1. Эксперимент

В данном разделе приводятся условия экспериментов, методология проведения измерений, а также анализ полученных результатов и профилирование кода.

1.1. Условия и методика эксперимента

Эксперимент проводился на ноутбуке со следующими характеристиками:

- **Процессор:** AMD Ryzen 5 7640HS.
- **Оперативная память:** 16 ГБ LPDDR5.
- **Операционная система:** Arch Linux (версия ядра 7.1.8-arch1-3, AMD64).

Для автоматизации запусков алгоритма и проведения измерений использовался бенчмарк Ильхома Комбаева CFG_bench[1]. Модификации для проведения эксперимента сохранены в форке CFG_bench[2] в ветке profiling.

Для проведения оценки производительности алгоритмов контекстно-свободной достижимости были выбраны следующие грамматики и соответствующие им графы с сайта[3].

1. Nested Parentheses:
   - eclass ($|V| = 239\,111$, $|E| = 360\,248$),
   - go_h ($|V| = 45\,007$, $|E| = 490\,109$),
   - go ($|V| = 582\,929$, $|E| = 1\,437\,437$).
2. Java Points-to:
   - avrora ($|V| = 24\,690$, $|E| = 25\,196$),
   - batik ($|V| = 60\,175$, $|E| = 63\,089$).

[1] https://github.com/homka122/CFG_bench (дата обращения: 28 августа 2026 г.)

[2] https://github.com/Zestria/CFG_bench (дата обращения: 29 августа 2026 г.)

[3] https://formallanguageconstrainedpathquerying.github.io/CFPQ_Data (дата обращения: 28 августа 2026 г.)

3. С Alias:
    - block ($|V| = 3\,423\,234$, $|E| = 2\,951\,393$),
    - init ($|V| = 2\,446\,224$, $|E| = 2\,112\,809$),
    - mm ($|V| = 2\,538\,243$, $|E| = 2\,191\,079$).

Использовалась следующая методика:

- Для большей точности практичности замеров в реальных условиях в качестве кандидатов в множество стартовых вершин S рассматриваются только те вершины $v \in V$, из которых достижима хотя бы одна вершина по заданной грамматике. Предварительный расчёт множества потенциальных источников производился с помощью алгоритма CFL_adv.
- Готовое множество рассматриваемых вершин случайно перемешивается. Для каждого фиксированного размера множества источников $|S| \in \{1, 10, 100\}$ формируется 30 независимых чанков, суммарно включающих $30 \cdot |S|$ повторяющихся стартовых вершин.
- Для снижения влияния окружающей среды каждый чанк вычисляется следующим образом:
    - Выполняется один прогревочный запуск, результаты которого не учитываются.
    - Выполняются три повторных запуска.
    - Выбирается минимальное время из трёх замеров.
- На основе полученного массива из 30 измерений времени выполнения считается среднее и стандартное отклонение.
- Поскольку алгоритм CFL_adv вычисляет достижимость между всеми парами вершин и не зависит от выбора S, для него проводится один прогревочный запуск и 10 повторных измерений, из которых выбирается минимальное время работы.

1.2. Результаты

Итоговые результаты замеров приведены в таблице 1. Замерить CFL_multsrc на грамматике Java Points-to не вышло. Выполнение заняло более 3 часов и отмечено как Out of time.

Таблица 1: Результаты измерений(в миллисекундах).

Graph	CFL_ADV	CFPQ_RSM			CFL_MULTSRC		
		1	10	100	1	10	100
block	3786	5.2 ± 5.6	10.4 ± 4.8	27.4 ± 9.0	1201 ± 837	2030 ± 778	3453 ± 884
init	2802	2.8 ± 2.1	10.6 ± 5.8	28.2 ± 7.1	538 ± 146	1481 ± 693	2501 ± 525
mm	3110	3.1 ± 3.3	10.5 ± 5.2	30.1 ± 8.7	583 ± 233	1469 ± 567	2672 ± 620
avrora	328	1362 ± 3415	5771 ± 5690	14428 ± 3410	Out of time	Out of time	Out of time
batik	556	1333 ± 2679	4945 ± 4231	14148 ± 4883	Out of time	Out of time	Out of time
eclass	39	2.3 ± 0.8	3.9 ± 0.9	5.8 ± 0.9	20.6 ± 0.5	20.9 ± 0.6	24.0 ± 1.5
go_h	44	3.2 ± 2.1	5.7 ± 13.9	30.7 ± 31.7	22.7 ± 6.1	16.4 ± 23.6	67.3 ± 45.2
go	201	17.8 ± 14.3	44.7 ± 20.9	127.8 ± 116.8	28.0 ± 1.0	35.4 ± 12.6	179.7 ± 255.2

1.3. Обсуждение результатов

На основе полученных результатов можно сделать следующие выводы:

1. При увеличении количества стартовых вершин от 1 до 100 наблюдается сублинейный рост алгоритма CFPQ_RSM.
2. Алгоритм CFPQ_RSM демонстрирует существенное преимущество по времени выполнения над CFL_MULTSRC.
3. На грамматиках C Alias и Nested Parentheses алгоритм CFPQ_RSM показал себя эффективнее CFL_ADV в сценарии с несколькими источниками. Однако применение CFPQ_RSM на Java Points-to кажется не разумным.
4. Сильное замедление времени на Java Points-to связано с большим размером грамматики.

С помощью профилирования выявлено, что в грамматике Java Points-to фаза 1 алгоритма CFPQ_RSM занимает почти всё время выполнения — 98%. Для грамматики Nested Parentheses результат ниже — 90%, а у C Alias — около 55%. Из этого следует, что переходы по терминалам занимают основное время во всех трёх грамматиках.