Команда 5. Описание решения

Copyright

В алфавитном порядке:

- Агата Аткарская
- Дмитрий Джулгаков
- Дмитрий Панин
- David Stolp

Git-репозиторий с исходными кодами доступен по адресу https://www.assembla.com/spaces/yssdc/.

Алгоритмическая часть

Мы использовали подход динамического программирования. Пусть максимальная длина гена, который мы хотим найти, составляет L, а длина входной строки – N. Введем величину $d_{i,j}$ – максимальное количество базовых пар в некоторой корректной вторичной структуре на отрезке $[i,j]$. Отрезок $[i,j]$ будет являться кандидат-геном, если для него будут выполняться 3 условия указанных в условии (ограничение на fPairedMin, Fat и длину). При этом $fPairedMin = \frac{d_{i,j}}{j-i+1}$. Научимся вычислять эту динамику.

Очевидно, что $d_{i,i} = 0$. Рассмотрим возможные переходы между состояниями:

1. Вторичная структура может состоять из двух отдельных вторичных структур расположенных рядом: $d_{i,j} \geq d_{i,k} + d_{k+1,j}, \forall k, i \leq k < j$. Заметим, что это покрывает и случай, когда крайние символы на отрезке $[i,j]$ не входят во вторичную структуру. В этом случае, минимум будет достигаться для $k = i$ или $k = j - 1$.
2. Края отрезка $[i,j]$ составляют валидный helix: $d_{i,j} \geq d_{i+k,j-k} + k$, для всех k, таких что пары $\{(i,j),(i+1,j-1),\dots,(i+k-1,j-k+1)\}$ являются комплементарными. В этом переходе нужно также проверить ограничение на минимальную и максимальную длину hairpin.

Поскольку переходы между состояниями зависят только от состояний с меньшей длиной, можно вычислять ее в порядке увеличения j и $i - j$. Поскольку мы заинтересованы только в генах с длиной не более L символов, необходимо вычислять значения динамики только для состояний с $j - i + 1 \leq L$.

Мы вычисляем N*L состояний, и для вычисления состояния требуется проверить порядка L вариантов. Таким образом, сложность составляет $O(N * L^2)$.

Рассмотрим, как оптимизировать данную динамику. Заметим, что переход 1 является избыточным. Действительно, если оптимальная вторичная структура на отрезке $[i,j]$ состоит из нескольких отдельных фрагментов, каждый из которых является корректной вторичной структурой, достаточно проверить только такие индексы k, что оптимальным на отрезке $[k+1,j]$ является вторичная структура, состоящая из одного фрагмента.

Под фрагментом мы здесь понимает такую вторичную структуру, которая содержит единственный внешний helix. Другими словами, такой фрагмент был получен с помощью перехода 2. Запомним для каждой позиции j список $F(j)$ всех фрагментов с правым концом j. Тогда в переходе 1 достаточно рассматривать только такие k, что $(k+1,j) \in F(j)$. Таким образом, переход 1 будет выполняться за

$O(|F(j)|)$. Также нужно рассмотреть 2 дополнительных перехода, покрывающие случай, когда крайние символы отрезка $[i,j]$ не входят в оптимальную структуру. А именно рассмотреть значения $k = i$ и $k = j - 1$.

Рассмотрим также как ускорить обработку перехода 1. Минимальный размер helix-а составляет 4. Если для отрезка $[i,j]$ есть внешний helix длиной $p > 4$, то для отрезка $[i+1, j-1]$ существует внешний helix с длиной $p - 1$. Будем запоминать во вспомогательном массиве $d^2_{i,j}$ значения динамики, полученные с помощью последнего перехода вида 2. Тогда достаточно рассмотреть только внешний helix размера 4 и проверить случай $d^2_{i,j} > 0$. Проверку helix-а длиной 4 можно ускорить технически, предпросчитав коды всех 4 последовательных символов. Итого, сложность перехода 2 составила $O(1)$.

Итоговая сложность полученной динамики составляет $O(N * L + L * \sum_1^N |F(i)|)$. Оценим $|F(i)|$. Поскольку минимальная длина helix составляет 4 символа и для каждого символа существует только один парный, вероятность того, что для фиксированного j и произвольного i отрезок $[i,j]$ будет содержать внешний helix равна $\frac{1}{4^4} = \frac{1}{256}$. Поскольку мы рассматриваем отрезки с длиной не более L, $E(|F(i)|) = \frac{L}{256}$. Итоговая сложность ускоренной динамики составляет $O(N * L * (1 + \frac{L}{256}))$. Асимптотически сложность решения не улучшилась, однако константа алгоритма была уменьшена значительно.

Технические замечания:

- Матрицу динамики можно хранить в массиве N*L, поскольку $j - i < L$. На самом деле достаточно массива L^2, поскольку мы обращаемся только к последним L строкам.
- В процессе вычисления матрицы динамики нужно запоминать выбранные переходы, чтобы затем восстанавливать ответ.
- Для восстановления ответа решение сохраняется в виде списка вложенных helix-ов, а не отдельных пар.
- Для быстрого вычисления Fat можно предпросчитать частичные суммы числа символов A и T.

Проверив все индексы $[i,j]$ на 3 условия для кандидат-генов, мы получаем набор кандидат-генов. Такой набор может быть достаточно больший, поскольку отдельные гены могут перекрывать.

Вторая фаза алгоритма занимается выбором подмножества непересекающихся кандидат-генов с максимальной суммой. Она реализована классическим алгоритмом оптимального расписания для фиксированных времен заданий и одного ресурса. Ее сложность составляет $O(N + C)$ или $O(C \log C)$, где C – количество кандидат-генов.

Распределенная часть

1. Сначала мы локально режем исходную строчку на блоки длины 10^5, перекрывающиеся по $3 * 10^3$ символов (чтобы не потерять гены на краях при склейке).

2. Складываем получившийся файл в HDFS.

3. Запускаем распределенный MapReduce процесс, в котором каждый Mapper считает динамику на предложенном ему блоке (более подробно об этом см. Алгоритмическую часть). Из соображений быстродействия Mapper-ы были написаны на C++, поэтому для их запуска используется hadoop streaming.

4. Java программа team5.jar->io.Hadoop анализирует полученный файл и выбирает из генов, полученных на предыдущем шаге, непересекающееся подмножество с максимальным количеством base pairs.

Нереализованные идеи

К сожалению, мы не успели реализовать стратегию динамического выбора окна L. Идея в следующем:

- Запустим динамику с маленькой длиной окна, скажем L = 1024. Такая динамика отработает очень быстро
- Найдем участки строки, на которых встречается много кандидат-генов и запустим динамику с большей длиной окна на этих участках.
- Будем повторять эти шаги, пока не будет достигнут лимит по времени.

Random notes (just for fun)

- Начальная реализация динамики была написана на Java, ее останки до сих пор хранятся в team5.jar. Переписав динамику на c++ скорость увеличилась в несколько раз (и код написанные за полтора дня был просто выкинут ☹)
- Для командной разработки был поднят git-репозиторий, суммарное количество коммитов за 3-х дневный период составило 74 штуки.
- У нас есть стойкое подозрение, что все команды реализовали один и тот же алгоритм, и победитель будет определен по выбору максимальной длины гена L. Мы побоялись ставить значение L > 2048 чтобы не схлопотать тайм-лимит. Что ж, по крайней мере мы надеемся посоревноваться за самое красивое описание решения ☺

We had a lot of fun! Thank you!