Санкт-Петербургский государственный университет

Кафедра Системного программирования

Ершов Кирилл Максимович

Синтаксический анализ графов с помеченными вершинами и ребрами

Курсовая работа

Научный руководитель:
ст. преп., к. ф.-м. н. Григорьев С. В.

Санкт-Петербург
2017

Оглавление

Введение

Помеченные графы являются удобным способом представления различных структурированных данных. Такие графы используются, например, в биоинформатике, логистике, графовых базах данных.

Иногда для представления данных с использованием графов обходятся только метками на рёбрах. Но в некоторых случаях метки на вершинах позволяют более наглядно отображать зависимости между сущностями. К примеру, в биоинформатике существует большое количество данных, содержащих взаимосвязь между генами и белками. Такие данные удобно представлять в виде графа, вершины которого помечены определенными генами и белками, а ребра показывают их отношение (например, ген кодирует белок).

Для поиска информации в помеченном графе необходимо иметь возможность выполнять запросы, задающие класс путей в графе. Пути рассматриваются как строки, состоящие из меток на рёбрах и вершинах. Тогда запрос можно представить в виде грамматики: путь удовлетворяет запросу, если он принадлежит языку, который порождает заданная грамматика. Таким образом, грамматика задаёт класс путей в графе и задача выполнения запросов сводится к задаче синтаксического анализа графа.

Существуют различные языки запросов для получения нужных путей из графа. Но многие из них позволяют задавать только регулярные запросы [6] [1] [5] или поддерживают графы с метками только на рёбрах [4]. В некоторых случаях с помощью регулярных грамматик невозможно задать нужные запросы. Поэтому актуальна задача организации более выразительных запросов, используя контекстно-свободные грамматики.

Для синтаксического анализа строки по произвольной КС-грамматике существуют различные алгоритмы. Например, Early parser [3] , CYK [13], GLR [10], GLL [8]. Алгоритм GLL имеет оптимальное время работы ($O(n^3)$ в худшем случае) и основан на идее нисходящего анализа, а значит более удобен для реализации. К тому же алгоритм GLL реализо-

ван в исследовательском проекте YaccConstructor [12], в рамках которого выполняется данная работа. Поэтому для синтаксического анализа графа используется именно этот алгоритм.

1. Постановка задачи

Целью данной работы является добавление в проект YaccConstructor возможности выполнения запросов с контекстно-свободными ограничениями к графу с метками на вершинах и рёбрах. Для её достижения были поставлены следующие задачи:

- реализовать возможность поиска путей в графе с помеченными вершинами и рёбрами по заданной КС-грамматике;
- реализовать удобный интерфейс для получения и обработки результатов;
- провести апробацию для оценки производительности.

2. Обзор

2.1. Синтаксический анализ графов

Для поиска путей в графе существует множество инструментов, позволяющих находить пути по регулярным грамматикам. Решений для поиска путей по КС-грамматике не так много, в особенности для графов с метками на вершинах и рёбрах.

В работе [9] решалась задача извлечения связного подграфа, состоящего из путей между двумя исходными вершинами, из графа с метками на вершинах и рёбрах. Класс подходящих путей описывается с помощью контекстно-свободной грамматики. Для синтаксического анализа используется алгоритм Earley, работающий в худшем случае за время $O(n^3)$. Однако, поиск путей производится не в исходном графе с метками на вершинах и рёбрах, а в преобразованном. Перед началом работы алгоритма из исходного получают новый двудольный граф с метками только на рёбрах. Новый граф имеет в 2 раза больше вершин и увеличивает число рёбер. Даже при небольших входных данных и для путей длины не больше 8 алгоритм работает 240 секунд, что делает его мало применимым на практике.

Одним из распространённых способов представлять данные в удобном для обработки виде является модель RDF. Данные, записанные в RDF, представляют собой набор триплетов субъект–предикат–объект. В совокупности они образуют помеченный ориентированный граф. Многие данные в биоинформатике представлены именно в таком формате.

Самым популярным языком для запросов к данным, представленным в формате RDF, является язык SPARQL [6]. Однако, он позволяет описывать только регулярные выражения. В статье [2] авторы описали алгоритм для поиска путей в RDF-графе, принадлежащих КС-языку, а также предложили язык csSPARQL, поддерживающий КС-грамматики. Показано, что сложность алгоритма $O((|N| * |G|)^3)$, где N — нетерминалы входной грамматики, G — RDF-граф.

Также задача выполнения КС-запросов к графу решалась в статье

[4]. В данной работе был разработан алгоритм поиска путей, удовлетворяющих конъюктивным контекстно-свободным грамматикам. Реализация основана на алгоритме синтаксического анализа CYK. Однако в статье запросы выполняются к графу без меток на вершинах. Для того, чтобы использовать этот алгоритм для графа с помеченными вершинами и рёбрами, необходимо сначала преобразовать граф, что потребует дополнительных ресурсов.

2.2. YaccConstructor

На кафедре Системного программирования в лаборатории языковых инструментов разрабатывается проект YaccConstructor. Это платформа для исследований в области синтаксического анализа, написанная на языке F#. YaccConstructor позволяет создавать синтаксические анализаторы и имеет модульную архитектуру.

В YaccConstructor реализован абстрактный алгоритм GLL. Исходная грамматика описывается на языке спецификации грамматик YARD [11]. Затем генератором из неё извлекается необходимая для работы алгоритма информация о грамматике. Во время выполнения алгоритм перемещается по входному объекту в зависимости от текущей позиции в грамматике. Объект, в котором требуется найти пути, удовлетворяющие исходной KC-грамматике, должен реализовывать интерфейс IParserInput. В результате работы алгоритма получается SPPF [7]. Это структура данных, которая эффективно хранит все деревья разбора, получаемые при синтаксическом анализе.

2.3. QuickGraph

В лаборатории языковых инструментов также поддерживается библиотека QuickGraph для платформы .NET, которая содержит различные реализации графов и алгоритмы для них. Для исполнения запросов к графам разрабатывается расширение библиотеки QuickGraph, позволяющая получать результаты запросов, например, в виде подграфа или множества путей.

3. Реализация

Абстрактный алгоритм GLL в YaccConstructor принимает на вход объект, с реализованным интерфейсом IParserInput. В рамках данной работы был реализован это интерфейс для графов с метками на вершинах и рёбрах. Для представления графа используется структура AdjacencyGraph из библиотеки QuickGraph, которая позволяет эффективно получать список рёбер исходящих из указанной вершины. Это используется при обходе графа алгоритмом GLL. Если текущая позиция — ребро, то следующей будет конечная вершина этого ребра. Если текущая позиция на вершине, то следующими будут все исходящие рёбра. Таким образом, алгоритмом проверяются все возможные пути в графе. Также для графа можно задать вершины, с которых будет начинать работу алгоритм и вершины, являющиеся конечными для синтаксического анализа. Для проверки работы алгоритма были написаны тесты.

В проекте QuickGraph есть метод, извлекающий подграф из SPPF. Но возвращает он граф с метками только рёбрах. Дополнительно была реализована возможность извлечения подграфа с метками на вершинах и рёбрах. Также реализована печать графа с метками на вершинах в dot-файл. Этот формат удобен для графического представления графов.

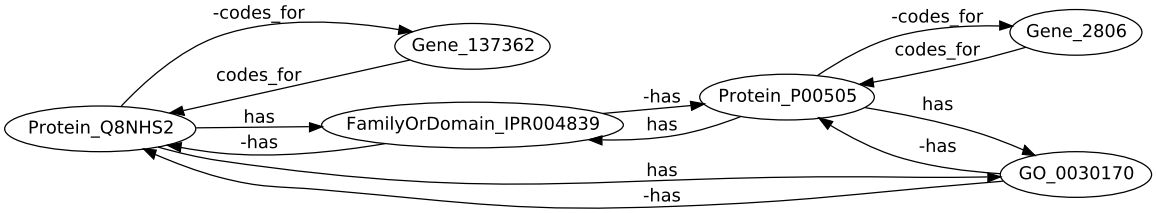

Рис. 1: Пример подграфа

4. Эксперименты

4.1. Данные

Существует большое количество биологических баз данных с открытым доступом, информация в которых может быть представлена как помеченный граф, в котором вершины соответствуют сущностям (протеины, гены, фенотипы), а рёбра отношениям между ними (взаимодействует, кодирует). Пути между вершинами позволяют найти новые связи в данных, либо показывают уже известные отношения. Подграф, построенный на всех найденных путях, более наглядно демонстрирует связи между вершинами.

Реальный набор биологических данных был собран из разных баз данных, находящихся в открытом доступе: Entrez Gene (информация о генах), UniProt (протеины), Gene Ontology (биологические процессы), STRING (связи между протеинами), InterPro (семейства белков), KEGG (связи между генами), HomoloGene (группы гомологий генов). Данные были ограничены набором из пяти организмов: Homo sapiens, Rattus norvegicus, Mus musculus, D. melanogaster и C. elegans. Объединенные в один файл данные состоят из троек: субъект, отношение, объект. Такие тройки образуют помеченный ориентированный граф.

```
[<Start>]
         s : gene
         v : protein | gene | GO | PATHWAY | FAMDOM
             | HOMOLOGENE
   similar : CODESFOR v RCODESFOR | BELONGS v RBELONGS
             | HAS v RHAS | HOMOLOGTO v RHOMOLOGTO
        ps :  (PROTEIN similar) *[1..2]
   protein : ps PROTEIN | PROTEIN
        gs : (GENE similar) *[1..2]
      gene : gs GENE | GENE
```

Рис. 2: Грамматика на языке YARD

4.2. Запросы

Все вершины в полученном графе имеют уникальную метку. Но для удобства будем различать их по типу: гены, фенотипы и т.д. Назовём две вершины в графе похожими, если они одного типа и имеют рёбра одного типа к похожим вершинам. Это определение рекурсивно. Таким образом, путь между похожими вершинами представляет собой палиндром, который нельзя задать с помощью регулярной грамматики.

На рисунке 2 показана КС-грамматика на языке YARD, задающая класс путей, в которых начальный и конечный терминалы являются похожими генами. Для поиска похожих генов нетерминал gene, обозначающий последовательность похожих генов, указывается стартовым. Нетерминал similar задаёт отношение схожести генов и протеинов: существуют рёбра одного типа (например, BELONGS и RBELONGS) к похожим вершинам, которые обозначаются нетерминалом v.

В статье [9] при тестировании производительности длину путей ограничивали от 4 до 8. В реализованном алгоритме GLL нет простого способа добавить такое ограничение. Поэтому, с целью уменьшения длины пути в грамматику были добавлены нетерминалы ps и gs, ограничивающие последовательность похожих протеинов и генов соответственно от от одного до двух.

На рисунке 1 показан пример подграфа, который является результа-

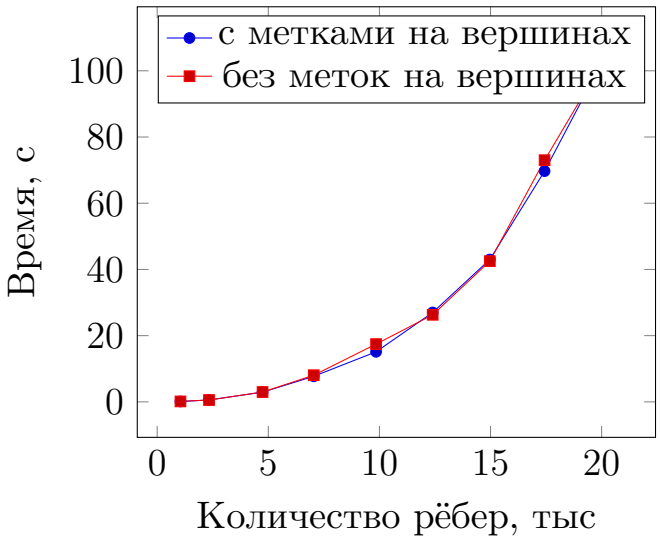

Рис. 3: Время работы алгоритма

том выполнения описанного выше запроса на небольших входных данных.

4.3. Производительность

Для оценки производительности была проведена серия экспериментов. Результаты приведены на графике, изображённом на рисунке 3. В статье [9] был проведён похожий эксперимент, но длины путей были ограничены от 4 до 8. В данной работе добиться такого ограничения не удалось, подграф строится по путям любой длины, поэтому нет возможности напрямую сравнить результаты.

Также была произведена серия экспериментов для сравнения производительности с алгоритмом GLL для графов без меток на вершинах. Замеры проведены на тех же данных, но с предварительной расклейкой графа в вершинно–рёберной граф (вершины с метками заменяются на рёбра). Результаты приведены на рисунке 3. Из графика видно, что выполнение запроса двумя способами занимает примерно одинаковое количество времени.

5. Заключение

В ходе работы получены следующие результаты:

- реализована возможность поиска путей в графе с помеченными вершинами и рёбрами по заданной КС-грамматике;
- написаны тесты;
- реализован удобный интерфейс для получения и обработки результатов;
- проведена апробация для оценки производительности.

5.1. Дальнейшее направление работ

Как показали результаты экспериментов, выполнение КС-запросов к графам уже с 20 тысячами рёбер занимает больше 100 секунд. Для уменьшения времени работы алгоритма и для выполнения запросов к графам большего размера исследовать возможность ограничения длины искомых путей в реализованном алгоритме GLL в YaccConstructor.

Список литературы

[1] Abiteboul Serge, Vianu Victor. Regular path queries with constraints // Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems / ACM. — 1997. — P. 122–133.

[2] Context-free path queries on RDF graphs / Xiaowang Zhang, Zhiyong Feng, Xin Wang et al. // International Semantic Web Conference / Springer. — 2016. — P. 632–648.

[3] Earley Jay. An efficient context-free parsing algorithm // Communications of the ACM. — 1970. — Vol. 13, no. 2. — P. 94–102.

[4] Hellings Jelle. Conjunctive context-free path queries. — 2014.

[5] Koschmieder André, Leser Ulf. Regular path queries on large graphs // Scientific and Statistical Database Management / Springer. — 2012. — P. 177–194.

[6] Prud'hommeaux Eric, Seaborne Andy. SPARQL Query Language for RDF. W3C Recommendation, January 2008. — 2008.

[7] Rekers Joan Gerard. Parser generation for interactive environments : Ph. D. thesis / Joan Gerard Rekers ; Universiteit van Amsterdam. — 1992.

[8] Scott Elizabeth, Johnstone Adrian. GLL parsing // Electronic Notes in Theoretical Computer Science. — 2010. — Vol. 253, no. 7. — P. 177–189.

[9] Sevon Petteri, Eronen Lauri. Subgraph queries by context-free grammars // Journal of Integrative Bioinformatics (JIB). — 2008. — Vol. 5, no. 2. — P. 157–172.

[10] Tomita Masaru. An efficient augmented-context-free parsing algorithm // Computational linguistics. — 1987. — Vol. 13, no. 1-2. — P. 31–46.

[11] YaccConstructor. YARD // YaccConstructor official page. — URL: http://yaccconstructor.github.io/YaccConstructor/yard.html.

[12] YaccConstructor. YaccConstructor // YaccConstructor official page. — URL: http://yaccconstructor.github.io.

[13] Younger Daniel H. Recognition and parsing of context-free languages in time n3 // Information and control. — 1967. — Vol. 10, no. 2. — P. 189–208.