Санкт-Петербургский государственный университет

Кафедра системного программирования

Алимов Павел Геннадьевич

Создание Python обёртки над библиотекой линейной разреженной алгебры

Курсовая работа

Научный руководитель:
к. ф.-м. н., доцент кафедры информатики Григорьев С. В.

Санкт-Петербург
2021

Оглавление

Введение

Зачастую решения задач в реальном мире сводятся к решению систем линейных уравнений, решению дифференциальных уравнениям в частных производных, вычислению над наборами статистических данных и тому подобным подходам, которое удобно производить с помощью компьютера. И как результат такого перехода возникает необходимость представления объектов в форме удобной для проведения вычислений над ними и хранения результатов, и в качестве такой формы можно выбрать матрицы.

При этом вышеупомянутые объекты не существуют изолированно друг от друга, между ними бывают зависимости или отношения, и часто встречается необходимость выявления более сложных взаимосвязей между объектами. Если эти отношения между объектами представить в виде графа, то в терминах теории графов выявления более сложных зависимостей является задачей достижимости. Одним из способов задания отношений между объектами графа является описание ограничений на пути между вершинами. Для этого можно использовать контекстно-свободные грамматики над алфавитом меток рёбер графа.

Проблема в том, что существующие для реальных задач графы зачастую являются очень большими и при представлении в виде матриц оказываются не очень плотными, то есть в них много ничего не значащих элементов, для их описания можно использовать разреженные матрицы. Из-за размера графов вычисления над ними занимают много времени, так, к примеру, в своём исследовании Йохем Куйперс и др. [3] провели сравнительный анализ наиболее известных алгоритмов поиска путей в графе с контекстно-свободными ограничениями и пришли к выводу, что существующие реализации обладают большим временем работы.

Но, в то же время, студенты кафедры Системного Программирования Никита Мишин и др. показали [2], что использование технологии GPGPU может ускорить выполнение контекстно-свободных запросов. При этом при помощи разреженных матриц можно ускорить вычис-

ления и обрабатывать большие матрицы на GPU. Вкупе с этим существуют различные библиотеки реализаций алгебраических действий над разреженными матрицами для вычислений на GPU, такие как clSPARSE[1], CuBool[2] и прочие, написанные на *C*, и при обосновании выбора библиотеки для использования необходимо иметь результаты сравнения их эффективности. Вместе с тем уже существует готовый инструментарий[3], созданный студентами кафедры Системного Программирования, Арсением Тереховым, Ильёй Эпельбаумом и др., для проведения экспериментов над графами с контекстно-свободными ограничениями, написанный на *Python*.

Таким образом мы приходим к тому, что необходимо иметь возможность в *Python* вызывать функции библиотек, написанных на *C*, взяв за основу интерфейс библиотеки *CuBool*, а значит требуется обёртка над ними, позволяющая обращаться к *C* функциям из *Python*.

[1]Библиотека работы с разреженными матрицами. Исходный код библиотеки: https://github.com/clMathLibraries/clSPARSE. Дата посещения: 13.02.2021.

[2]Библиотека работы с разреженными матрицами. Исходный код библиотеки: https://github.com/JetBrains-Research/cuBool. Дата посещения: 13.02.2021.

[3]Окружение для разработки, тестирования и оценки эффективности алгоритмов, решающих задачи вычисления запросов в графах с ограничениями на пути между вершинами в виде формальных языков. Исходный код окружения: https://github.com/JetBrains-Research/CFPQ_PyAlgo. Дата посещения: 26.12.2020.

1. Цели и задачи

Цель — создание обёртки для библиотеки алгебраических действий над разреженными матрицами.

Задачи:

1. Выбрать инструмент для создания обёртки, позволяющей вызывать функции библиотек из *Python*.

2. Создать обёртку над библиотекой CuBool.

2. Обзор инструментов для создания обёртки

Исходя из того, что библиотеки написаны на *C/C++*, а тестовое окружение для алгоритма решения задачи контестно-свободной достижимости [1] написано на *Python*, следует, что выбранный инструмент должен позволять вызывать функции *C/C++* из *Python*. А также по возможности хотелось бы не вносить изменения в исходный код библиотек, так как внесённые изменения могут привести к ухудшению их производительности, приходим к следующим требованиям для инструмента.

- Инструмент должен позволять вызывать функции написанные на *C/C++* из *Python*.
- Инструмент не должен требовать изменения исходного кода написанного на *C/C++*.

Под первый критерий, как самый общий, подходят следующие инструменты: SWIG, CFFI, Boost Python, Ctypes, Python/C API. Рассмотрим эти инструменты на соответствие второму критерию.

SWIG *(Simplified Wrapper and Interface Generator)*[4] — это инструмент разработки программного обеспечения, который связывает программы, написанные на *C* и *C++*, с множеством языков программирования высокого уровня, таких как: *Python, Javascript, Ruby и другие.*

Требования к использованию:

- исходный код на *C/C++*;
- специальный файл интерфейса, написанный на SWIG, для исходного кода;
- собранная библиотека исходного кода вместе с файлом интерфейса.

[4]Официальный сайт SWIG: http://www.swig.org. Дата посещения: 26.12.2020.

Вывод — этот инструмент вполне подходит для создания нужной обёртки, но требует ручной сборки библиотеки, то есть не удовлетворяет второму критерию.

CFFI *(C Foreign Function Interface)*[5] — позволяет взаимодействовать практически с любым кодом C из Python на основе C-подобных объявлений.

Требования к использованию:

- исходный код только на *C*. Нет поддержки условных директив препроцессора;
- собранная библиотека исходного кода.

Вывод — этот инструмент не поддерживает условные директивы препроцессора, а, к примеру, библиотека *clSPARSE* написана на *C++* и обёрнута в *C*, и отсюда следует вывод, что *CFFI* не подходит для реализации обёртки.

Boost Python[6] — это фреймворк для взаимодействия *Python* и *C++*. Он позволяет обращаться к функциям и объектам классов *C++* для *Python* и наоборот.

Требования к использованию:

- исходный код на *C/C++*;
- написанная обёртка над исходным кодом с использованием функций из фреймворка Boost;
- собранная библиотека исходного кода.

Вывод — этот инструмент требует изменения исходного кода библиотек, для создания обёртки над ней, то есть не подходит по второму критерию.

Ctypes[7] — это библиотека для языка Python, которая предоставляет типы данных, совместимые с C, и позволяет вызывать функции

[5]Официальный сайт CFFI: https://cffi.readthedocs.io/en/latest/. Дата посещения: 26.12.2020.

[6]Официальный сайт Boost Python: https://www.boost.org/doc/libs/1_75_0/libs/python/doc/html/index.html. Дата посещения: 26.12.2020.

[7]Официальный сайт ctypes: https://docs.python.org/3/library/ctypes.html. Дата посещения: 26.12.2020.

библиотек написанных на *C* в *Python*. *Ctypes* можно использовать для обертывания этих библиотек на Python.

Требования к использованию:

- исходный код только на *C*. Нет поддержки *C++*;
- Собранная библиотека исходного кода.

Вывод — этот инструмент полностью удовлетворяет по функциональности и возможностям, и не требует никаких дополнительных действий для создания обёртки, то есть подходит по обоим критериям.

Python/C API[8] — дает программистам на *C* и *C++* доступ к интерпретатору Python на различных уровнях. API можно использовать и с *C++*, но для краткости его обычно называют *Python/C API*.

Требования к использованию:

- исходный код написанный с использованием библиотеки *"python"* для *C/C++*;
- Собранный *C/C++* файл в модуль Python.

Вывод — этот инструмент требует исходного кода написанного с использованием библиотеки *python*, а, выбранная библиотека CuBool не соответствует этому требованию, то есть использование этого интсрумента требует внесения изменений в исходную библиотеку, то есть не подходит по второму критерию.

Таким образом можно сделать заключение, что из вышеперечисленных инструментов подходит только *Ctypes*, так как остальные инструменты не удовлетворяют как минимум одному из поставленных критериев.

[8]Официальный сайт Python/C API: https://docs.python.org/3.8/c-api/. Дата посещения: 26.12.2020.

3. Реализация

Архитектуру всего проекта можно описать следующим образом. На вход реализованному с помощью вышеупомянутого окружения *CFPQ_PyAlgo*[9] алгоритму подаётся граф и контекстно-свободная грамматика. А затем выполняется сам алгоритм, используя функции библиотеки разреженной алгербы с помощью созданной в ходе этой работы обёртки, в данном случае CuBool. На рисунке 1 изображена общая архитектура проекта описанная выше.

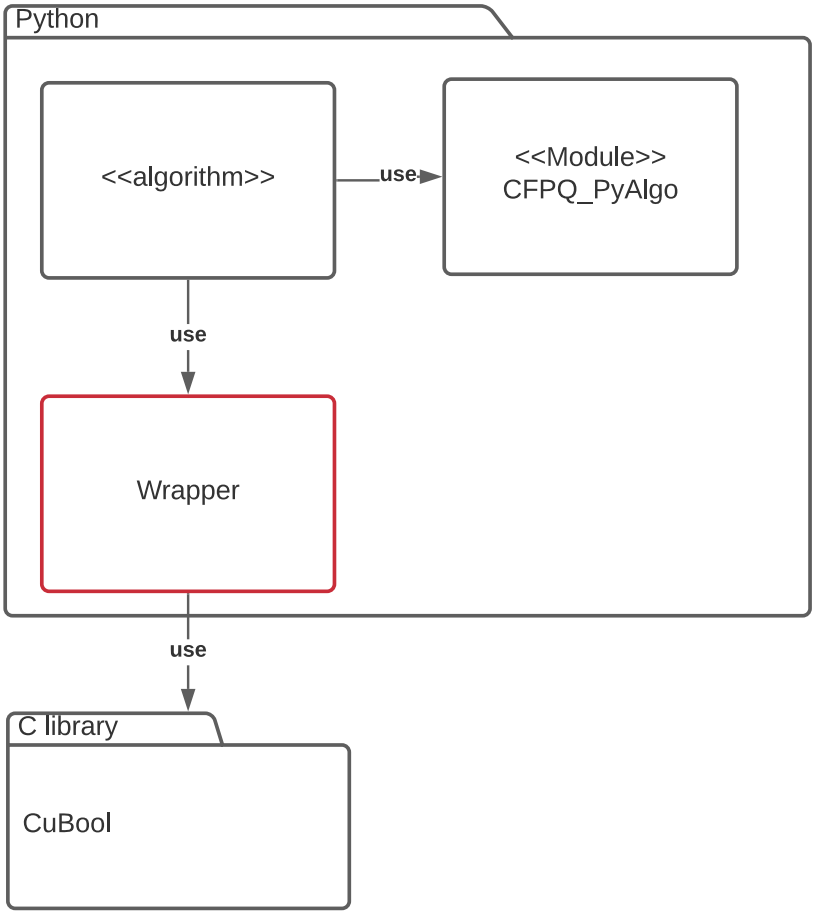

Рис. 1: Общая архитектура. Красным выделена цель данной работы.

[9]Окружение для разработки, тестирования и оценки эффективности алгоритмов, решающих задачи вычисления запросов в графах с ограничениями на пути между вершинами в виде формальных языков. Исходный код окружения: https://github.com/JetBrains-Research/CFPQ_PyAlgo. Дата посещения: 26.12.2020.

3.1. Обёртка

Было принято решение делать обёртку, опираясь на API библиотеки CuBool[10], так как общий интерфейс библиотек должен быть похожим на интерфейс библиотеки CuBool. Таким образом была написана обёртка над библиотекой CuBool, позволяющая выполнять алгебраические операции над матрицами. При разработке было решено сделать API обёртки похожим на pygraphblas[11], как на уже существующий известный инструмент работы с матрицами, для упрощения восприятия.

На рисунке 2 представлена получившаяся архитектура решения. В ходе работы, в репозитории библиотеки *CuBool*[4] в разделе *python* был написан модуль *bridge*, в котором определяются функции исходной библиотеки *CuBool*, такие как создание и удаление объекта типа *"матрица"*, поэлементное сложение матриц, произведение Кронекера, матричное умножение, взятие подматрицы, создание копии, транспонирование, получение информации о матрице. А также написан модуль *wrapper*, который хранит определения уже упомянутых функций в виде единого объекта и предоставляет доступ к ним. При этом был создан класс *Matrix*, который предоставляет доступ пользователю к библиотечным функциям с помощью своих методов. Для упрощения ввода и вывода матриц и демонстрации результата были написаны функции, позволяющие генерировать матрицу, считывать матрицу в mtx формате из файла и записывать матрицу в файл в mtx формате. Функция генерации матриц в качестве входных параметров принимает плотность заполнения и кортеж из числа столбцов и строк, и возвращает случайным образом заполненную матрицу заданной формы и плотности.

[10]Библиотека работы с разреженными матрицами. Исходный код библиотеки: https://github.com/JetBrains-Research/cuBool. Дата посещения: 13.02.2021.

[11]pygraphblas - *Python* обёртка над GraphBLAS API. Исходный код обёртки - https://github.com/Graphegon/pygraphblas. Дата посещения: 30.03.2021

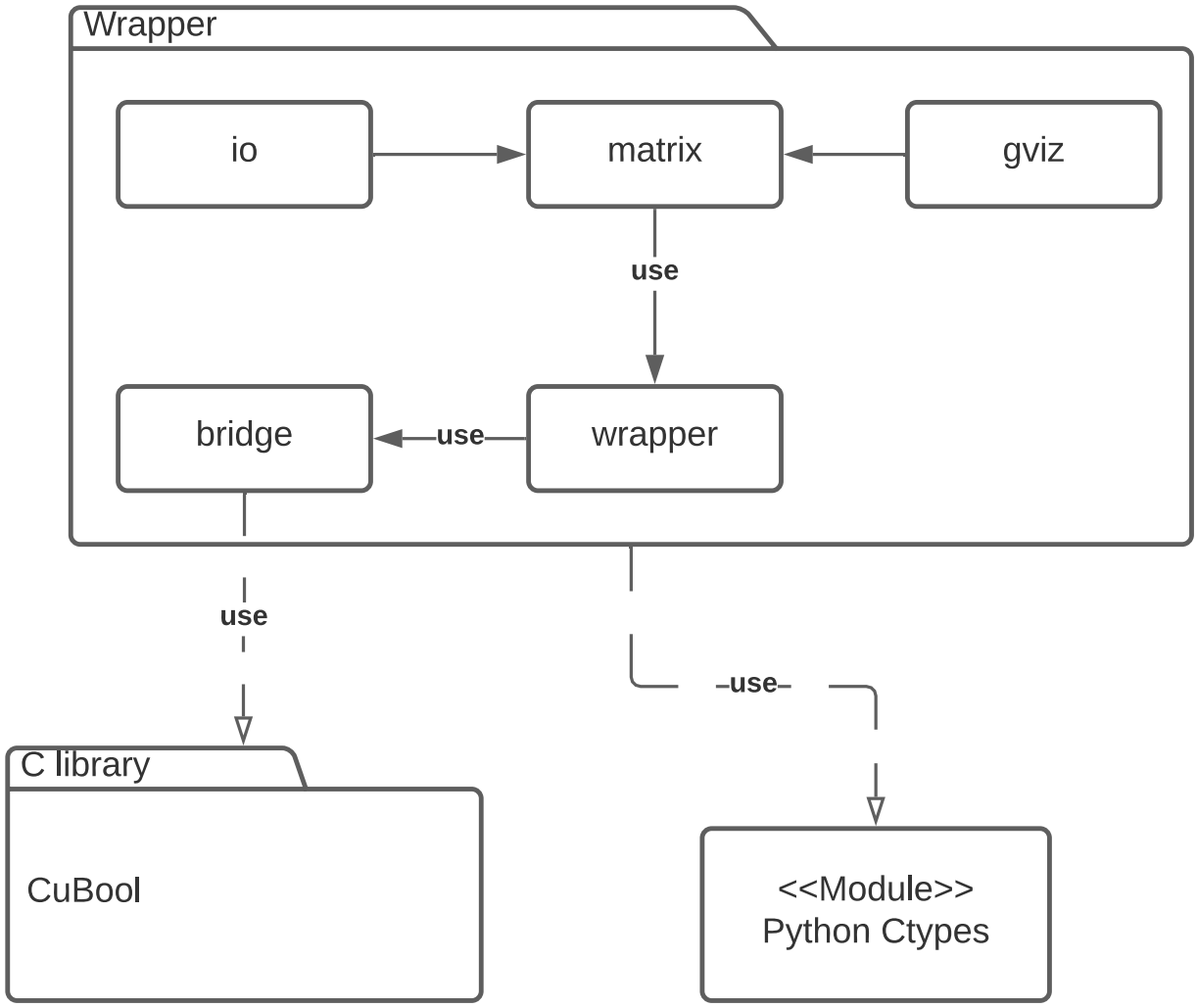

Рис. 2: Архитектура работы.

3.2. Регрессионные тесты

С помощью вышеописанных функций был сгенерирован набор матриц различной формы для тестирования каждой из матричных операций.

Заодно были написаны сами тесты на каждую матричную операцию, которые считывают уже сгенерированную матрицу из входного файла, применяют к входной матрице соответствующую функцию и сравнивают получившийся результат с прошлым результатом работы.

3.3. Графовое представление матрицы смежности

Также была написана функция, переводящая матрицы в *Graphviz* скрипт[12], для возможности визуализации представляемых графов. Было принято решение добавить возможность выбора цвета для ребёр и вершин графов, а также номер начальной вершины и метки на рёбрах для построения соответствующей строки на языке *DOT*, который поддерживает *Graphviz*. Таким образом получилась функция, которые по нескольким входным матрицам строит их графовое представление в рамках одного графа, что можно использовать, в частности, для визуализации графов с контекстно-свободными ограничениями.

К примеру, для двух матриц смежности:

$$a = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \text{ и } b = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix};$$

задав для матрицы a красный цвет рёбер и для матрицы b зелёный цвет рёбер, получим скрипт описанный в листинге 1, по которому строится граф представленный на рисунке 3.

[12]Graphviz - это программа для визуализации графов. Официальный сайт - https://graphviz.org. Дата посещения: 30.03.2021.

Listing 1: Скрипт на языке DOT, описывающий граф.

```
digraph G {
    graph [label=Test];
    node [color=black];
    1 -> 3 [label=a,color=red];
    2 -> 3 [label=a,color=red];
    3 -> 7 [label=a,color=red];
    4 -> 6 [label=a,color=red];
    5 -> 6 [label=a,color=red];
    6 -> 7 [label=a,color=red];
    3 -> 1 [label=b,color=green];
    3 -> 2 [label=b,color=green];
    7 -> 3 [label=b,color=green];
    6 -> 4 [label=b,color=green];
    6 -> 5 [label=b,color=green];
    7 -> 6 [label=b,color=green];
}
```

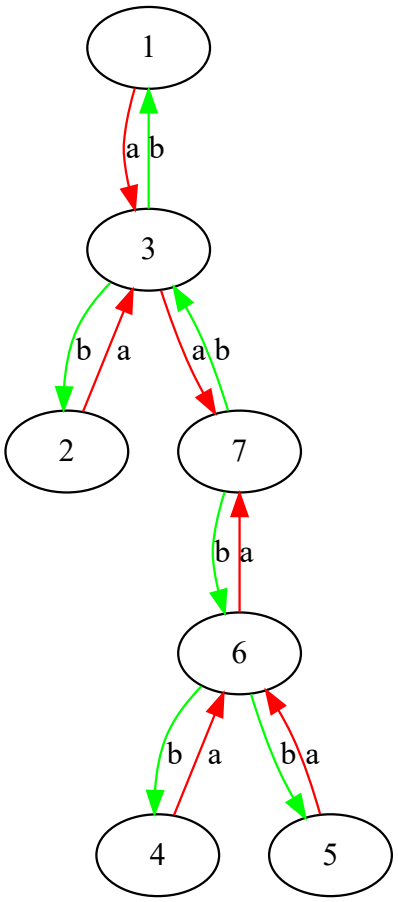

Рис. 3: Граф с метками на рёбрах

3.4. Примеры использования

В дополнение к созданной обёртке были написаны примеры[13] использования методов класса *Matrix* с выводом полученного результата и подробными комментариями. А также создан сопровождающий текстовый документ[14] с кодом примеров и соответствующими результатами их исполнения.

[13]Примеры использования методов класса Matrix. Исходный код примеров: https://github.com/JetBrains-Research/cuBool/tree/master/python/examples. Дата посещения: 04.06.2021.

[14]Документ со текстом всех примеров использования методов класса Matrix и результатом их исполнения. Ссылка на документ: https://github.com/JetBrains-Research/cuBool/blob/master/docs/examples/python_examples.md. Дата посещения: 04.06.2021.

Заключение

В рамках данной работы были получены следующие результаты:

- был выбран инструмент *Ctypes* для создания обёртки;
- с помощью выбранного инструмента в репозитории библиотеки *CuBool*[4] на *Python* реализовано:
  1. чтение матрицы из файла;
  2. запись матрицы в файл;
  3. создание матрицы, как объекта библиотеки;
  4. вызов библиотечных алгебраических операций над матрицами;
  5. написаны регрессионные тесты к обёртке;
  6. перевод матрицы в *Graphviz* скрипт;
  7. написаны примеры использования.

Список литературы

[1] Azimov Rustam, Grigorev Semyon. Context-Free Path Querying by Matrix Multiplication // Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences Systems (GRADES) and Network Data Analytics (NDA). — GRADES-NDA '18. — New York, NY, USA : Association for Computing Machinery, 2018. — 10 p. — URL: https://doi.org/10.1145/3210259.3210264.

[2] Evaluation of the Context-Free Path Querying Algorithm Based on Matrix Multiplication / Nikita Mishin, Iaroslav Sokolov, Egor Spirin et al. // Proceedings of the 2nd Joint International Workshop on Graph Data Management Experiences Systems (GRADES) and Network Data Analytics (NDA). — GRADES-NDA'19. — New York, NY, USA : Association for Computing Machinery, 2019. — 5 p. — URL: https://doi.org/10.1145/3327964.3328503.

[3] An Experimental Study of Context-Free Path Query Evaluation Methods / Jochem Kuijpers, George Fletcher, Nikolay Yakovets, Tobias Lindaaker // Proceedings of the 31st International Conference on Scientific and Statistical Database Management. — SSDBM '19. — New York, NY, USA : Association for Computing Machinery, 2019. — P. 121–132. — URL: https://doi.org/10.1145/3335783.3335791.

[4] Orachyov Egor, Alimov Pavel, Grigorev Semyon. cuBool: sparse Boolean linear algebra for Nvidia Cuda. — 2021. — Version 1.1.0. URL: https://github.com/JetBrains-Research/cuBool.