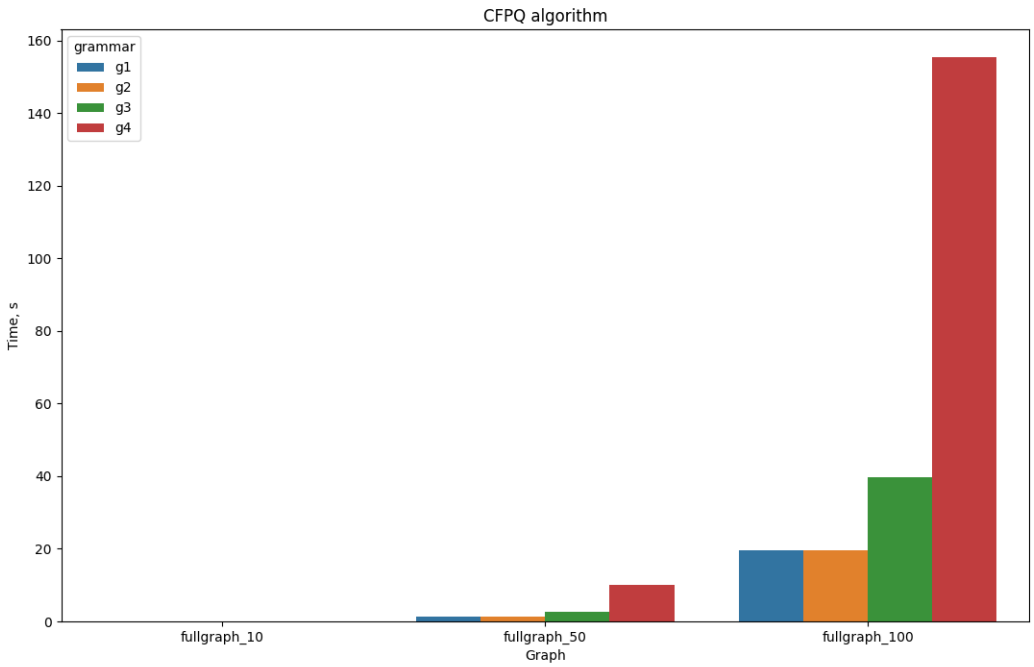
Все данные по измерениям производительности сохранены в файле Results.txt. В каждой строке находится одна запись в виде (алгоритм, группа, имя файла с графом, имя файла с грамматикой, время в секундах). Время указано с точностью 0,01 секунда.

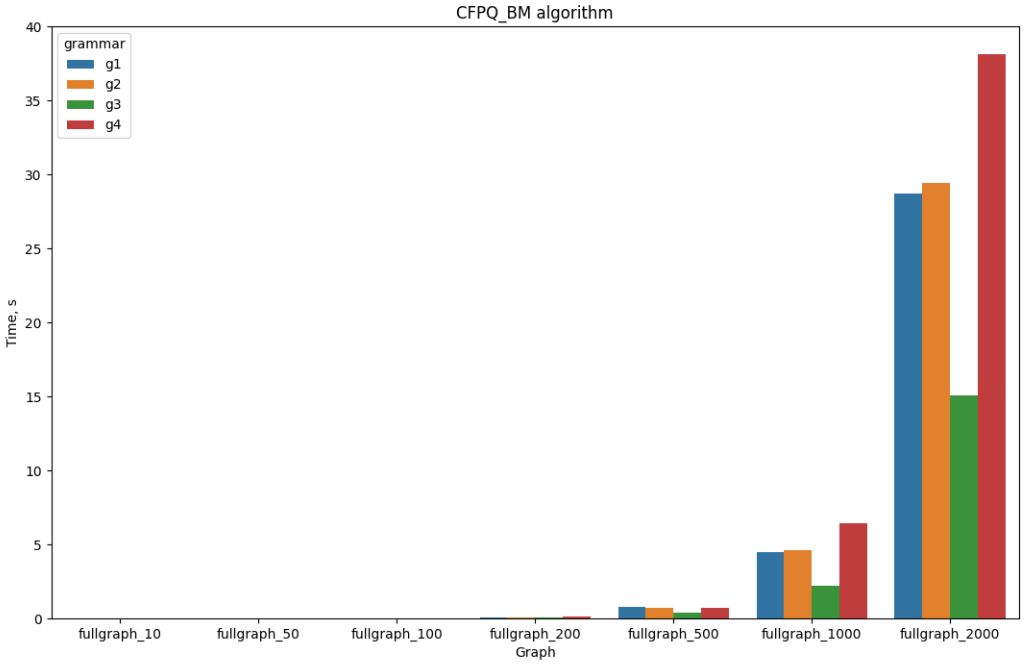
Ниже приведены графики результатов тестирования реализаций алгоритмов на различных базах данных и с различными запросами. Для удобства базы данных и соответствующие им запросы будут разделены на 3 группы: FullGraph, MemoryAliases и WorstCase.

Примечание: Для всех тестов был установлен timeout = 5 минут. Поэтому многие пары (база данных, запрос) не были протестированы. Например, для группы SparseGraph результатов тестирования нет вовсе.

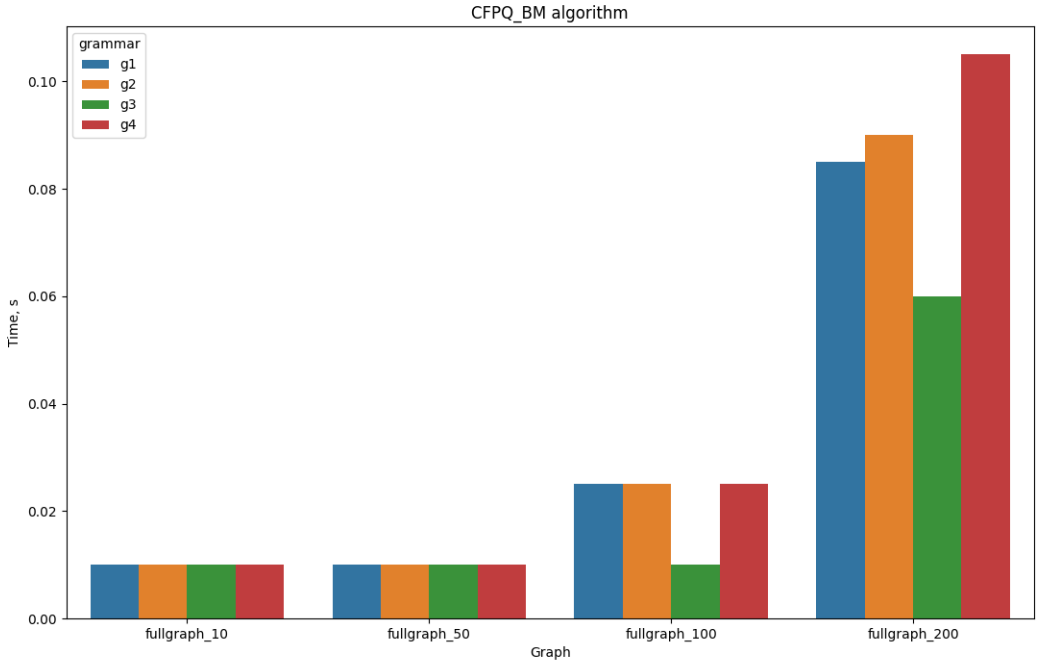
FullGraph

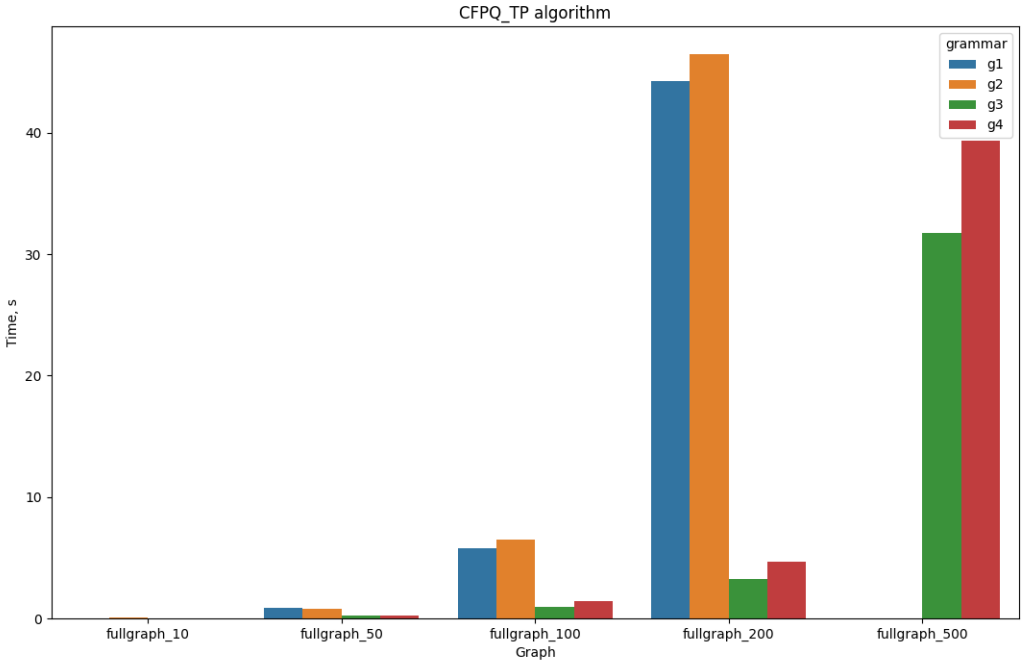
Результаты зависимости времени работы алгоритмов от пары (база данных, запрос):

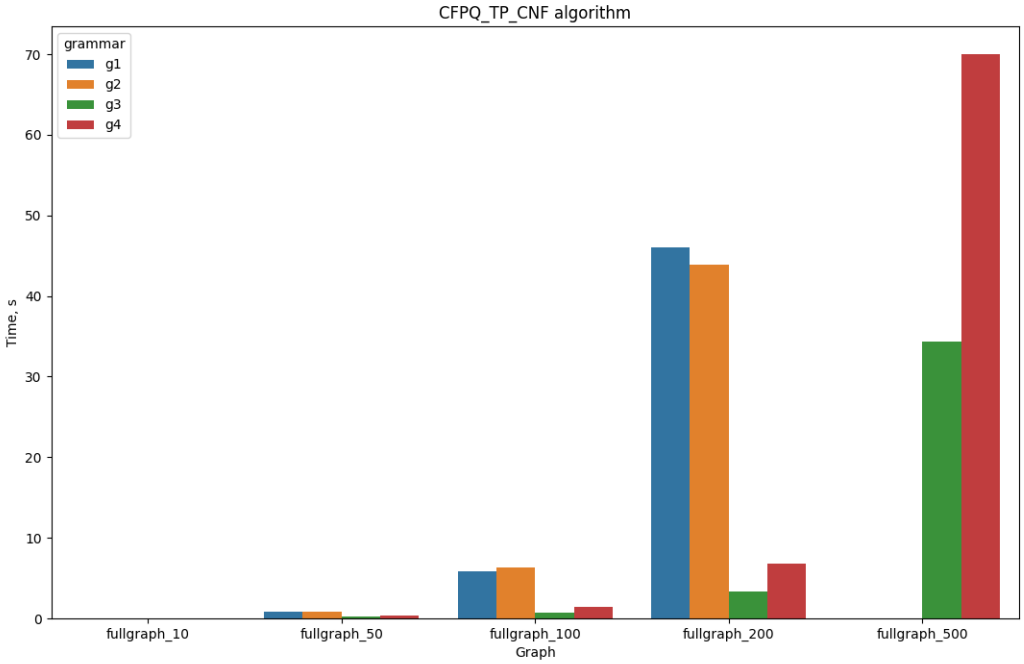




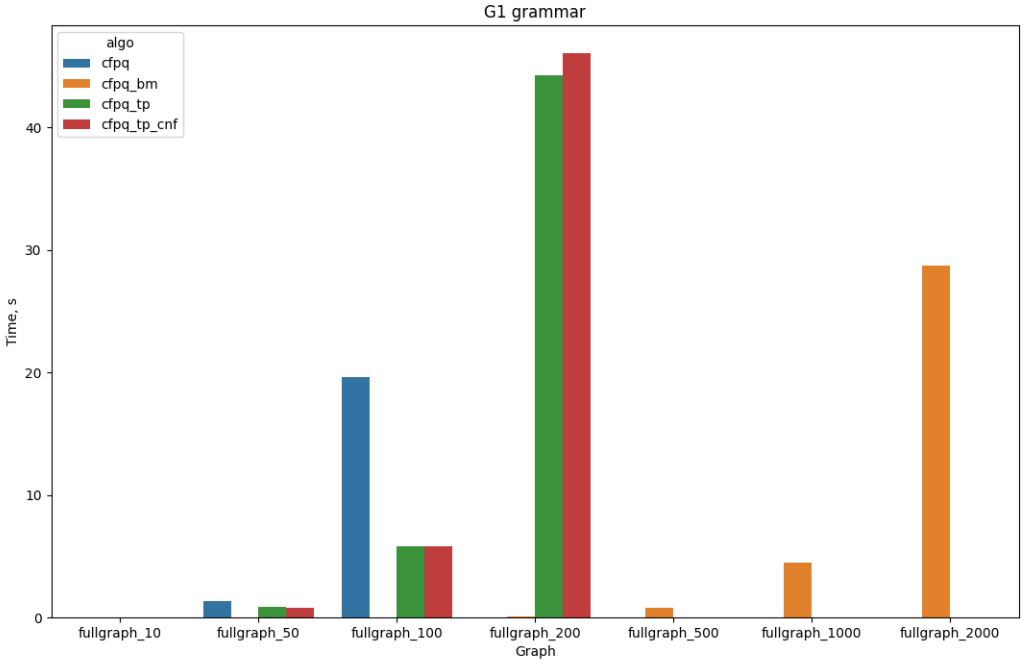
Для первых 4 баз данных для CFPQ_BM график будет таким:

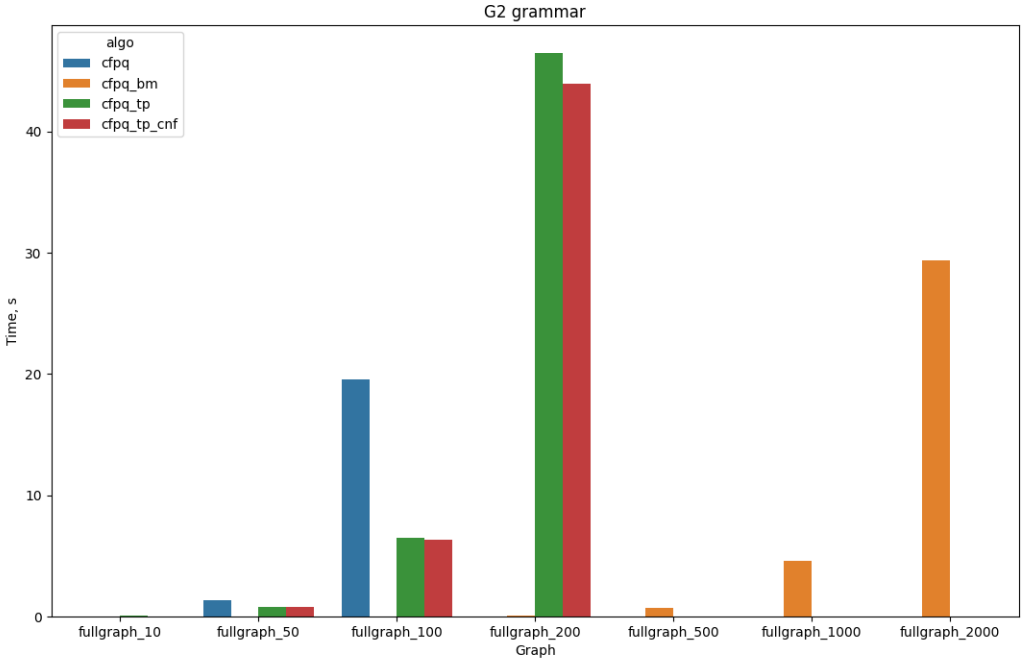


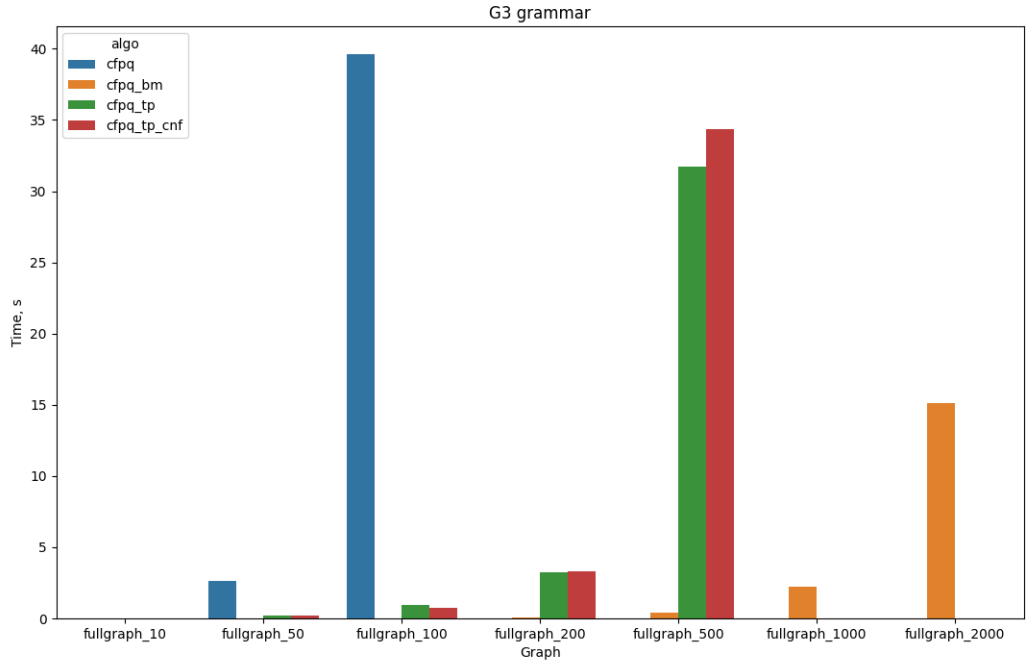

CFPQ_TP algorithm
grammar
g1
g2
g3
g4
Time, s
0
10
20
30
40
fullgraph_10
fullgraph_50
fullgraph_100
fullgraph_200
fullgraph_500
Graph

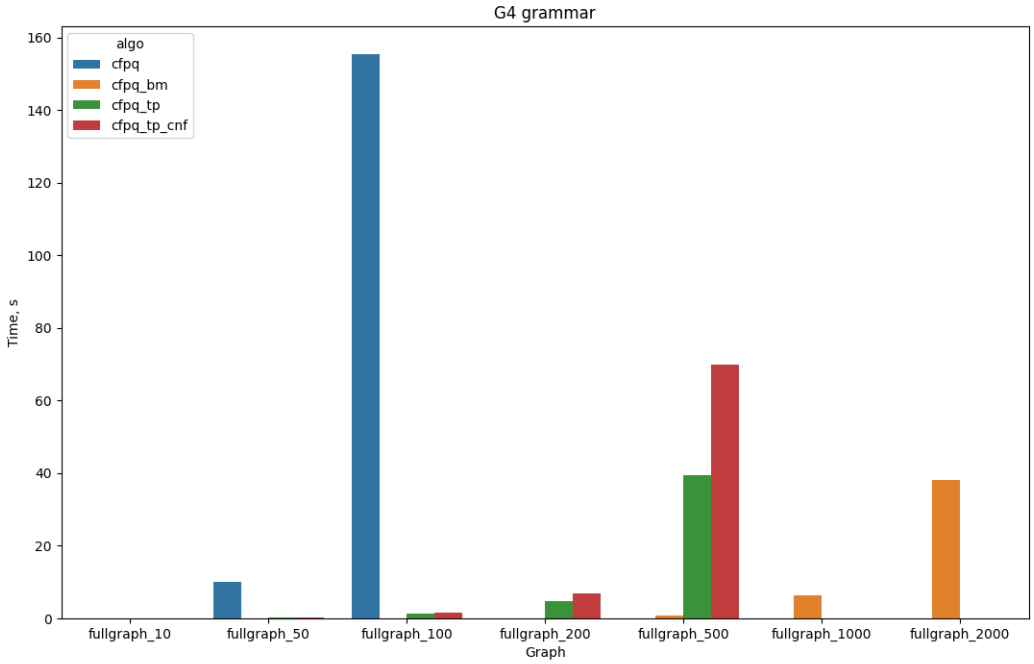

CFPQ_TP_CNF algorithm
grammar
g1
g2
g3
g4
Time, s
0
10
20
30
40
50
60
70
fullgraph_10
fullgraph_50
fullgraph_100
fullgraph_200
fullgraph_500
Graph

Результаты зависимости времени работы алгоритмов от пары (база данных, алгоритм):



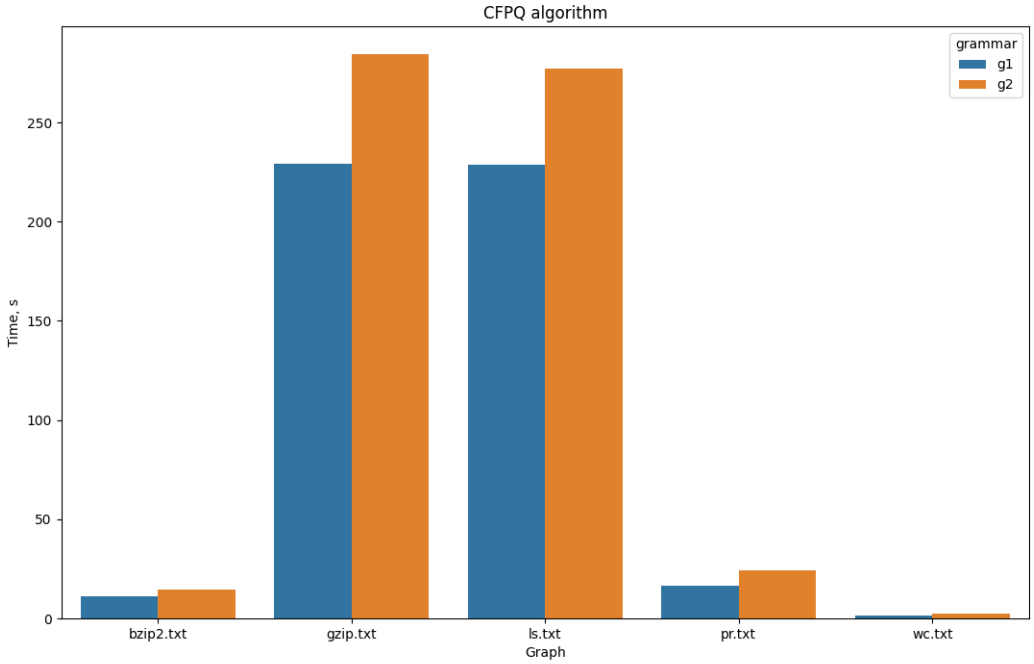


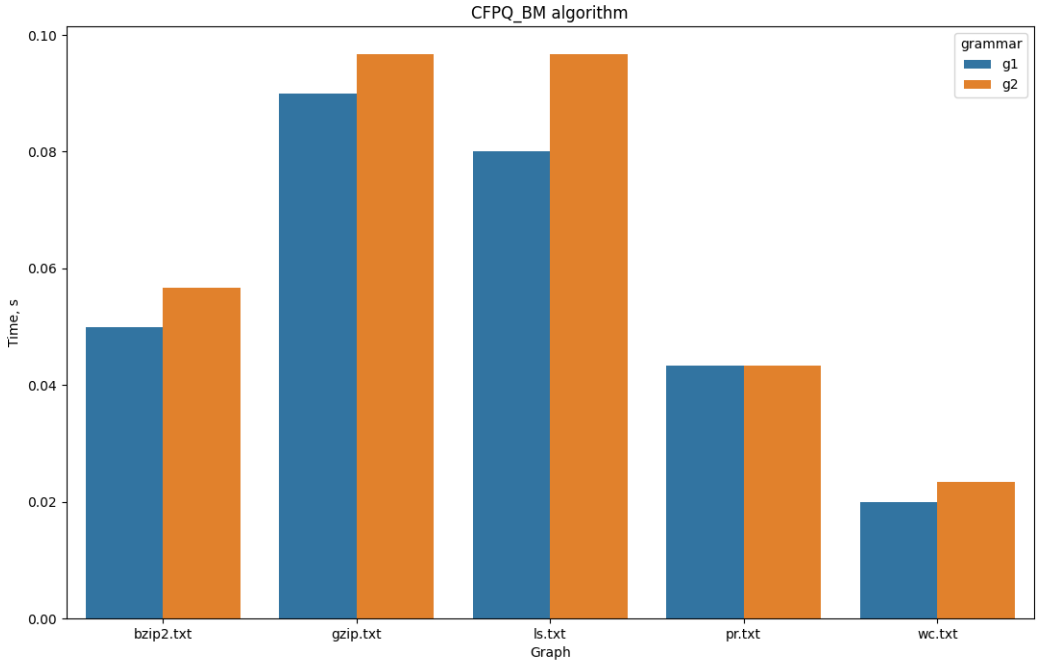

G3 grammar
algo
cfpq
cfpq_bm
cfpq_tp
cfpq_tp_cnf
Time, s
0
5
10
15
20
25
30
35
40
fullgraph_10
fullgraph_50
fullgraph_100
fullgraph_200
fullgraph_500
fullgraph_1000
fullgraph_2000
Graph

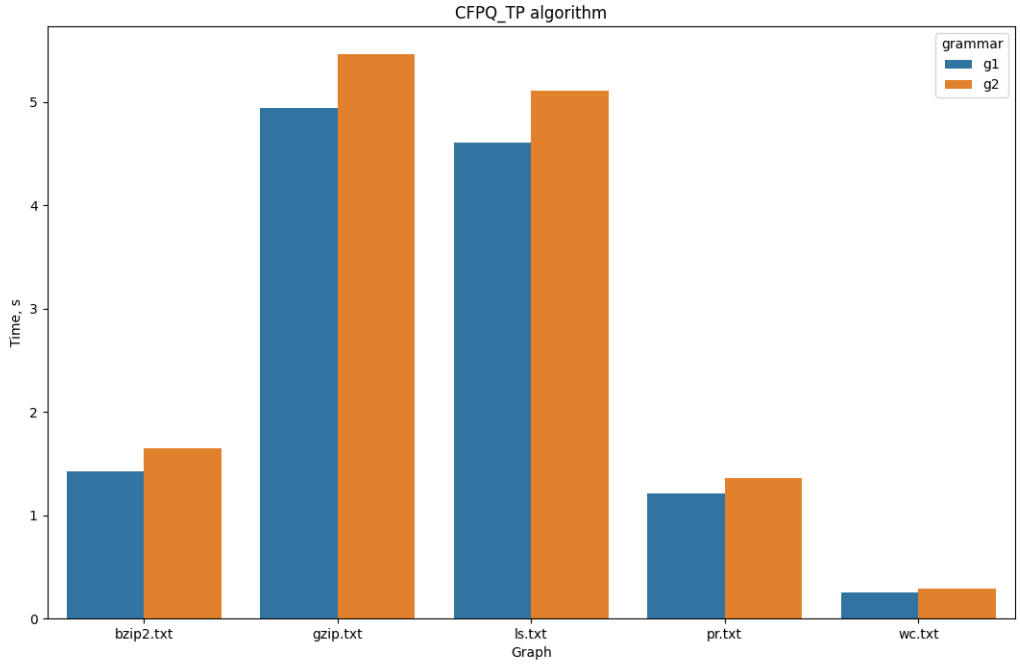

G4 grammar
algo
cfpq
cfpq_bm
cfpq_tp
cfpq_tp_cnf
Time, s
0
20
40
60
80
100
120
140
160
fullgraph_10
fullgraph_50
fullgraph_100
fullgraph_200
fullgraph_500
fullgraph_1000
fullgraph_2000
Graph

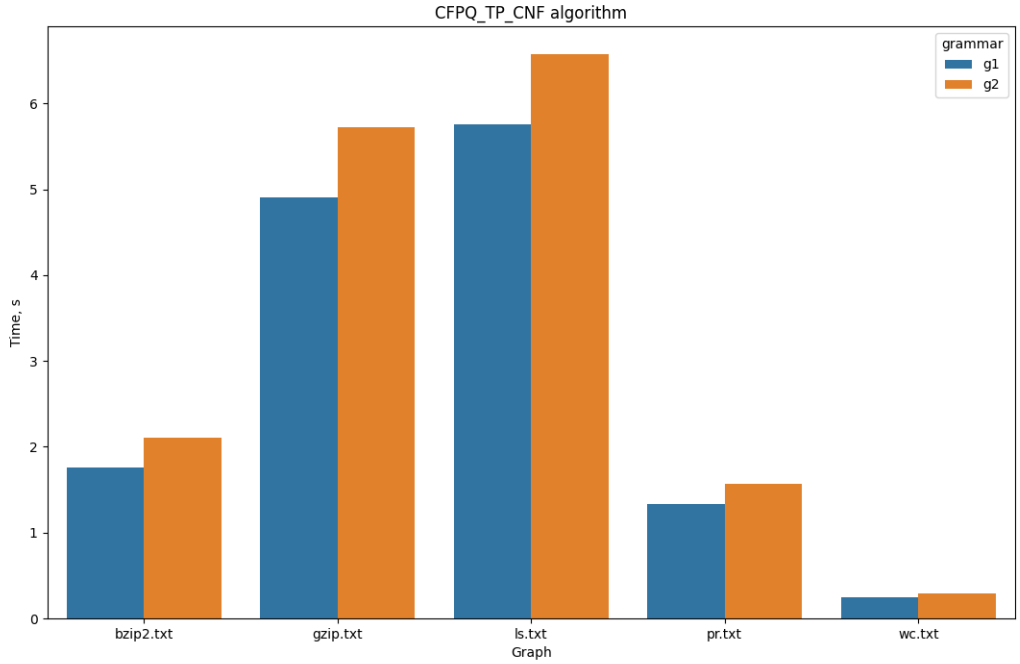
MemoryAliases

Результаты зависимости времени работы алгоритмов от пары (база данных, запрос):

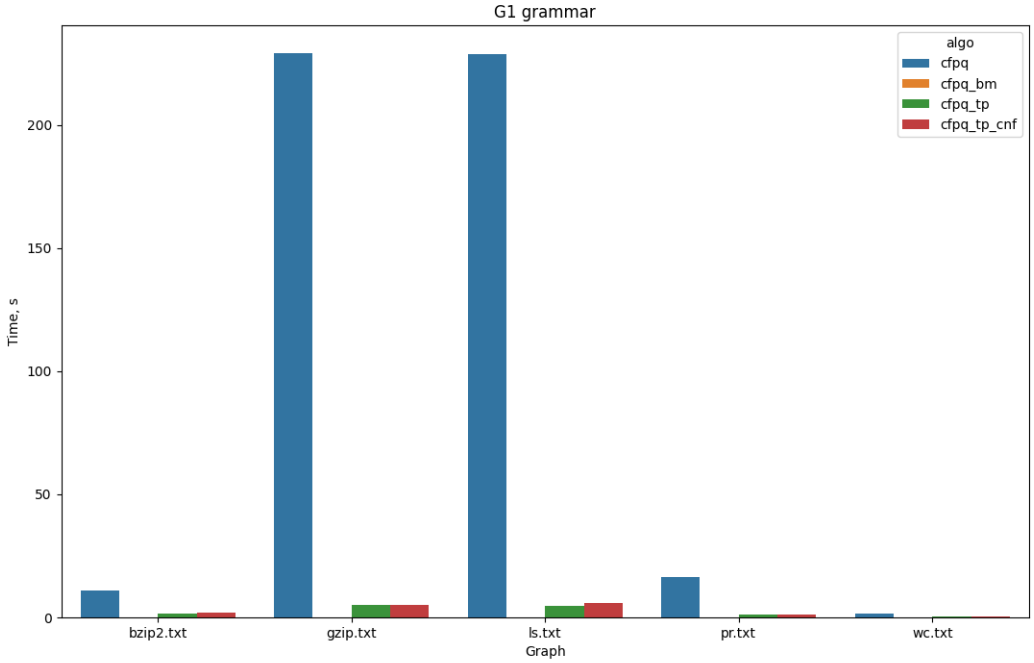


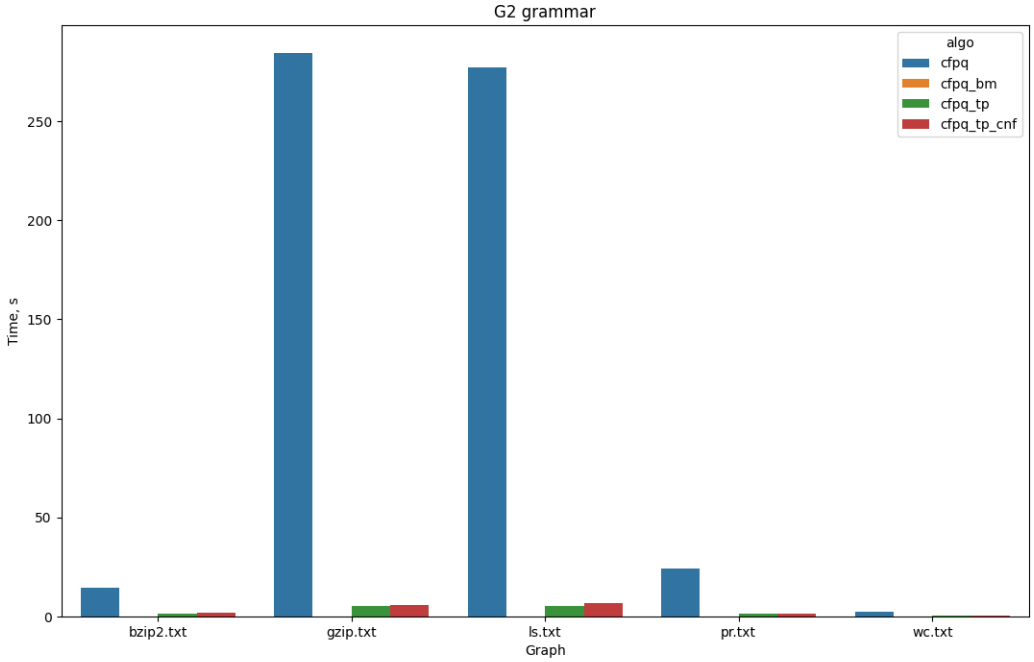



CFPQ_TP algorithm
grammar
g1
g2
Time, s
0
1
2
3
4
5
bzip2.txt
gzip.txt
ls.txt
pr.txt
wc.txt
Graph

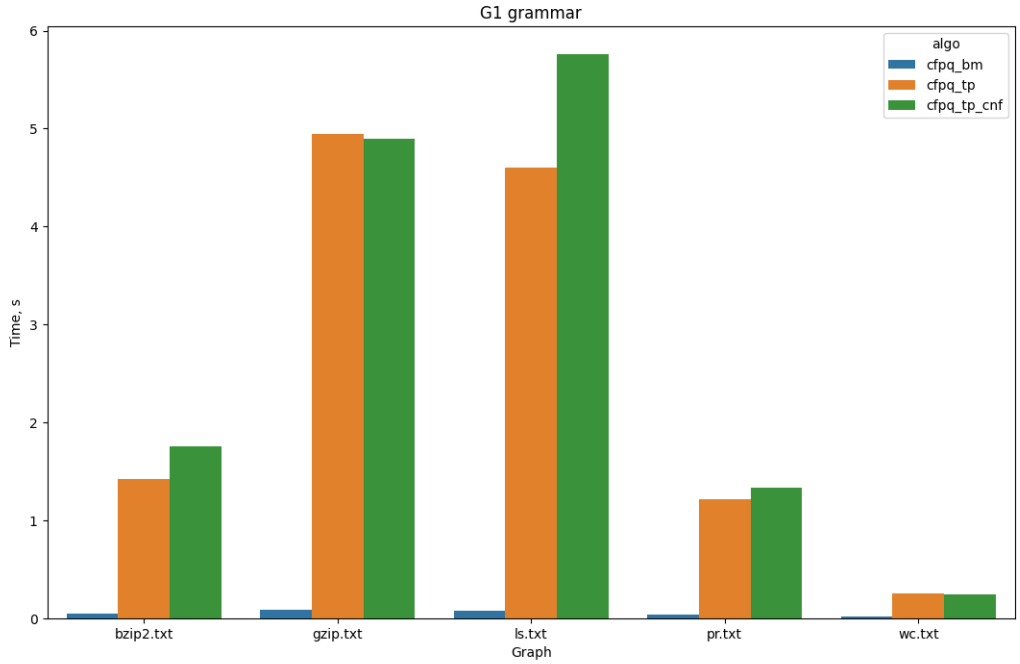

CFPQ_TP_CNF algorithm
grammar
g1
g2
Time, s
0
1
2
3
4
5
6
bzip2.txt
gzip.txt
ls.txt
pr.txt
wc.txt
Graph

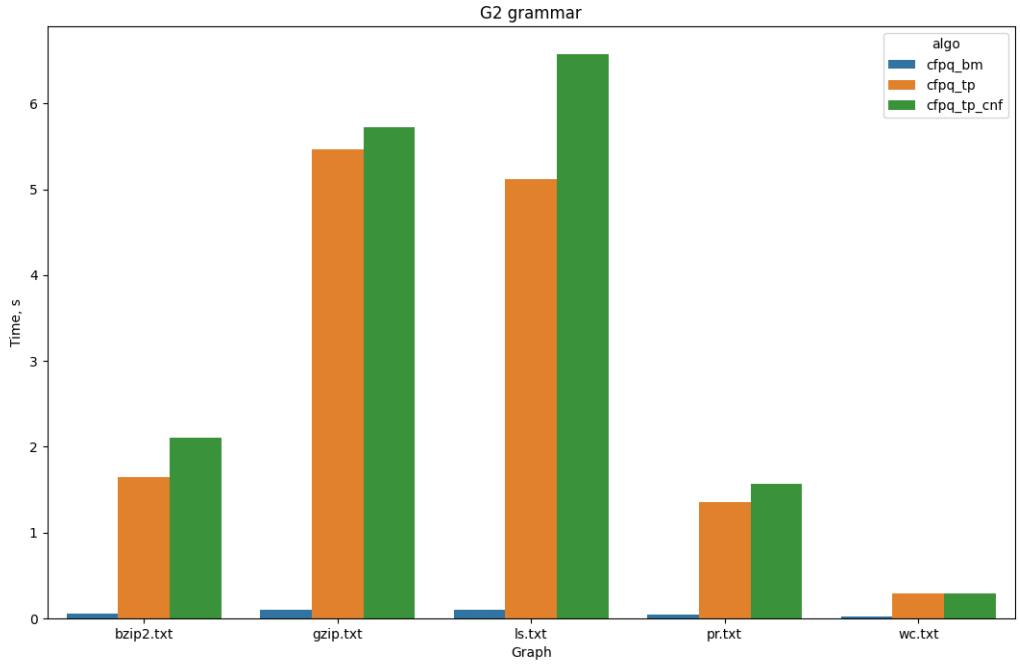
Результаты зависимости времени работы алгоритмов от пары (база данных, алгоритм):





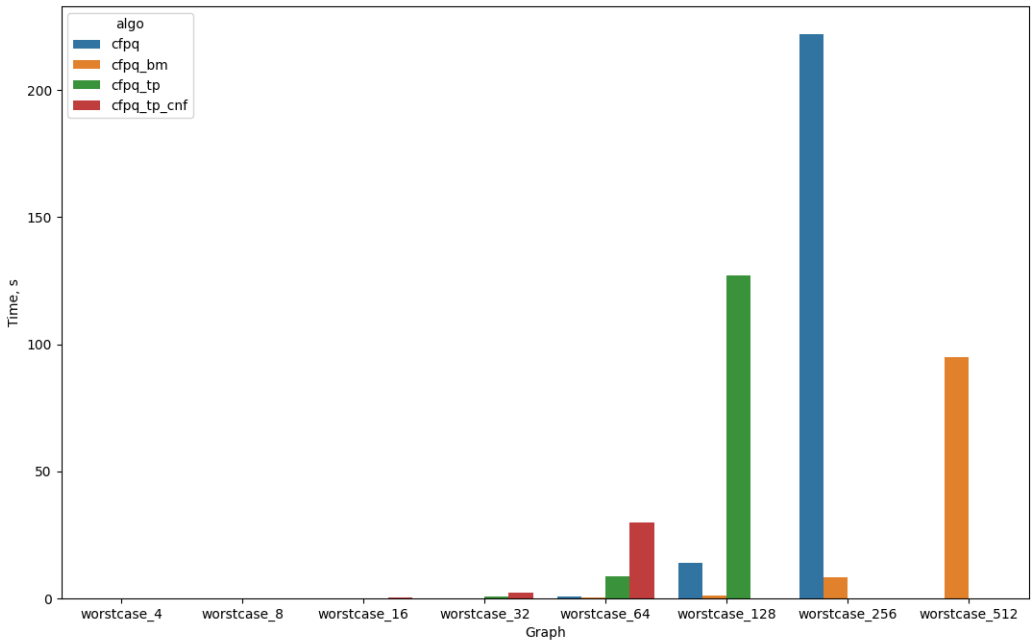
Те же результаты, но без алгоритма CFPQ:



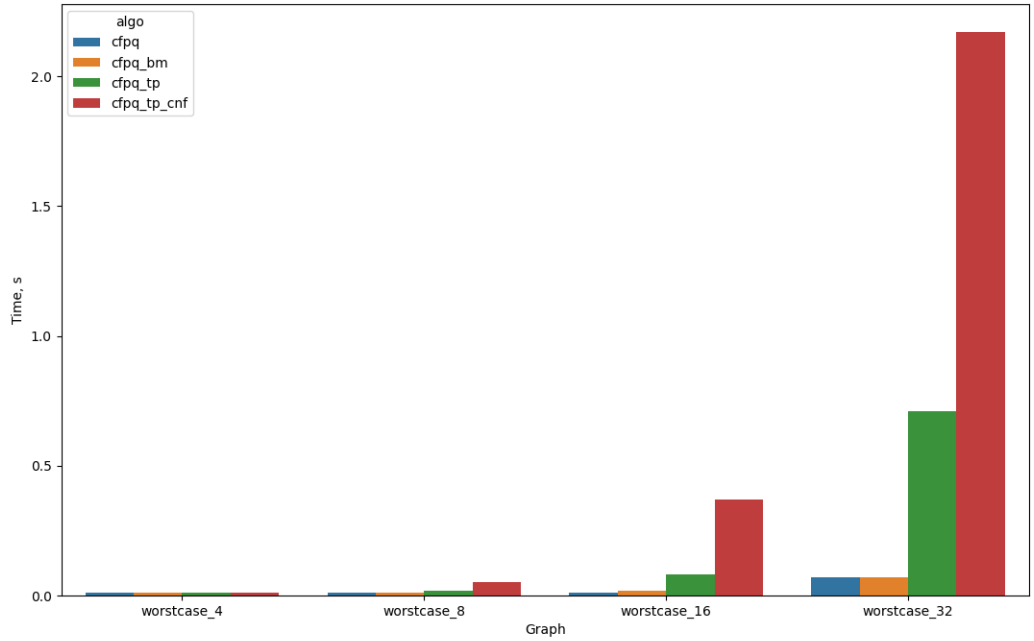


WorstCase

Результаты зависимости времени работы алгоритмов от пары (база данных, алгоритм):



Те же результаты, но для первых 4 баз данных:



Выводы:

Мною были проанализированы полученные данные и отмечены следующие наблюдения:

- Реализация CFPQ_BM работает значительно быстрее других реализаций на всех парах (база данных, запрос)
- Реализация CFPQ для групп FullGraph и MemoryAliases работает значительно медленнее других реализаций, но для группы WorstCase – уступает только CFPQ_BM
- Реализация CFPQ_TP_CNF работает не лучше, чем CFPQ_TP. Стоит отметить, что для группы FullGraph эти 2 реализации почти не отличаются по времени работы друг от друга, для группы MemoryAliases CFPQ_TP работает быстрее CFPQ_TP_CNF, но незначительно, а для группы WorstCase CFPQ_TP работает значительно быстрее CFPQ_TP_CNF