Substructure search optimizations

8 апреля 2023 г.

1 Abstract

TODO

2 Abount Fingerprint-based screening

Что-то примерно, как описано в Sachem поиске

3 Algorithm description

3.1 Notation

многие факты здесь введены, наверное, не очень своевременно

- $\mathbb{M}$ — множество тех молекул, среди которых хотим организвовать поиск
- $substructure(M_1, M_2)$ — предикат для молекул M_1, M_2, который равен $true$ тогда и только тогда, когда M_1 является подстуктурой M_2
- Фингерпринт — битовая строка константной длины. Зафикисруем константу fl равную длине фингерпринта.
- Для каждой молекулы можно построить фингерпринт. $fp : \mathbb{M} \to \mathbb{F}$ — функция, которая строит фингерпринт по молекуле.
- $F[i]$ — i -й бит фингерпринта F
- Будем писать $F_1 \leq F_2$, для фингерпринтов F_1, F_2 если: $\forall i \in \{1, 2, \ldots, fl\}\ F_1[i] \leq F_2[i]$
- $\forall M_1, M_2 \in \mathbb{M}\ substructure(M_1, M_2) \Rightarrow fp(M_1) \leq fp(M_2)$ вроде не относится к Notation, но надо где-то зафиксировать. Вероятно, это будет позже обозначено в секции «About fingerprint-based screening»
- $\mathbb{F} = \{fp(M) \mid M \in \mathbb{M}\}$ — множество фингерпринтов построенных по множеству $\mathbb{M}$

- $\mathbb{F}_M = \{F' \in \mathbb{F} \mid fp(M) \leq F'\}$ — множество всех фингерпринтов, являющихся надмаской фингерпринта $fp(M)$.
- $\mathbb{T}$ — бинарное дерево поиска для фингерпринтов из $\mathbb{F}$.
- $\mathbb{T}.root$ — корень дерева
- d — глубина дерева $\mathbb{T}$. Дерево $\mathbb{T}$ будет полным бинарным деревом глубины d
- $v.left, v.right$ — левое и правое соотвественно поддеревья вершины v бинарного дерева
- $v.set$ — множетсво фингерпринтов, хранящееся в вершине v бинарного дерева. При этом $\bigsqcup\limits_{l \text{ — лист } \mathbb{T}} l.set = \mathbb{F}$ (каждый элемент $\mathbb{F}$ лежит ровно в одном листе)
- $v.leaves$ — множество всех листов поддерева вершины v
- $v.centroid = \bigvee\limits_{l \in v.leaves} \bigvee\limits_{F \in l.set} F$ — центроид записанный в вершине v. По смыслу $v.centroid$ — это фингерпринт F, у которого $\forall i\ F[i] = 1$ тогда и только тогда, когда существует фингерпринт F' в поддереве v, такой что $F'[i] = 1$. centroid — термин из BallTree, который выглядит немного неестественно в данном частном случае. Возможно, стоит ввести переобозначение

3.2 Общая идея

насколько нужен этот раздел? Ниже всё равно опишем то же самое, но более формально

- Не будем работать напрямую с молекулами. Для заданного множества $\mathbb{M}$ построим множество $\mathbb{F}$ и будем решать задачу о поиске $\{F' \in \mathbb{F} \mid F \text{ is submask of } F'\}$ для заданного фингерпринта F
- Тогда для поиска всех надструктур молекулы M сначала найдём множество $\mathbb{F}_M$. Тогда ответом будет $\{M' \in \bigcup\limits_{F' \in \mathbb{F}_M} fp^{-1}(F') \mid M' \text{ is substructure of } M\}$, где $fp^{-1}(F') = \{M' \in \mathbb{M} \mid fp(M') = F'\}$, а проверка «$M'$ is substructure of M» осуществляется с помощью сторонних алгоритмов.
- Для эффективного поиска F_M будем использовать BallTree с метрикой Russel-Rao для множества битовый строк $\mathbb{F}$. В общем-то, это довольно частный случай BallTree, поэтому будем описывать ниже нашу идею, без привязки к обобщённой версии BallTree. слишком неаккуратно написана связь нашего дерева с BallTree

3.3 Поиск в дереве

- Для полученной молекулы M строим $F = fp(M)$. И для F запускаем поиск в дереве.
- Рекурсивно спускаемся в обоих детей, начиная с корня.
- Если оказались в врешине v для которой $F \not\leq v.centroid$, то прекращаем рекурсивный спуск из v.
- Если дошли таким образом до листа l и $F \leq l.centroid$, то добавим в F_M $\{F' \in v.set \mid fp(M) \leq F'\}$ (то есть в листе организуем обычный перебор)

Псевдокод функции поиска по фингерпринту в дереве описан в 1. Псевдокод функции поиска надструктур заданной молекулы описан в 2.

в какое-то место надо засунуть про то, что такой подход можно распараллелить

Algorithm 1 Поиск всех подходящих фингерпринтов в поддереве

Require: v — вершина в дереве. F — фингерпринт
Ensure: $\{F' \in \bigcup_{l \in v.leaves} l.set \mid F \leq F'\}$

```
1:  procedure FindInSubtree(v, F)
2:      if F ⋠ v.centroid then
3:          return ∅
4:      else if v is leaf then
5:          return {F' ∈ v.set | F ≤ F'}
6:      else
7:          left ← FindInSubtree(v.left, F)
8:          right ← FindInSubtree(v.right, F)
9:          return Concatenate(left, right)
10:     end if
11: end procedure
```

Algorithm 2 Поиск всех надструктур заданной молекулы

Require: M — молекула
Ensure: $\{M' \in \mathbb{M} \mid substructure(M, M')\}$

```
1: procedure FindMetaStructures(M)
2:     F ← fp(M)
3:     F_M ← FindInSubtree(𝕋.root, F)
4:     return ⋃_{F' ∈ F_M} fp^{-1}(F')        ▷ fp^{-1}(F) = {M' ∈ 𝕄 | fp(M) = F}
5: end procedure
```

Algorithm 3 Построение дерева

Require: $\mathbb{F}$ — множество фингерпринтов, d — глубина дерева
Ensure: $\mathbb{T}$ — BallTree, для поиска надмасок фингерпринтов

```
1: procedure BuildTree(𝔽, d)
2:     v ← new node
3:     if d = 1 then                ▷ уточнить, остановка на d = 1 или на d = 0
4:         v.set ← 𝔽
5:         v.centroid ← ⋁_{F∈𝔽} F
6:         return v
7:     else
8:         𝔽_l, 𝔽_r ← SplitFingerprints(𝔽)
9:         v.left ← BuildTree(𝔽_l, d − 1)
10:        v.right ← BuildTree(𝔽_r, d − 1)
11:        v.centroid ← v.left.centroid ∨ v.right.centroid
12:        return v
13:    end if
14: end procedure
```

3.4 Построение дерева

Для начала создадим тривиальное дерево из одной вершины $\mathbb{T}.root$. Зададим $\mathbb{T}.root.set = \mathbb{F}$. Далее будем индукционно разделять листы дерева на 2 части и тем самым добавлять новые вершины в дерево. Более формально, будем для каждого листа l дерева разделять $l.set$ с помощью некоторой функции SplitFingerprints: $\mathbb{F}_l, \mathbb{F}_r \leftarrow$ SplitFingerprints$(l.set)$ $(\mathbb{F}_l \sqcup \mathbb{F}_r = l.set)$. А далее рекурсивно строить деревья $l.left, l.right$ для множеств $\mathbb{F}_l, \mathbb{F}_r$.

Будем продолжать такое разделение листов до тех пор, пока $\mathbb{T}$ не станет полным бинарным деревом глубины d. Псевдокод с описанным выше алгоритмом можно найти в 3.

Хочется где-то явно обозначить, что $\mathbb{T}.root \leftarrow$ BuildTree$(\mathbb{F}, d)$

Хотим так выполнять разделения, чтобы в среднем часто происходили отсечения при переборе ветвей. То есть часто выполнялся if в строке 2 алгоритма 1. Обсудим детальнее работу функции SplitFingerprints.

Изначально мы пробовали выбирать некоторый бит i, и отправлять в левое поддерево все фингерпринты F, такие что $F[i] = 0$, а в правое поддерево те, у которых $F[i] = 1$. Тогда при поиске надструктур фингерпринта F', если $F'[i] = 1$, то отсекаем всё левое поддерево. На практике получается, что после небольшого количества разделений при выборе любого бита левая и правая доли сильно различаются. Поэтому не получается сделать достаточно глубокое сбаланстированное дерево. А дерево малой глубины или несбалансированное дерево не даёт серьёзного прироста по сравнению с полным перебором, так как почти не отсекает ветви перебора. стоит ли описать формальнее идею? Или может вообще надо убрать данное описание, так как в итоговом алгоритме используется другой подход

Algorithm 4 Алгоритм разделение фингерпринтов попалам при построении дерева

Require: $\mathbb{F}$ — множество фингерпринтов для разделения
Ensure: $\mathbb{F}_l, \mathbb{F}_r$ — разделённое множество $\mathbb{F}$

procedure SplitFingerprints($\mathbb{F}$)
  $b \leftarrow \arg\min_i \{||\mathbb{F}| - 2k| \mid k = \#\{F \in \mathbb{F} \mid F_i = 1\}\}$ ▷ стоит ли пояснить формулу?
  $\mathbb{F}_l \leftarrow \{F \in \mathbb{F} \mid F[i] = 0\}$
  $\mathbb{F}_r \leftarrow \{F \in \mathbb{F} \mid F[i] = 1\}$
  if $|\mathbb{F}_l| > \lfloor \frac{n}{2} \rfloor$ **then**
    $\mathbb{F}_r \leftarrow \mathbb{F}_r \cup$ TakeLastElements($\mathbb{F}_l, |\mathbb{F}_l| - \lfloor \frac{n}{2} \rfloor$)
    $\mathbb{F}_l \leftarrow$ DropLastElements($\mathbb{F}_l, |\mathbb{F}_l| - \lfloor \frac{n}{2} \rfloor$)
  else if $|\mathbb{F}_r| > \lceil \frac{n}{2} \rceil$ **then**
    $\mathbb{F}_l \leftarrow \mathbb{F}_l \cup$ TakeLastElements($\mathbb{F}_r, |\mathbb{F}_r| - \lceil \frac{n}{2} \rceil$)
    $\mathbb{F}_r \leftarrow$ DropLastElements($\mathbb{F}_r, |\mathbb{F}_r| - \lceil \frac{n}{2} \rceil$)
  end if
  return $\mathbb{F}_l,\ \mathbb{F}_r$
end procedure

Поэтому была выбрана следующая идея: будем выбирать бит так же как описано выше. Но по факту делить так, чтобы размеры совпадали. То есть если лучшее разделение n фингерпринтов даёт доли размеров $n_0, n_1 (n_0 < n_1 \wedge n_0 + n_1 = n)$, то в левую долю отправятся все значений с нулём, а значения с единцей распределятся так, чтобы итоговый размер левой и правой долей были равны $\lfloor \frac{n}{2} \rfloor, \lceil \frac{n}{2} \rceil$ соотвественно. Если $n_0 > n_1$, то будем действтивать симмтрично. Алгоритм функции SplitFingerprints можно найти в псевдокоде 4

4 Benchmarks

время работы на базе pubchem на разных aws машинах. Какой процент молекул отсекаем, сколько в среднем «бесполеных» вершин, в которых придётся идти и влево и вправо, какой процент работает наша часть, а какой процент работает «чёрный ящик»

замерить скорость работы полного перебора в оперативной памяти

5 References

TODO

- Найти что-то про описание ball tree и метрику Rassel-Rao
- Pubchem

- Indigo fingerprint, RDKit fingerprint
- Что-то про AWS