Открытые проблемы по Веб-алгоритмам
Лекция № 11 курса
«Алгоритмы для Интернета»

Юрий Лифшиц*

7 декабря 2006 г.

Содержание

1. Как поставить хорошую задачу?

Критерии выбора задачи

Как говорил Алексей Николаевич Крылов: «Поставить задачу — это уже наполовину ее решить». Поэтому необходимо уметь выбирать такие задачи, которые было бы возможно и интересно решать. Каждый в данном случае может руководствоваться своими собственными критериями, и таких критериев огромное множество. Вот наиболее распространенные из них:

- непосредственная связь с разработкой новых технологий;
- знакомство с областями приложений;
- взаимосвязь нескольких научных направлений;
- сводимость других задач к данной;
- возможность проверить правильность разрабатываемого решения;
- новизна.

*Законспектировал Глеб Рыбаков.

Помимо этих критериев можно предложить еще несколько не таких известных, но, несомненно, заслуживающих внимания:

- технологическая сложность;
- знаменитость и возраст задачи;
- известность автора задачи;
- патентность задачи (поскольку именно патенты являются непосредственным результатом работы ученого в компании).

Формат представления задачи

Представление задачи в правильном формате имеет первостепенное значение, поскольку именно на этом этапе можно рассмотреть задачу целиком и адекватно оценить сложность, ресурсоемкость задачи и т.д. Рассмотрим один из возможных форматов описания проблемы в целом:

- область (технология) для использования;
- формализация описания;
- вовлеченные научные направления;
- близкие классические результаты;
- план исследований.

2. Крупномасштабная фильтрация (Large-scale filtering)

Каждый день в мире появляется огромное количество новостей. Предположим, что мы хотим научиться предлагать пользователям новости, которые были бы им интересны. Сколько людей — столько и мнений, поэтому надо научиться анализировать предпочтения отдельных пользователей и в зависимости от них предлагать им соответствующие ресурсы. Изучим эту задачу в соответствии с поставленным планом.

Технологическая проблема

У нас есть сущности двух типов: пользователи и новости, и есть их системы описания. Система описания пользователей — это их предпочтения и пожелания: например, любимые авторы, ключевые слова, метки (темы) и т.д. Система описания новости содержит текст, тематику, рейтинг, рекомендации и т.д. Поставим себе такую задачу: для каждого пользователя сопоставить десять наиболее интересных для него сообщений. Примеры таких систем уже есть: Google News, Google Reader, Yandex Lenta, Livejournal Friends... но все они далеко не идеальны.

Формализация

Представим описанную систему в виде геометрической модели. Опишем систему предпочтений каждого участника красным нормализованным вектором (точкой на сфере) в n-мерном пространстве признаков. Аналогично описание новости будем представлять нормализованным вектором (точкой на сфере) синего цвета в том же пространстве. Возникает вопрос: каким образом теперь сопоставить описания новостей предпочтениям пользователей? Для этого используем косинусную меру (скалярное произведение векторов), которая будет определять, насколько новость «подходит» пользователю. Несомненно, можно было бы посчитать скалярные произведения для всех пар «пользователь–новость» и вычислить для каждой персоны десять наиболее подходящих новостей. Но количество сообщений и людей велико настолько, что такое вычисление для всего Интернета заняло бы ни одни сутки только для одного признака! А такие вычисления придется выполнять каждый раз, когда появляются новые новости. Поэтому необходимо

провести (пред)вычисления на исходных векторах так, чтобы впоследствии можно было очень быстро выполнять описанные операции.

Вовлеченные направления

Рассмотрим, какие технологии могут потребоваться для решения поставленной задачи.

- Классификация текстов. Определяет формат представления новостей. Позволяет разбивать новости на группы.
- Вычислительная геометрия. Алгоритмы поиска ближайших соседей. Поскольку мы решаем задачу в геометрической модели, для поиска схожих по тематике новостей для данного сообщения необходимо определить его ближайших соседей в пространстве векторов.
- Структуры данных. Скорее всего, для хранения такого большого объема информации и предоставления быстрой навигации между векторами (новостями и пользователями) нам потребуется разработать новые или изменить уже известные структуры данных в рамках нашей задачи.
- Архивирование (редкие векторы). Количество векторов огромно, поэтому невозможно хранить их в явном виде. Поскольку каждый вектор (пользователь или новость) обычно характеризуется лишь небольшим подмножеством признаков из их общего количества, координаты векторов будут содержать большое количество нулей, чем необходимо воспользоваться для эффективного сжатия.
- Линейная алгебра (сингулярное разложение). Помимо хранения векторов в сжатом виде, необходимо достаточно быстро производить различные операции над ними, будь то скалярное произведение или сортировка.
- Модели распределенных вычислений. Позволит увеличить скорость выполнения.
- Квантовые алгоритмы? Поскольку основной задачей является быстрая фильтрация большого количества векторов по большому количеству параметров (меток), вполне возможно, что здесь будут иметь место квантовые алгоритмы.

Алгоритм Клейнберга

Рассмотрим один из наиболее известных алгоритмов для поиска ближайших соседей. Есть n точек в d-мерном пространстве (n и d огромны), необходимо записать их в некоторую удобную структуру данных, чтобы можно было быстро, возможно с некоторыми ошибками, находить ближайшие точки к данной за время, полиномиальное относительно $d \log n$. Общая схема алгоритма такова.

1. Выбираем k случайных контрольных векторов с нормой 1.
2. Проектируем на каждый из них все имеющиеся у нас точки. Таким образом, точки выстраиваются в некотором порядке.
3. При проектировании новой точки мы рассматриваем совокупность интервалов на контрольных векторах, в которые она попадает, и находим ближайшую точку для каждой системы согласованных интервалов.
4. Для обработки запроса необходимо найти соответствующую систему интервалов и по ней обратиться к структуре данных.

В результате могут возникнуть и ошибки, но вероятность их появления можно контролировать.

План исследований

Рассмотрим основные этапы.

1. Построить быстрый алгоритм фильтрации всех новостей для всех пользователей. Хотелось бы построить такой алгоритм фильтрации на всех пользователях и отдельно отсортировать все новости так, чтобы впоследствии можно было быстро строить соответствия.

2. Найти наиболее эффективные структуры данных для хранения пользователей/новостей.

3. Изучить динамические аспекты. Описания новостей и пользователей быстро меняются, поэтому необходимо, чтобы структуры были легко расширяемыми и модифицируемыми и обновление взаимосвязей не требовало много времени.

4. Разработать систему предотвращения спама в системе персонального сбора новостей. Контролировать, например, возможность некоторого круга лиц «продвигать» некоторую группу новостей.

5. Как уравнять в правах свежие и старые новости? Предоставить возможность развития новым новостям, прежде чем они наберут достаточную популярность для нормальной конкуренции со старыми.

3. Распространение меток (Tag Propagation)

Технологическая задача

Рассмотрим задачу начального распространения ключевых слов на весь Веб. Как известно, Веб состоит из миллиардов страниц, а люди используют для классификации миллионы ключевых слов (меток). Необходим быстрый алгоритм, который будет подбирать метки для произвольной страницы. Зачем это надо?

- Настройка рекламных объявлений. Будущее рекламы за Вебом, использование меток позволит сделать рекламу более широкой, но и избирательной.
- Аннотирование результатов выдачи в поисковых системах.
- Кластеризация результатов (разбиение на группы по меткам, «склеивание» дублей).

Формализация

Есть два подхода для распространения меток. Первый заключается в распространении по контенту, а второй — по графу ссылок. Первый подход состоит в следующем: если новая страница похожа по словам на старую, то ей должны соответствовать те же метки. Мы рассмотрим второй подход: если старая страница ссылается на новую, то надо использовать метку старой страницы для текущей. Рассмотрим общий план действий этого подхода.

- Задан граф ссылок. Рассматриваем одну метку. Каждой вершине графа ссылок сопоставим tag-rank (TR) — величину, определяющую, насколько эта вершина подходит под эту метку.
- Зафиксируем начальные значения TR. Пусть для исходных помеченных страниц $T_0(i) = 1$, а для остальных — $T_0(i) = 0$. Заметим, что начальные значения могут определяться пользовательскими предпочтениями — сколько пользователей сохранило данную страницу с данной меткой. Возможно, стоит сделать нормализацию по сумме значений TR.
- Рассмотрим предел по рекуррентным соотношениям

$$T_k(i) = T_{k-1}(i) + \alpha \sum_{j \in J(i)} \frac{T_{k-1}(j) - T_{k-2}(j)}{Out(j)},$$

где $J(i)$ — множество вершин, ссылающихся на i, а $Out(j)$ — общее количество исходящих ссылок из j. TR будет расти со временем. На k-й секунде TR вершины будет определяться суммой ее предыдущего значения и частью изменения TR на $(k-1)$-й секунде у вершин, на нее ссылающихся. Эта часть и определяется коэффициентом «неиспарения» α.

Рассмотрим работу алгоритма на следующем примере.

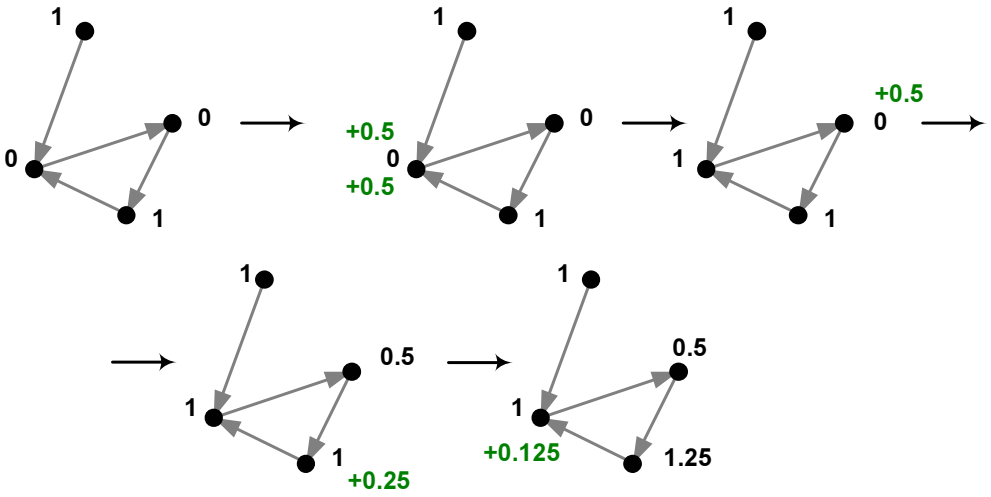

Рис. 1. Пример «перемещения» TR со временем для коэффициента $\alpha = 0.5$.

Вовлеченные направления

- Структуры данных.
- Архивирование (разреженные множества).
- Численные методы. Нас интересует скорость сходимости TR в вершинах в случае распространения его по указанной выше формуле.
- Алгоритмы ранжирования и PageRank. Необходимо грамотно выбирать начальное распределение меток на основе пользовательских предпочтений.

Различные подходы

Вычислять TR можно по-разному. Сравним две рекурсивные формулы.

$$PR_k(i) = \frac{\varepsilon}{N} + (1-\varepsilon)\sum_{i=1}^{n}\frac{PR_{k-1}(T_i)}{C(T_i)},$$

$$T_k(i) = T_{k-1}(i) + \alpha\sum_{j\in J(i)}\frac{T_{k-1}(j) - T_{k-2}(j)}{Out(j)}.$$

Рассмотрим первую формулу. Если следовать ей, то с вероятностью ε мы «перепрыгиваем» на случайный сайт, а с вероятностью $1-\varepsilon$ мы переходим по ссылкам с одного из предыдущих. Этот подход отличается от второго тем, что здесь нет системы накопления — на каждом шаге все ранги куда-то переходят. Во втором случае все ранги уже закреплены и лишь небольшая их часть еще «путешествует» по графу ссылок, приближая значения TR к пределу в вершинах. Понятно, что если система накопления отсутствует, то используя любое начальное распределение, мы получим PageRank вне зависимости от исходной расстановки тегов, так как предельные вероятности не зависят от исходного распределения меток. Все рано или поздно стабилизируется к самым популярным сайтам вне зависимости от стартовой точки. Второй подход позволяет сохранить эту зависимость, для чего и используется система накопления.

План исследований

1. Определить формулы распространения меток. Предложенная формула не обязательно является оптимальной. Необходимо рассмотреть различные варианты распределения TR и выбрать наиболее оптимальный.

2. Построить алгоритм быстрой предварительной обработки учебной коллекции и on-line аннотирования. Возможно, мы не будем проводить вычисления для всего Веба сразу, а начнем обучение на некоторой небольшой коллекции, вычисляя значения TR для новых страниц по требованию: будем перемещаться от непомеченной страницы к ближайшей помеченной, после чего можно будет вернуться назад и построить TR для непомеченной страницы.

4. Выявление структур (Structure discovery)

Посмотрим на метки, которые мы используем. Необходимо построить некоторую систему отношений (иерархию, например) между ними. Это позволит связать вместе схожие по смыслу метки в единые группы.

Технологическая задача

Существуют огромные коллекции данных: истории покупок, звонков, поисковых запросов, RSS-запросов, социальные сети... на данный момент такая информация собирается различными спецслужбами. Но не только, например, компания Google ведет сбор информации о запросах пользователей. Как можно извлечь пользу из этой информации? Ситуация обычно усугубляется тем, что эти данные очень разрозненные, и почти невозможно в них осуществлять навигацию. Конкретные приложения этой задачи:

- Улучшение визуализации данных. Представление информации в понятном виде.
- Упрощение навигации в больших объемах информации.
- Решение проблемы синонимов. Группировка однотипной информации.

Формализация

- У каждой метки есть ее описание. Например, это могут быть все сайты, которые на нее сохранены. Мы хотим построить дерево меток, учитывая синонимы и отношение «общее–частное». При этом, возможно, имеет смысл ввести два типа вершин: вершины «И» (родитель — «программирование», а дети — Microsoft, C++ и т.д.) и вершины «исключающее ИЛИ» (животное — это или кот, или собака).
- Принцип оптимального дерева. Хотим строить оптимальные деревья в смысле минимальности отклонения от идеала. Идеальное дерево — то, в котором дети различных вершин не пересекаются, множество родителя содержит множества всех детей (совокупность сайтов, помеченных родителем, должна содержать все сайты, помеченные ребенком).

Вовлеченные направления

- Вычислительная биология (алгоритмы филогенетики). Пусть у нас есть генотипы различных живых существ, мы хотим восстановить дерево и генотипы общих предков.
- Приближенные алгоритмы. Вполне вероятно, что вычислять точное оптимальное дерево будет очень долго. Потребуется алгоритм, который позволит строить «приближенно–оптимальное» дерево.
- Добыча данных (data mining, web mining). Наша задача полностью соответствует названию этой науки. Мы хотим из неупорядоченных данных извлечь какие-то закономерности и информацию.

Алгоритм Фитча

Рассмотрим алгоритм Фитча, который позволяет найти оптимальное дерево с минимальным количеством «мутаций» (изменений от предков к потомкам).

- Отдельно работаем для каждой позиции.
- Для каждой внутренней вершины составим список разумных кандидатов S_v. Кандидаты выбираются из соответствующих элементов на текущей позиции у детей.
- Проход снизу вверх: если w — родитель u и v, и $S_u \cap S_v = \emptyset$, то $S_w = S_u \cup S_v$, иначе $S_w = S_u \cap S_v$.
- Проход сверху вниз: выбираем символ для корня, дальше берем символ родителя если, он входит в множество ребенка, иначе — произвольный символ из множества ребенка.

Продемонстрируем работу алгоритма на примере построения дерева для кодов ДНК.

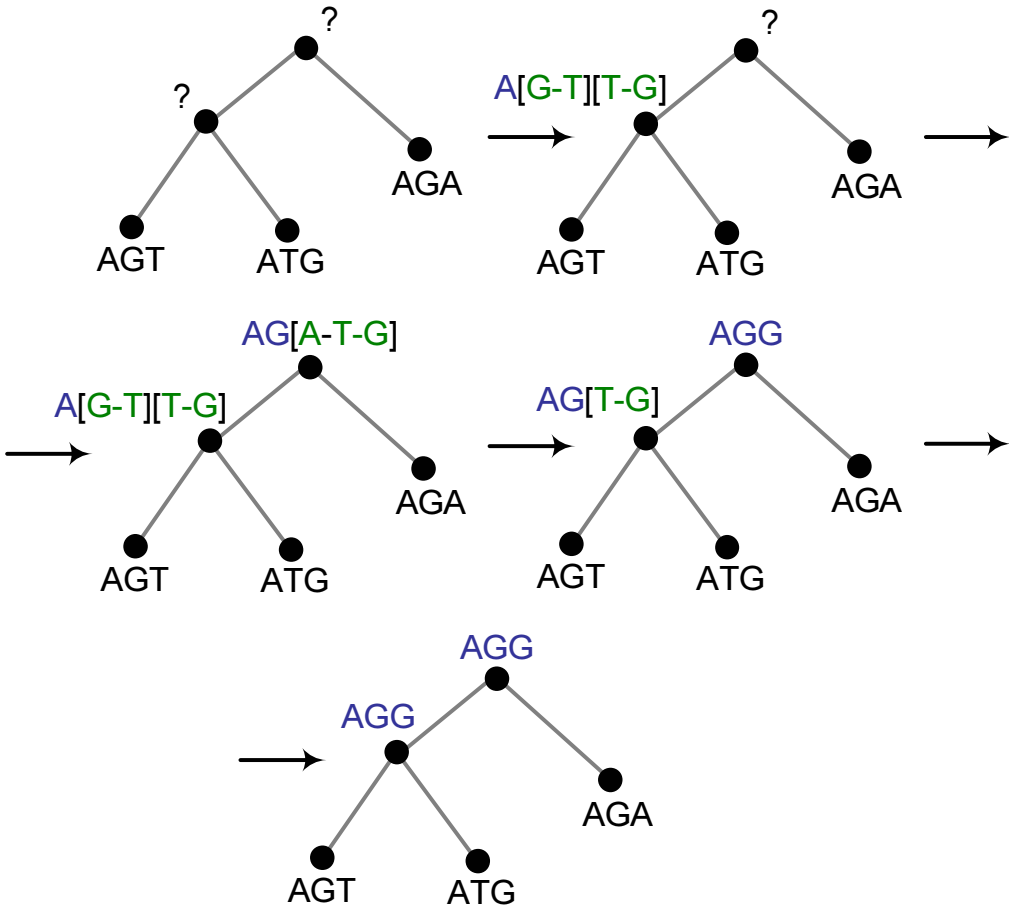

Рис. 2. Пример построения дерева для кодов ДНК с оптимальным количеством мутаций. Черным цветом помечены исходные символы на листьях. В квадратных скобках приведены и выделены зеленым цветом списки возможных кандидатов для каждой позиции. Синим цветом помечены символы, выбранные в ходе работы алгоритма.

План исследований

1. Выбрать формат представления метки и определить критерии идеальной иерархии меток.
2. Найти быстрый алгоритм построения оптимальной иерархии.
3. Изучить взаимосвязи с алгоритмами филогенетики.

Задача

Пусть $|v| < |u|$, докажите, что с вероятностью не менее $\frac{1}{2}$ для случайного вектора r выполнено $r \cdot v < r \cdot u$.

Итоги

Итак, мы рассмотрели следующие технологические задачи: персональный сбор новостей, использование больших объемов данных, автоматическое аннотирование больших объемов данных. Во всех этих задачах мы столкнулись с основной алгоритмической проблемой: алгоритмы выполняются на миллиардах объектов, поэтому необходимы эффективные структуры данных и быстрая обработка запросов. Наивные алгоритмы «каждый–каждый» не обладают достаточной производительностью.

Источники

[1] Jon Kleinberg. Two algorithms for nearest-neighbor search in high dimensions
http://citeseer.ist.psu.edu/kleinberg97two.html

[2] Ron Shamir. Phylogenetics
http://www.cs.tau.ac.il/~rshamir/algmb/01/scribe08/lec08.pdf

[3] AN Langville, CD Meyer. Deeper Inside PageRank
http://meyer.math.ncsu.edu/Meyer/PS_Files/DeeperInsidePR.pdf

[4] Страница курса
http://logic.pdmi.ras.ru/~yura/internet.html