Оптимизация программного кода алгоритма быстрого поиска похожих образцов в метагеномных базах данных

Николай Ромащенко

Институт биоинформатики

13.09.2016

Цель и задачи

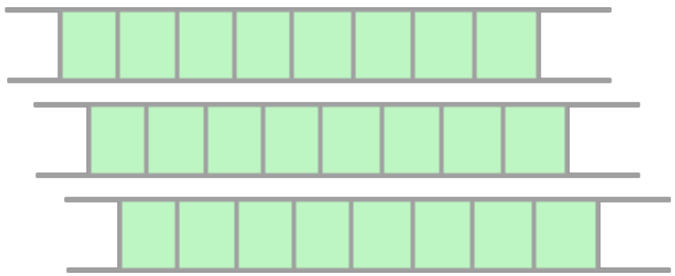
Цель

Подготовка кода и материалов к публикации

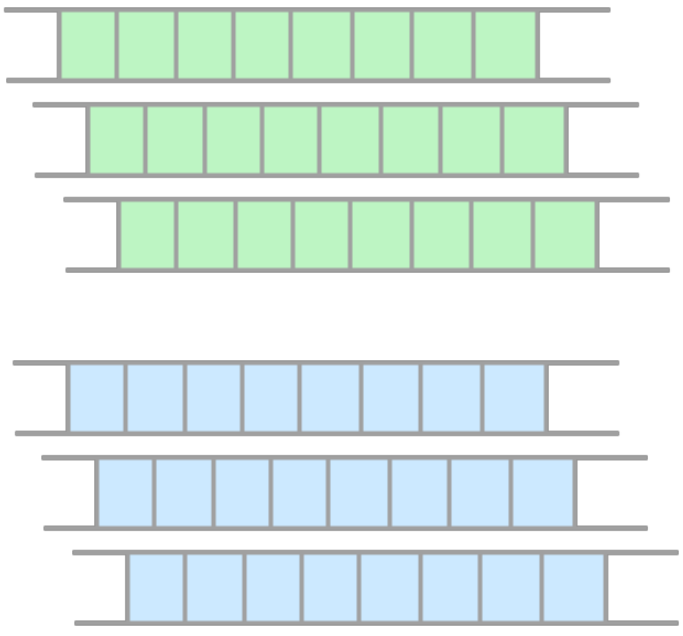
Задачи

- Оптимизация кода
- Рефакторинг
- Тестирование на крупных датасетах

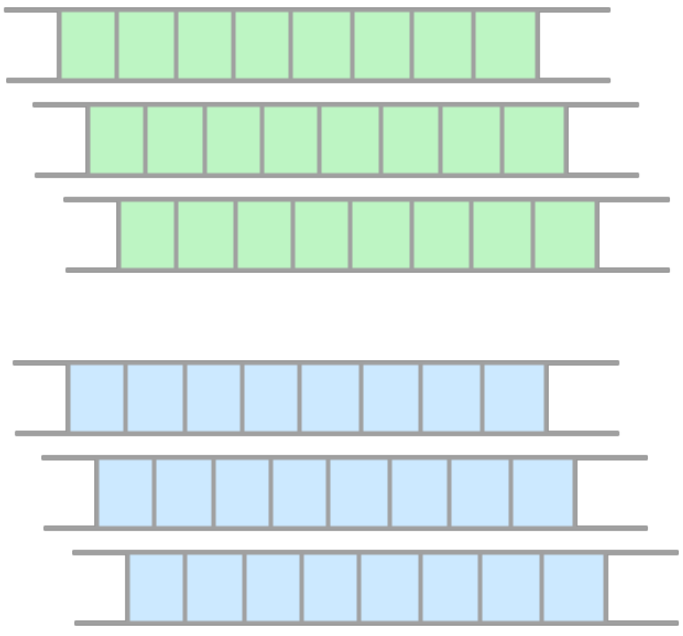
Подсчет k-меров

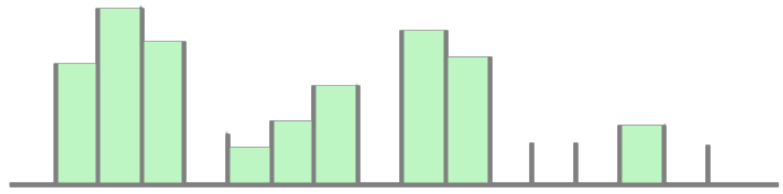


Подсчет k-меров

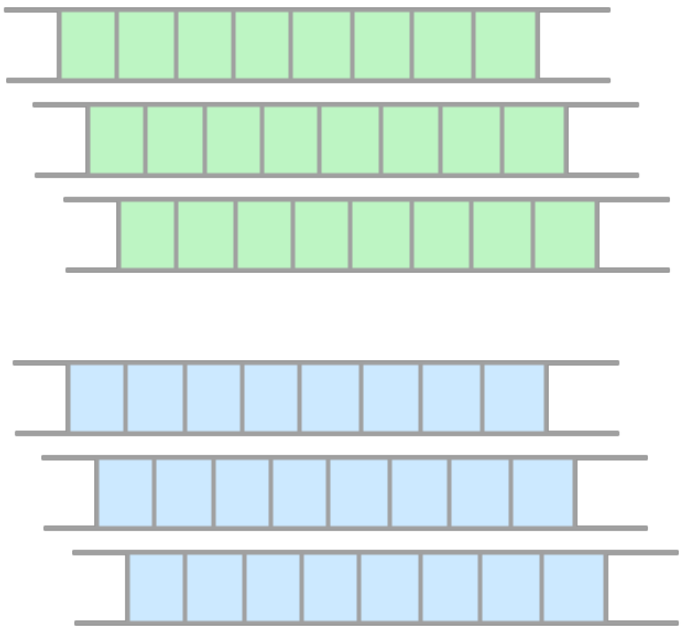


Подсчет k-меров

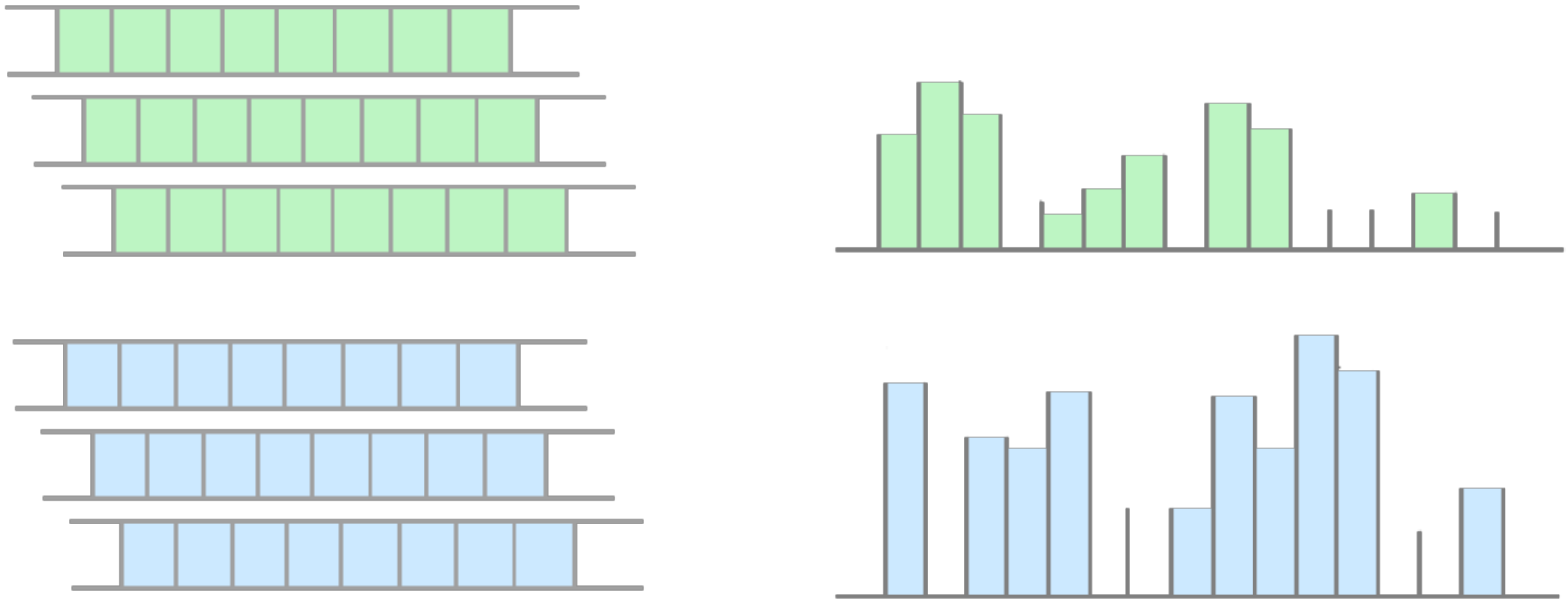




Подсчет k-меров



Подсчет k-меров



$$\mathrm{JSD}(P\|Q) = \frac{1}{2}\mathrm{KL}(P\|M) + \frac{1}{2}\mathrm{KL}(Q\|M)$$

$$\mathrm{KL}(P\|Q) = \sum_i P(i)\ \log \frac{P(i)}{Q(i)}.$$

Проблема хранения k-mer индекса

- Для вычисления JSD k-mer индексы нужно хранить более-менее явно, следовательно...
- Наивный подход слишком дорог по памяти
- Фильтры Блума имеют свои недостатки

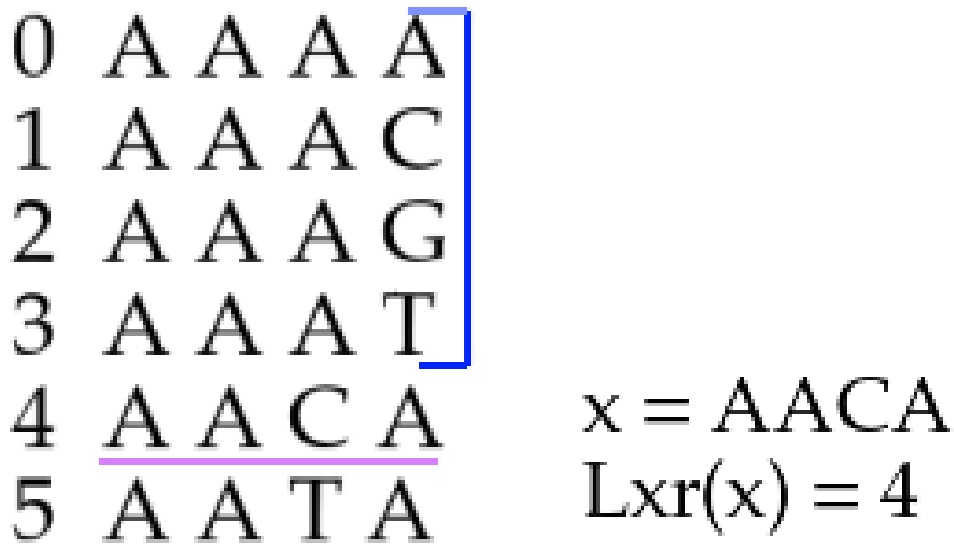
Лексикографический ранг k-мера

- $\mathrm{Lxr}(x)$ - позиция данного k-мера в отсортированном списке **всевозможных** k-меров.
- Альтернативно: число **всевозможных** k-меров, лексикографически меньших данного.

0 A A A A
1 A A A C
2 A A A G
3 A A A T
4 A A C A
5 A A T A

Лексикографический ранг k-мера

- $\mathrm{Lxr}(x)$ - позиция данного k-мера в отсортированном списке **всевозможных** k-меров.
- Альтернативно: число **всевозможных** k-меров, лексикографически меньших данного.



Лексикографический ранг k-мера

Несложные комбинаторные соображения позволяют вычислить лексикографический ранг k-мера "на лету" за $O(k)$.
Рекурсивная и явная формулы для ранга таковы:

- $\mathrm{Lxr}(s, k) = \mathrm{ord}(s_0) \cdot 4^{k-1} \cdot \mathrm{Lxr}(s_{[1:]}, k-1)$
- $\mathrm{Lxr}(s) = \prod_{i=0}^{k-1} \mathrm{ord}(s_i) \cdot 4^{k-i-1}$

Кольцевое хеширование k-меров

Из формул видно, что лексикографические ранги рядом стоящих k-меров могут быть пересчитаны друг через друга:

$$\mathrm{Lxr}(s_{[i:\dots]}) = 4 \cdot \left(\mathrm{Lxr}(s_{[i-1:\dots]}) - o_{i-1} \cdot 4^{k-1}\right) + o_{i+k-1}$$

где предполагается

$$o_i = \mathrm{ord}(s_i)$$

$$s_{[i:\dots]} = s_{[i:i+k-1]}$$

$$s_{[i-1:\dots]} = s_{[i-1:i+k-2]}$$

Данный факт позволяет нам найти лексикографические ранги всех k-меров строки S за $O(|S|)$.

Проблема индексации

Очевидно, что:

$$\mathrm{Lxr}(x) \in [0; 4^k - 1]$$

что означает, что лексикографические ранги могут быть действительно большими числами.

Следствия

- $k \leq 31$ для 64x-машины
- подобранная нами функция позволяет не дает коллизий и позволяет не хранить k-меры в памяти

Для хранения k-мер индексов реализованы быстрые разреженные массивы на C++.

Сравнение с qiime

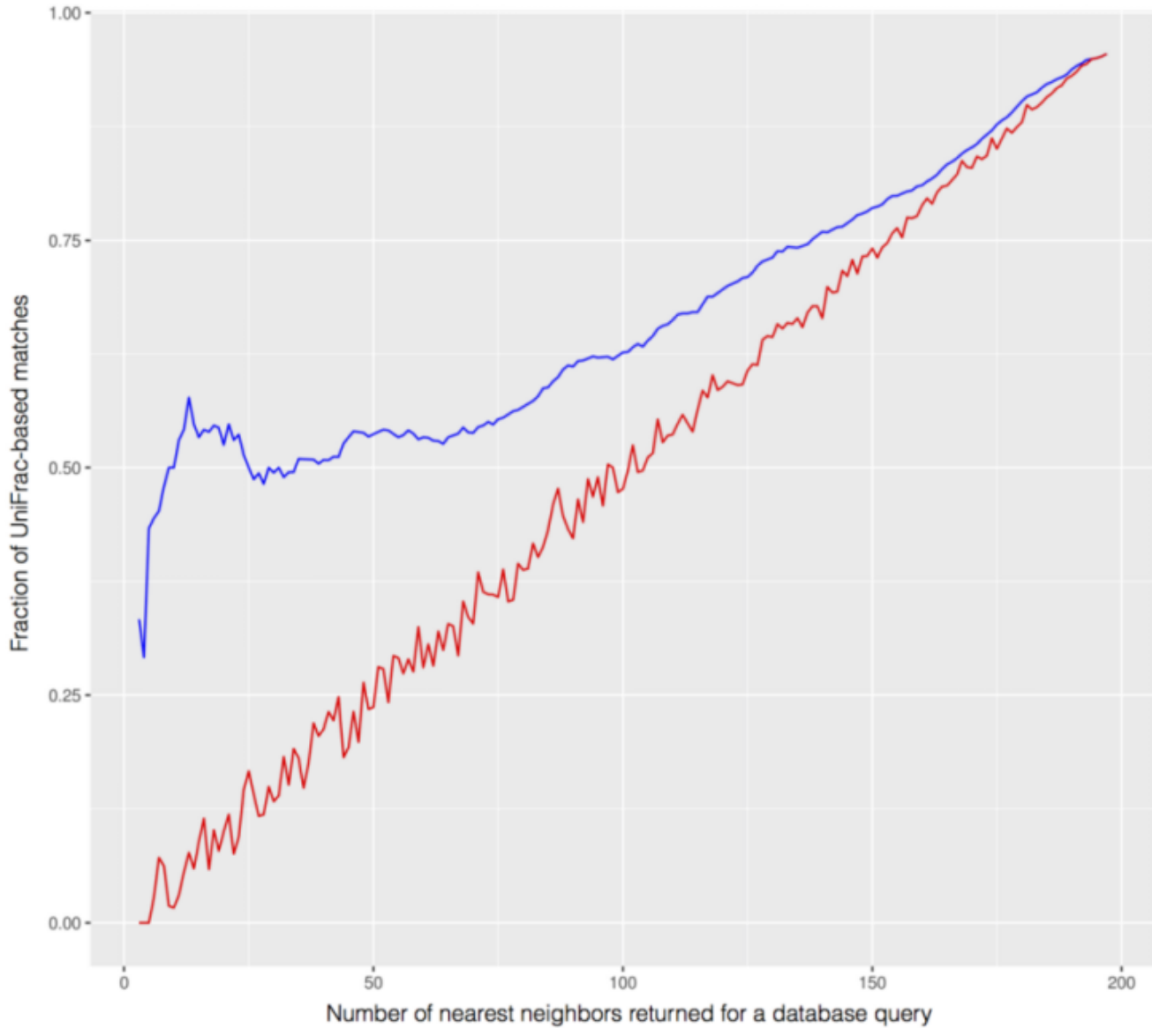
Данные

- 250 библиотек, 1-10k ридов в образцах
- $\sim$ 200Mb в несжатом виде
- 200 образцов для open-reference OTU picking/построения, остальные для добавления

Pipeline	Time	RAM
qiime	180m	$\sim$ 10Gb
amquery	15m	$\sim$ 0.5Gb

Ранги ближайших соседей **коррелируют по Спирману в среднем на 0.91** с ранками соседей по weighted Unifrac.

F-1 score


1.00
0.75
0.50
0.25
0.00
Fraction of UniFrac-based matches
0
50
100
150
200
Number of nearest neighbors returned for a database query

AmQuery

https://github.com/nromashchenko/amquery

Итоги

- Проведен масштабный рефакторинг приложения
- Предложен и реализован более эффективный способ хранения k-меров
- Гиперпараметры алгоритма обучены на базе из 250 образцов
- Проведены сравнения со стандартным пайплайном qiime

Дальнейшие планы

- help, man pages, PyPi
- Исследовать возможность применения LSH-схемы
- Обучение параметров/бенчмарки на других данных
- Публикация результатов