Оптимизация алгоритма контекстно-свободной достижимости, основанного на операциях линейной алгебры

Муравьев И.В., СПбГУ, Санкт-Петербург, muravjovilya@gmail.com

Аннотация

Различные задачи статического анализа кода и анализа RDF-графов сводятся к задаче контекстно-свободной (КС) достижимости. Одним из многообещающих способов решения задачи КС-достижимости для встречающихся на практике больших графов является решение задачи в терминах операций линейной алгебры с разреженными матрицами, так как эти операции крайне эффективно выполняются на современном оборудовании. В работе предложены, реализованы и протестированы шесть оптимизаций матричного алгоритма КС-достижимости, основанных на свойствах полукольца КС-достижимости, специфике представления КС-языков и особенностях популярной библиотеки линейной алгебры SuiteSparse:GraphBLAS. Экспериментально установлено, что предложенные оптимизации позволяют в подавляющем большинстве случаев на порядки ускорить матричный решатель КС-достижимости и обогнать другие передовые универсальные КС-решатели.

Введение

Задача контекстно-свободной (КС) достижимости [1] — это задача поиска в рёберно-меченном графе пар вершин, соединённых таким путём, что метки рёбер вдоль этого пути формируют слово из заданного контекстно-свободного языка. Ряд вычислительно затратных задач сводится к задаче КС-достижимости, в частности: задача анализа потока значений (англ. *value-flow analysis*) [2], задача анализа псевдонимов (англ. *alias analysis*) [3], задача анализа указателей (англ. *points-to analysis*) [4] и задача поиска концептов одного уровня в RDF-графах (англ. *same layer of the hierarchy of RDF graphs*) [5].

Существует множество алгоритмов для решения задачи КС-достижимости [6, 7, 8, 2, 9]. Одним из перспективных универсальных подходов является матричный алгоритм Рустама Азимова [6]. Данный алгоритм стремится достичь высокой производительности за счёт решения задачи в терминах операций с разреженными матрицами, эффективно выполняемыми на современном параллельном аппаратном обеспечении, и

часто превосходит передовые аналоги по скорости работы [10]. Тем не менее, несмотря на эффективность матричных операций, в ряде случаев матричный алгоритм, как и другие инструменты, работает крайне долго [11, 12], в частности когда требуется найти пути с глубокими деревьями вывода и/или используются КС-грамматики с большим количеством правил вывода.

Следовательно, одним из перспективных способов ускорить решение упомянутых вычислительно-затратных задач является оптимизация матричного алгоритма КС-достижимости, чему и посвящена данная работа.

Терминология

Определение 1. (Решение задачи КС-достижимости) Пусть G — это рёберно-меченный граф с вершинами V, а Gr — КС-грамматика со стартовым нетерминалом S. Тогда *решением задачи КС-достижимости* называют множество $\{\langle v_i, v_j\rangle \in V^2 \mid \exists$ путь из v_i в v_j, выводимый из нетерминала $S\}$.

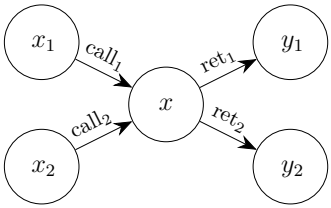

Рис. 1: Граф для анализа потока значений в программе на рис. 2.

```
def identity(x):
    return x

def main():
    y1 = identity(x1)
    y2 = identity(x2)
```

Рис. 2: Пример анализируемой Python-программы.

Пример 1. Пусть G — это граф на рис. 1, а Gr — КС-грамматика со следующими правилами вывода:

$$\forall i \in \{1, 2\} : S \to call_i\ ret_i;$$
$$\forall i \in \{1, 2\} : S \to call_i\ S\ ret_i\ S.$$

Тогда решением задачи КС-достижимости для G и Gr является множество $\{\langle x_1, y_1\rangle, \langle x_2, y_2\rangle\}$, так как $x_1 \xrightarrow{\text{call}_1\ \text{ret}_1} y_1$ и $x_2 \xrightarrow{\text{call}_2\ \text{ret}_2} y_2$.

Оптимизации

В ходе работы путём профилирования и анализа временной сложности последовательно выявлялись и ускорялись наиболее узкие места матричного

алгоритма КС-достижимости Рустама Азимова [6]. В результате этого процесса были предложены следующие оптимизации:

- инкрементальное вычисление произведений матриц (переиспользование результатов предыдущих итераций алгоритма);
- быстрое выполнение операций с нулевыми матрицами без вызова функций библиотеки линейной алгебры;
- динамический выбор между строчным и столбцовым форматами представления матриц с возможностью хранения частей одной матрицы над полукольцом КС-достижимости в разных форматах;
- динамический выбор между символьными и конкретными вычислениями (иногда целесообразнее не вычислять значение матрицы, а запомнить выражение, позволяющее его вычислить, чтобы потом подставлять это выражение при каждом использовании матрицы и производить упрощения);
- использование блочных матриц для эффективной работы с встречающимися на практике КС-грамматиками с большим количеством однотипных правил вывода, то есть КС-грамматиками, в которых, как в КС-грамматике из примера 1, есть правила, повторяющиеся для всех допустимых значений i;
- выявление эмпирическим путём эквивалентных КС-грамматик, задающих тот же КС-язык, что и входная КС-грамматика, но требующих меньшего объёма вычислений для решения задачи.

Реализация

Предложенные оптимизации были реализованы в разработанном в рамках работы матричном КС-решателе FastMatrixCFPQ[1]. Решатель был разработан на языке программирования Python с использованием библиотеки разреженной линейной алгебры SuiteSparse:GraphBLAS[2] и состоит из четырёх компонентов верхнего уровня со следующими зонами ответственности:

- `cfpq_matrix` — оптимизации на уровне отдельных матриц, а также ускорение сборки мусора (матриц, которые перестали использоваться);

[1]Разработанный в рамках работы решатель задачи КС-достижимости — https://github.com/FormalLanguageConstrainedPathQuerying/CFPQ_PyAlgo/tree/murav/optimize-matrix.

[2]Библиотека для параллельной работы с графами на языке разреженной линейной алгебры, SuiteSparse:GraphBLAS — https://github.com/DrTimothyAldenDavis/GraphBLAS.

- `cfpq_model` — представление графов и КС-языков;
- `cfpq_algo` — реализации матричного алгоритма КС-достижимости;
- `cfpq_cli` — интерфейс командной строки.

Большую часть предложенных оптимизаций удалось реализовать в виде обёрток, которые можно добавлять и убирать, что позволило покрыть интеграционными тестами все 64 комбинации предложенных оптимизаций.

Помимо КС-решателя, в рамках работы была разработана система, автоматизирующая экспериментальное сравнение КС-решателей[3]. Эта система была проинтегрирована с шестью передовыми КС-решателями [4, 2, 6, 9, 13, 14] и поддерживает настройку параметров экспериментов, прерывание и возобновление экспериментов, а также визуализацию результатов.

Наконец, на языке программирования Kotlin был разработан вспомогательный проект[4], отвечающий за построение дополнительных наборов входных данных по Java-программам с помощью анализа трёхадресных инструкций, в которые библиотека JacoDB[5] преобразует JVM-байткод.

Эксперимент

Перед проведением экспериментального исследования были сформулированы два исследовательских вопроса.

RQ1. Даёт ли FastMatrixCFPQ правильные ответы?

RQ2. Когда FastMatrixCFPQ работает быстрее аналогов и/или использует меньше оперативной памяти?

Для ответа на эти вопросы было проведено экспериментальное сравнение FastMatrixCFPQ с передовыми аналогами: MatrixCFPQ [6], PEARL 2023 [9], POCR 2022 [2], KotGLL 2023 [14], Graspan [13] и Gigascale [4]. Входные данные были взяты из наборов данных CFPQ_Data[6] и CPU 17[7], на которых замеряли свою производительность перечисленные аналоги. Входные данные

[3]Разработанная в рамках работы система автоматизации экспериментов — https://github.com/FormalLanguageConstrainedPathQuerying/CFPQ_PyAlgo/tree/murav/optimize-matrix/cfpq_eval.

[4]Разработанный в рамках работы проект для построения графов по Java-программам — https://github.com/FormalLanguageConstrainedPathQuerying/CFPQ_JavaGraphMiner.

[5]Библиотека для анализа JVM-байткода, JacoDB — https://jacodb.org.

[6]Набор данных CFPQ_Data, графы и КС-грамматики для оценки производительности КС-решателей — https://github.com/FormalLanguageConstrainedPathQuerying/CFPQ_Data.

[7]Набор данных CPU 17, графы программ из набора данных SPEC 2017 C/C++ — https://github.com/kisslune/CPU17-graphs.

были дополнены графами для анализа указателей, построенными в рамках работы по популярным Java-библиотекам.

Все рассмотренные наборы данных в совокупности содержат более 50 графов. Каждый граф относится к одной из пяти прикладных задач, требующих использования различных КС-грамматик. На рис. 3 и 4 каждая задача представлена одним графом. Чтобы выборка представленных графов была максимально непредвзятой, для каждой задачи выбран тот граф, который FastMatrixCFPQ анализировал дольше остальных графов для той же задачи.

Для замеров производительности использовались центральный процессор AMD Ryzen 9 7900X и 128 ГБ оперативной памяти DDR5. Программное окружение было зафиксировано с помощью Docker-образа[8].

RQ1: правильность ответов. Для всех рассмотренных входных данных FastMatrixCFPQ получил ответы, совпавшие с ответами других корректно работающих КС-решателей.

RQ2: производительность. Как показано на рис. 3, FastMatrixCFPQ для показанных графов обгоняет все рассмотренные аналоги, кроме Gigascale [4]. Это объясняется тем, что Gigascale — это специализированный инструмент, оптимизированный для работы с одним конкретным КС-языком. Что касается использования оперативной памяти, то, как показано на рис. 4, для FastMatrixCFPQ обычно требуется меньше памяти, чем для аналогов, но есть исключения, в частности, для графа «taxon_h» меньше памяти требуется инструменту Graspan [13], а для графа «perlb_aa» — инструменту MatrixCFPQ [6]. Однако стоит отметить, что в данных случаях соответствующие инструменты работают более чем в 8 раз дольше (см. рис. 3).

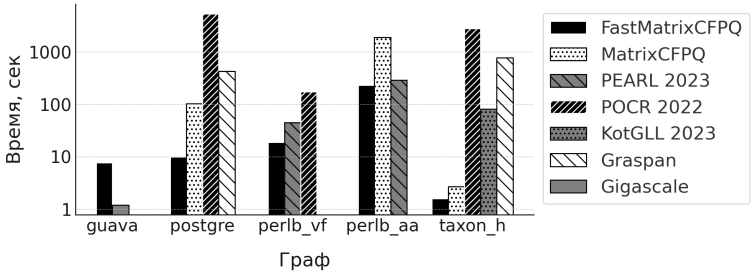

Рис. 3: Время работы КС-решателей, выборочное стандартное отклонение по результатам пяти замеров в пределах 7%. Отсутствие столбца обозначает превышение ограничения по времени в 10 000 секунд, ограничения по памяти в 128 ГБ или несовместимость инструмента с задачей.

[8]Репозиторий с Docker-образом для проведения экспериментального сравнения решателей КС-достижимости — https://hub.docker.com/r/cfpq/py_algo_eval.

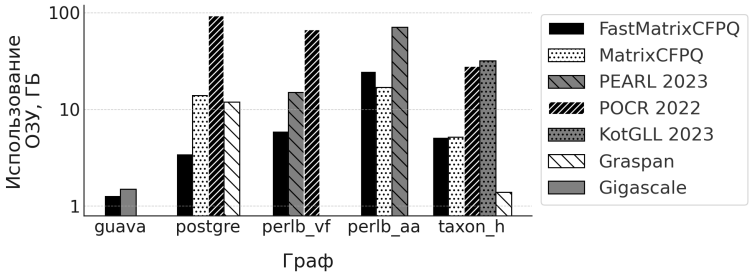

Рис. 4: Использование ОЗУ КС-решателями, выборочное стандартное отклонение по результатам пяти замеров в пределах 4%. Обозначения аналогичны рис. 3.

Заключение

В работе были предложены шесть ортогональных оптимизаций матричного алгоритма КС-достижимости, позволившие обогнать передовые универсальные КС-решатели. Тем не менее, предложенных оптимизаций оказалось недостаточно, чтобы обогнать инструмент Gigascale [4], специализированный для одного конкретного КС-языка, из чего можно сделать вывод, что перспективным направлением дальнейшего развития может стать обобщение подходов Gigascale [4] для произвольных КС-языков.

Список литературы

[1] Yannakakis M. Graph-Theoretic Methods in Database Theory // Proc. of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '90). — 1990. — P. 230–242. https://doi.org/10.1145/298514.298576

[2] Lei Y., et al. Taming Transitive Redundancy for Context-Free Language Reachability // Proc. ACM Program. Lang. — 2022. — Vol. 6, no. OOPSLA2, art. no. 180. https://doi.org/10.1145/3563343

[3] Zheng X., Rugina R. Demand-Driven Alias Analysis for C // SIGPLAN Not. — 2008. — P. 197–208. https://doi.org/10.1145/1328897.1328464

[4] Dietrich J., et al. Giga-Scale Exhaustive Points-to Analysis for Java in under a Minute // Proc. of the 2015 ACM SIGPLAN International Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2015). — 2015. — P. 535–551. https://doi.org/10.1145/2814270.2814307

[5] Zhang X., et al. Context-Free Path Queries on RDF Graphs // The Semantic Web – ISWC 2016. — 2016. — P. 632–648. https://link.springer.com/chapter/10.1007/978-3-319-46523-4_38

[6] Azimov R., Grigorev S. Context-Free Path Querying by Matrix Multiplication // Proc. of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems and Network Data Analytics (GRADES-NDA '18). — 2018. — Art. no. 5. https://doi.org/10.1145/3210259.3210264

[7] Medeiros C., et al. Costa U. Efficient Evaluation of Context-Free Path Queries for Graph Databases // Proc. of the 33rd Annual ACM Symposium on Applied Computing (SAC '18). — 2018. — P. 1230–1237. https://doi.org/10.1145/3167132.3167265

[8] Orachev E., et al. Context-Free Path Querying by Kronecker Product // Advances in Databases and Information Systems: 24th Europ. Conf., ADBIS 2020 — 2020. — P. 49–59. https://doi.org/10.1007/978-3-030-54832-2_6

[9] Shi C., et al. Two Birds with One Stone: Multi-Derivation for Fast Context-Free Language Reachability Analysis // 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE). — 2023. — P. 624-636. https://doi.ieeecomputersociety.org/10.1109/ASE56229.2023.00118

[10] Azimov R. Context-Free Path Querying Using Linear Algebra. PhD thesis, St. Petersburg State University, 2022. https://disser.spbu.ru/files/2022/disser_azimov.pdf

[11] Kutuev V. Experimental investigation of context-free-language reachability algorithms as applied to static code analysis. Master's thesis, St. Petersburg State University, 2023. http://hdl.handle.net/11701/42628

[12] Kuijpers J., et al. An Experimental Study of Context-Free Path Query Evaluation Methods // Proc. of the 31st International Conference on Scientific and Statistical Database Management (SSDBM '19). — 2019. — P. 121–132. https://doi.org/10.1145/3335783.3335791

[13] Wang K., et al. Graspan: A Single-Machine Disk-Based Graph System for Interprocedural Static Analyses of Large-Scale Systems Code // SIGPLAN Not. — 2017. — Vol. 52, no. 4. — P. 389–404. https://doi.org/10.1145/3093336.3037744

[14] Abzalov V. Implementation and experimental investigation of the GLL parser based on a recursive automaton. Bachelor's thesis, St. Petersburg State University, 2023. http://hdl.handle.net/11701/42730