Substructure search optimizations

9 мая 2023 г.

1 Introduction

Задача поиска подграфа в возникает во многих областях, в том числе в биологии. Например, химические соединения представимы в виде графа, а задача поиска химических соединений содержащих данный фрагмент,в больших базах данных является важной подзадачей в процессе разработки лекарств. Возможно, пример стоит выкинуть

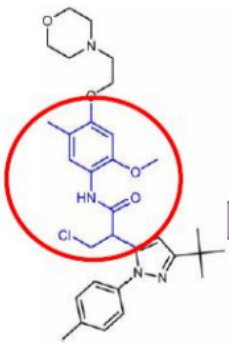

Рис. 1: Пример молекулы и её подструктуры

Например, молекула на рисунке 1 является активной, но не может быть использована по причине:

- фармакологических проблем (ADMET - absorption, distribution, metabolism, excretion and toxicity)
- защищенности патентом

Мы знаем что активность данной молекулы обусловлена выделенным фрагментом.

Значит существует вероятность найти активные молекулы среди ее производных, содержащих данный фрагмент.

Однако задача поиска подграфа в графе является NP-полной, поэтому эффективное решение в общем случае неизвестно. Тем не менее, можно применить эвристические методы ускорения поиска.

A typical pattern to achieve this is the so-called Filter-and-Verification paradigm, which proceeds in two steps. First, a fast filtering step is performed, that significantly reduces the size of the potential solution set by eliminating candidates with a heuristic, then a verification step tries to verify the subgraph relationship for the remaining candidates by a subgraph somorphism test. Абзац сворован из CT-index

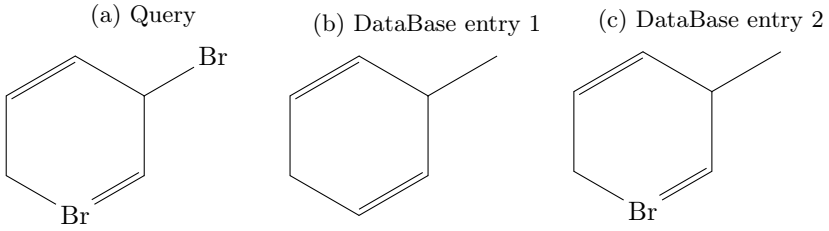

Рис. 2: Пример выделения фичей

Основным методом реализации фильтрации является выделение фич и построение индекса. Каждая фича может быть бинарной или количественной. Предполагается, что если в графе Q_1 есть фича X, а в графе Q_2 её нет, то Q_1 не может быть подструктурой Q_2 . Аналогичное рассуждение можно провести для количественных фич.

кажется, можно либо выкинуть пример, либо сделать его описание проще Рассмотрим примеры фичей на базе данных из двух молекул и одного запроса к данной базе, показанных на рисунке 2

- Примером бинарной фичи может быть «Наличие брома». По данной фиче при выполнении запроса 2a можно отфильтровать молекулу 2b, так как понятно, что в запросе 2a есть бром, а в молекуле 2b его нет, поэтому 2a не может быть подграфом 2b. Однако, как в 2a, так и в 2c есть бром. Поэтому данная фича не отфильтрует 2c и придётся запускать алгоритм верификации, чтобы понять, что 2a не является подструктурой 2c

- Примером количественной фичи может быть «Количество молекул брома в соединении». Данная фича более информативная, чем предыдущая. По ней можно сразу понять, что в запросе 2a 2 молекулы брома, а в элементах базы данных 2b и 2c их 0 и 1 соответственно, а значит запрос 2a не является подструктурой обоих элементов базы данных.

На практике чаще встречаются бинарные фичи, так как количественную фичу N можно разбить на несколько интервалов и задать интервалы бинарными фичами «$N > x_1$», «$N > x_2$», ... и далее работать с бинарными значениями, что обычно является сильно более компактным и лишь немного менее информативным способом представить количественные фичи.

Фингерпринтом молекулы называют битовую строку фиксированной длины, в которой i -й бит равен единице или нулю в зависимости от наличия или отсутствия у молекулы фичи с номером i. Тогда построение индекса представляет из себя построение фингерпринта для каждой молекулы базы данных по заданному набору фич. А фильтрация по такому индексу предполагает проверку для каждого элемента базы данных, что фингерпринт данной молекулы является надмаской запроса.

Современные решения данной задачи концентрируют своё внимание на ускорении поиска с помощью построения качественного фингерпринта, предполагая, что стадия фильтрации будет происходить перебором всех фингерпринтов. Однако количество открытых молекул быстро увеличивается. статистика изменений размера базы pubchem: (27 млн в 2011, CT-index), (94 млн в 2017, sachem), (115 млн в 2023). Поэтому есть предположение, что в будущем крассические подходы к скринингу будут дорожать. Например, уже сейчас при работе с базой данных размером $4 \cdot 10^9$ Дима сказал, что такая база у Quantori есть и обещал найти её. Bingo fingerprint (один из популярных фингерпринтов) для всей базы данных будет весить 1.25 Тб, что потребует для быстрой работы дорогие ресурсы (либо дорогую большую оперативную память, либо дорогие быстрые твердотельные накопители). хорошо бы потестить наше решение на большой базе. А то пока это голословно утверждать, что наше решение успешно будет работать на таких данных

Именно поэтому есть интерес к развитию альтернативных подходов к скринингу. К сожалению, поиск хотя бы одной надструктуры в базе данных для запроса является OV трудной тут надо быть осторожным, так как я сам придумал доказательство OV трудности, поэтому не найдено решение, которое бы в общем случае работало бы сильно быстрее.

В данной статье мы рассматриваем новый подход к скринингу, основанный на бинарном дереве Ball-Tree, который в среднем работает лучше полного перебора, что позволяет ускорить подструктурный поиск.

2 Future development

Представленный подход может быть в дальнейшем модернизирован, так как классические фингерпринты обладают рядом недостатков, которых теоретически можно избежать в реализации с деревом:

- В них не может быть слишком много фич, так как фингерпринты оптимизируются для полного перебора, а значит не могут занимать слишком много памяти.
- Фингерпринты плохо работают со специфичными молекулами, так как не могут разобрать частные случаи и выделить особенные фичи.

Таким образом, предполагается сделать следующий алгоритм:

1. Для данного множества выделить фичу, которая делит его примерно попалам (по наличию/отсутсвию фичи. Можно выделять несколько фичей и решать задачу о выборе комбинации фичей так, чтобы делить множество примерно поровну)
2. Делить множество попалам на 2 поддерева согласно найденной фиче.
3. Рекурсивно продолжать в детях.

Такой подход позволит сэкономить ресурсы, так как в хорошем случае можно будет полностью отказаться от фингерпринтов. А на каждом шаге дерева нужно будет проверить только одну фичу. Также можно будет переиспользовать информацию собранную о запросе и молекулах в листе при спуске к листу (некоторые filter-and-verify подходы так делают, но опять же наш подход предположительно должен быть лучше благодаря его адаптивности к конкретным данным поддереве)

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: $\{F' \in \bigcup_{l \in v.leaves} l.set \mid F \leq F'\}$

procedure FindInSubtree(v, F)
  if $F \not\leq v.centroid$ **then**
    return $\varnothing$
  else if v is leaf **then**
    return $\{F' \in v.set \mid F \leq F'\}$
  else
    $left \leftarrow$ FindInSubtree($v.left, F$)
    $right \leftarrow$ FindInSubtree($v.right, F$)
    return Concatenate($left, right$)
  end if
end procedure

Algorithm 2 Поиск всех надструктур заданной молекулы

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

procedure FindMetaStructures(M)
  $F \leftarrow fp(M)$
  $F_M \leftarrow$ FindInSubtree($\mathbb{T}.root, F$)
  return $\bigcup_{F' \in F_M} fp^{-1}(F')$ ▷ $fp^{-1}(F) = \{M' \in \mathbb{M} \mid fp(M) = F\}$
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($\mathbb{F}, d$)

Algorithm 3 Построение дерева

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

procedure BuildTree($\mathbb{F}, d$)
  $v \leftarrow$ new node
  if $d = 1$ **then** ▷ уточнить, остановка на $d = 1$ или на $d = 0$
    $v.set \leftarrow \mathbb{F}$
    $v.centroid \leftarrow \bigvee_{F \in \mathbb{F}} F$
    return v
  else
    $\mathbb{F}_l, \mathbb{F}_r \leftarrow$ SplitFingerprints($\mathbb{F}$)
    $v.left \leftarrow$ BuildTree($\mathbb{F}_l, d-1$)
    $v.right \leftarrow$ BuildTree($\mathbb{F}_r, d-1$)
    $v.centroid \leftarrow v.left.centroid \vee v.right.centroid$
    return v
  end if
end procedure

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

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

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»

- сравниться с полным перебором (своя версия)
- Сравниться с распределённым полным перебором
- сравниться с pubchem, используя REST-API.

5 References

TODO

- Найти что-то про описание ball tree и метрику Rassel-Rao
- Pubchem
- Indigo fingerprint, RDKit fingerprint
- Что-то про AWS