Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра системного программирования

Кладов Алексей Александрович

Разработка и оптимизация тестирующей системы и задач по биоинформатике в рамках платформы Розалинд

Курсовая работа

Научный руководитель:
Вяххи Н. И.

Санкт-Петербург
2013

Оглавление

Введение

В последнее время становится более популярным on-line образование. Однако нельзя сказать, что все проблемы в этой области уже решены. Об этом может свидетельствовать, например, то что черезвычайно высокий процент студентов не доходят курс до конца. Другой важной проблемой является недостаток курсов по узко специализированным темам.

Rosalind – веб-платформа для изучания биоинформатики с помощью решения задач. Проект назван в честь Розалинд Франклин, одной из первооткрывателей спиралевидной структуры цепочки ДНК. Проект вдохновлён ProjectEuler.net [4] и Google Code Jam [2]. Разработка началась в мае 2012 года.

Главное содержание сайта – набор задач, организованный в граф зависимостей. Каждая задача состоит из теоретического материала и вычислительной задачи. В теоретической части задачи рассказывается о каком-нибудь биологическом феномене, вводятся различные термины и строятся математические модели. При этом все термины собраны в специальном глоссарии.

Условие вычислительной задачи строится на основании модели, сформулированной в теоретической части. Для решения задачи пользователю необходимо написать программу на любом языке программирования, скачать файл с входными данными, успеть обработать его за отведённое время и послать файл с ответом.

Задачи связанны между собой зависимостями. Чтобы решить задачу X, пользователю сначала необходимо решить все её зависимости. Таким образом задачи образуют направленный ацикличный граф. Если у задачи нет зависимостей, то её можно решать сразу.

Граф зависимостей – это направленный ацикличный граф, такой, что пользователь может решать задачу X, только если решены все задачи, от которых X зависит.

Отличительные особенности проекта:

1. **Геймификация** – это использование игровых механик для повышения активности и привлечения новых пользователй.

2. **Граф задач** – организация задач в виде направленного ацикличного графа, благодоря чему пользователь может решать задачи не в линейном порядке, изучая только наиболее интересные ему темы.

3. **Глоссарий** – все биологические и алгоритмические термины из задач собраны в специальном глоссарии который можно испольльзовать в качестве справочного метериала.

4. **Классы** – можно создать свой набор задач из существующих и использовать Розалинд в качестве учебного пособия в курсе по биоиформатике.

5. Возможность решать задачи на любом языке программирования.

Сейчас в Розалинд более 7000 пользователей. Классы Розалинд успешно используются для обучения студентов биоинформатике в нескольких университетах мира.

Проект активно развивается. Ближайшими целями являются добавление задач по новым тематикам, отличным от биоинформатики, и создание учебных материалов по принципу Википедии.

1. Постановка задачи

Одной из самых важных и трудоёмких частей проекта является создание задач. От качества задач зависит заинтересованность в проекте обычных пользователей. Удобство их создания напрямую влияет на возможность сотрудничать с различными университетами и другими образовательными организациями, а также на эффективность пользователей в качестве авторов задач.

Создавать новые задачи трудно по многим причинам. Они делятся на две группы: авторские трудности при создании условий и технические трудности при создании решений, тестов и интеграции задач. Среди второй категории можно отметить следующие.

1. Многие задачи вычислительной биологии весьма трудоёмки. И если пользователи могут тратить много времени на решение одной задачи (лимит времени по умолчанию - 5 минут на задачу), то на сервере необходимо решать задачи быстро. Поэтому необходимо использовать ассимптотически оптимальные алгоритмы. Более того, зачастую генерация датасета и проверка правильности ответа пользователя являются не тривиальными задачи.

   Генерировать датасеты сложно, потому что случайный датасет, зачастую, не очень подходит. Во-первых, он может не выявить некоторые типичные ошибки пользователя. Во-вторых, он может оказаться слишком простым даже для не оптимального алгоритма. Хорошим примером могут служить вариации задачи о поиске подстроки в строке. Типичная ошибка, которую допускают пользователи – пропуск перекрывающихся вхождений. При этом в случайном датасете таких вхождений почти наверняка не будет. Более того, там не будет и длинных совпадающих подстрок, поэтому даже метод перебора будет работать за линейное время.

   Основная сложность при проверке ответа заключается в том, что не всегда правильный ответ – единственный.

2. Требуется работать со специфическими биоинформатическими форматами данных. Существуют библиотеки, которые помогают с этим справится. Это, прежде всего, BioPython [1]. Однако эти библиотеки не всегда хорошо документированны, эффективны и корректно работают.

3. Разработка ведётся на языке Python. Это современный интерпертируемый язык, удобный для разработки. К сожалению, за скорость разработки приходится платить быстродействием, зачастую недостаточным для решения сложных задач вычислительной биологии за приемлемое время.

4. Задачи связанны между собой, поэтому необходимо следить не только за тем, чтобы каждая задача по отдельности работала, но и за тем, чтобы вместе они были согласованы.

Поэтому были поставлены следующие задачи:

1. Разработать набор задач по биоинформатике.
2. Ускорить подсистему проверки и тестирования задач.
3. Разработать средства для упрощения разработки задач.
4. Предложить возможные пути улучшения API подсистемы проверки и тестирования задач.

2. Результаты

2.1. Создание задач

На текущий момент на сайте rosalind.info опубликовано 123 задачи, из которых 80 созданы мной. Задачи сильно варьируются по сложности и тематике.

Например, в первой задаче, DNA, требуется посчитать количество букв A, T, G, C в строке длинной около ста букв. Эта задача рассчитана на то, что даже человек, не знакомый с программированием, справится с ней.

В задаче QRTD, которая является одной из самых сложных, требуется найти квартетное расстояние между двумя неукоренёнными бинарными деревьями. Квартет – это четвёрка терминальных вершин A, B, C, D такая что существует ребро, разделяющее пары A, B и C, D. Подробно решение это задачи рассмотрено в [3]. Задача QRTD предлагалась на чемпионате СПбГУ по программированию.

Среди интересных структур данных и алгоритмов, использовавшихся для создания задач, стоит отметить следующие.

1. Динамическое программирование. Одной из классических задач биоинформатики является задача выравнивания геномных последовательностей. В случае двух последовательностей её можно эффективно решить при помощи алгоритма Смитта-Ватермана [6].

2. Суффиксные деревья. Так как во многих задачах приходится иметь дело с последовательностями символов, то суффиксные деревья часто оказываются весьма эффективной структурой данных. Например, с их помощью можно эффективно измерять сложность генома [5].

3. Графы Де-Брюйна CITE. Задача сборки генома сводится к поиску определённого пути в графе. Если при использовании графа перекрытий приходится искать Гамильтонов путь, т.е. решать NP сложную задачу, то граф Де-Брюйна сводит задачу сборки генома к полиномиальной задаче нахождения эйлерова пути.

4. Хэш-функции на деревьях. В вычислительное филогении приходится иметь дело с эволюционными деревьями. Одна из задач сравнения таких деревьев решается при помощи хорошо определённой хэш-функции.

2.2. Задачи на C++

Как уже было сказанно, некоторые задачи обладают большой вычислительной трудоёмкостью, поэтому решения, написанные на Python, требуют непростительно много ресурсов.

Чтобы решить проблемы с производительностью, была добавлена возможность использовать задачи и библиотеки, написанные на языке C++ с помощью SWIG CITE. Это не только существенно ускорило некоторые задачи, но и позволило людям, не знающим Python, но владеющим C++, также создавать задачи. В таблице приводится сравнительное время работы двух задач на C++ и Python, для которых разница в производительности особенно существенна.

Задача	Python	C++
GAFF	20 с	0.01 с
QRTD	163 с	0.03 с

Всего более 30 задач были ускоренны при помощи C++.

2.3. Улучшение тестирования

При разработке новых задач и общих библиотек важно постоянно проводить регрессионное тестирование, для предотвращения возникновения новых ошибок.

Тестирование в данном случае обладает некоторыми особенностями.

1. Надо проверять большое количество относительно независимых однотипных задач.

2. Ошибки бывают разной значимости. Большинство задач, находящихся в стадии черновика, тесты не проходят, но это не является ошибкой. Однако, если тестирование не проходит опубликованная задача, то это серьёзная проблема, требующая решения.

3. Тестирование всего набора задач занимает много времени, что не удобно для разработки.

Проблема с производительностью была решена при помощи распараллеливания тестов на несколько процессов. Было использовано многопроцессное, а не многопоточное решение, потому что global interpreter lock в Cpython не даёт эффективно использовать многопоточность. Для распараллеливания была использована библиотека joblib.

В результате, время тестирование упало с 8 до 2 минут (на 4 CPU).

Также были улучшены диагностические сообщения тестирующей системы, позволяющие точнее локализовать большее количество ошибок и потенциальных проблем, как, например, слишком большие датасеты или отсутствие дополнительного решения для перепроверки.

Интересным опытом оказалось использование пользовательского кода для проверки авторских решений и алгоритмов генерации датасета. Однако пока не удалось автоматизировать подобные проверки.

2.4. Валидация условий задач

Условие задачи это текстовый файл в формате markdown со специфическими для Розалинд расширениями синтаксиса. Помимо, собственно, условия, этот файл явно или неявно включает в себя метаинформацию о задаче: её статус, название, использованные термины, зависимости.

Создание условий задач – неизбежно трудоёмкий процесс, так как условия невозможно полностью проверять в автоматизированном режиме. Однако, возможность автоматически обнаруживать некоторые ошибки в файлах условий существенно упрощает этот процесс.

Были реализован инструмент для проведения проверок и локализации ошибок. Он позволяет обнаруживать

1. Наличие циклов в графе зависимостей.
2. Наличие транзитивных зависимостей. Если задача A зависит от B и C, но при этом B также зависит от C, то зависимость AC является излишней.
3. Ссылки на неопределённые в глоссарии термины.
4. Синтаксические ошибки.

2.5. Улучшение API задач

В результате работы были обнаружены некоторые существенные недостатки в текущем API задач, в их числе:

1. Невозможность выдавать пользователю детальное сообщение об ошибке в решении.
2. Неэффективная проверка решений пользователя.
3. Излишне сложное тестирование отдельных задач.

Было предложен новый API устраняющий эти недостатки. В данный момент он реализуется в новой версии платформы.

Заключение

Были реализованны задачи по биоинформатике для проекта Розалинд.

На основании опыта, полученного при реализации, была улучшена тестирующая подсистема и повышена эффективность разработки задач.

Достигнут значительный прирост производительности на наиболее трудоёмких задачах.

Разработка задач стала более простой благодоря автоматическим проверкам задач.

Результаты работы в настоящее время используются при разработке двух новых наборов задач, один из которых будет посвящён классическим алгоритмам, а другой создаётся специально для курса по биоинформатике на Coursera CITE.

Полученный опыт используется при разработке новой версии платформы.

Список литературы

[1] Biopython: freely available Python tools for computational molecular biology and bioinformatics / Cock PJ, Antao T, Chang JT et al. // Bioinformatics.

[2] Code Jam. — https://code.google.com/codejam/.

[3] Fast calculation of the Quartet Distance between trees of arbitrary degree / C. Christiansen, T. Mailund, C. N. S. Pedersen et al.

[4] Project Euler. — http://projecteuler.net/.

[5] Sequence complexity profiles of prokaryotic genomic sequences: a fast algorithm for calculating linguistic complexity / Troyanskaya OG, Arbell O, Koren Y et al. // Bioinformatics.

[6] Smith Temple F., Waterman Michael S. Identification of Common Molecular Subsequences // Journal of Molecular Biology.

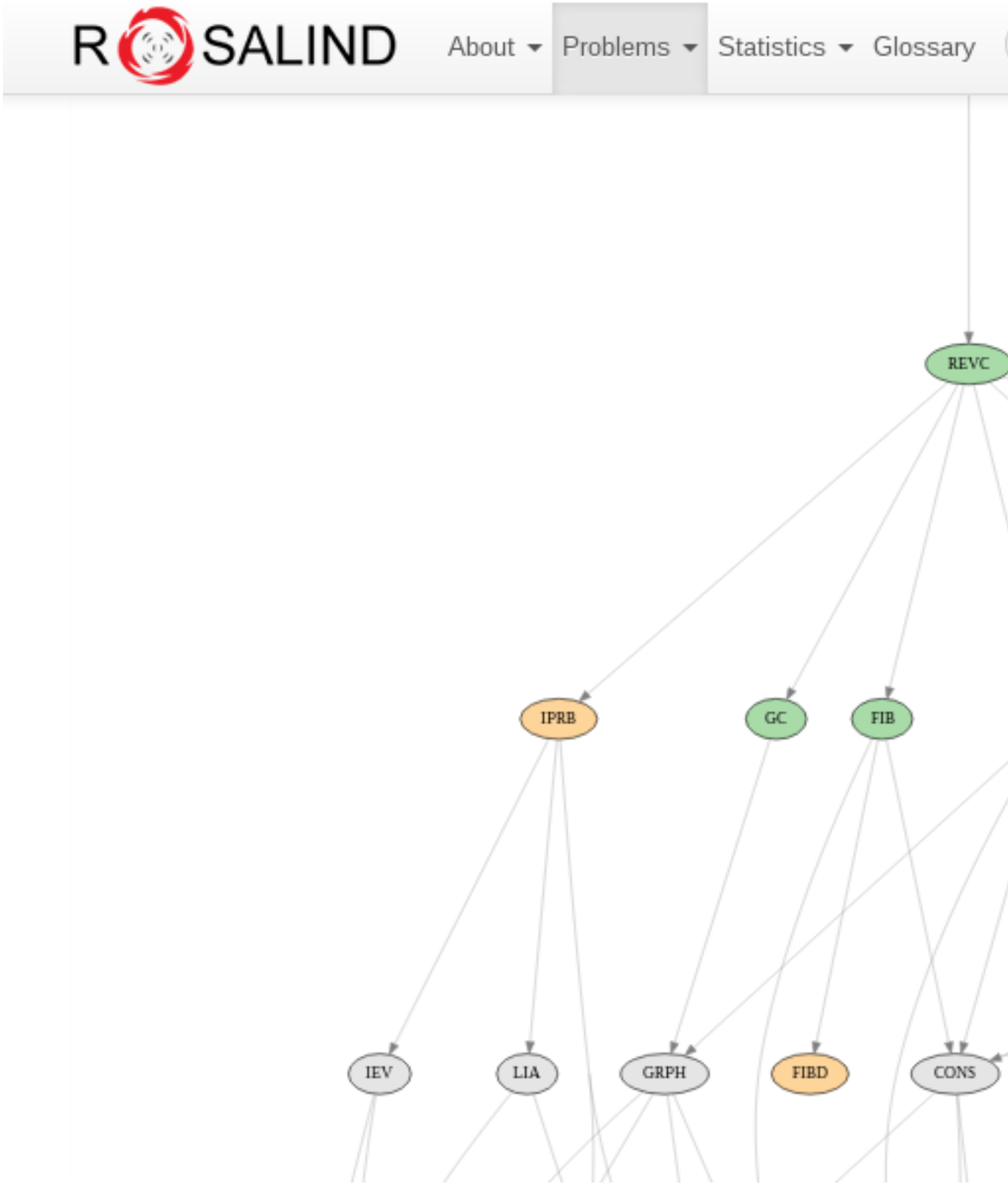

Рис. 1: Граф зависимостей