

# Substructure search optimizations

15 апреля 2023 г.

## 1 Introduction

Достижения в синтетической химии приводят к тому, что молекулы, синтезируемые в настоящее время, состоят из более сложных объектов с механическими связями и более обширных каркасов. Все более важным становится вопрос о том, как пользователи могут эффективно искать такие структуры в больших базах данных.

Задача поиска химических соединений, содержащих заданный фрагмент, в больших базах данных также является одной из задач в процессе разработки лекарств и позволяет решать конкретные проблемы при разработке новых лекарств. Рассмотрим на конкретном примере.

Эта молекула 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.

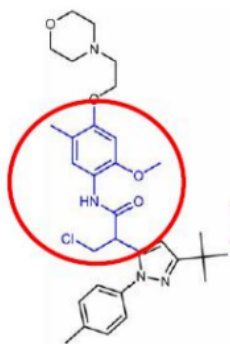
Значит существует вероятность найти активные молекулы среди ее производных, содержащих данный фрагмент.

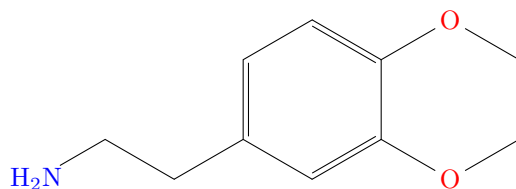
проверить достоверность примера. Точно ли данная молекула обладает данными проблемами? Может составитель задачи подобрал пример с подтекста?

Сделать нормальную картинку в L<sup>A</sup>T<sub>E</sub>X

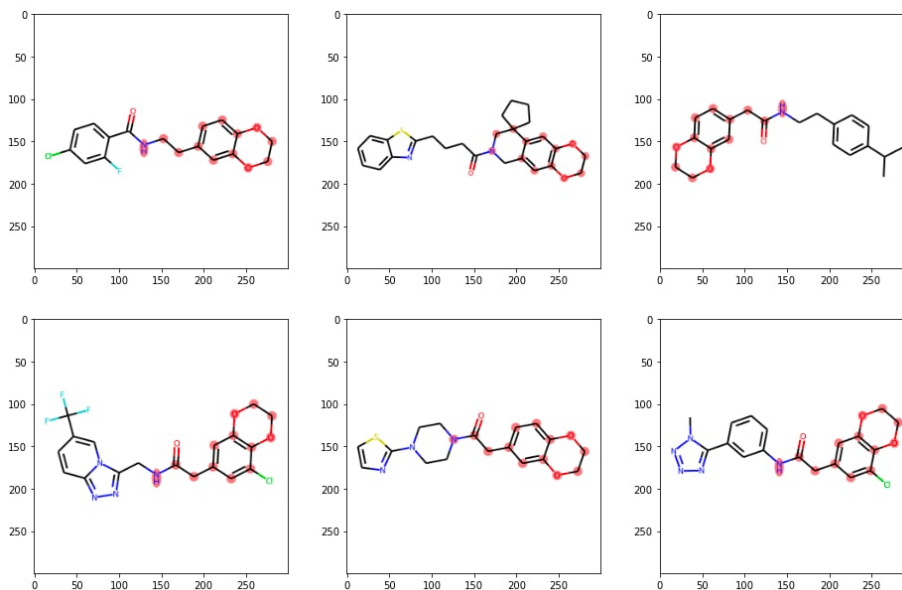
### 1.1 Постановка задачи

Рассмотрим химическое соединение *C*:





Требуется разработать алгоритм, который будет находить все такие химические соединения  $C'$ , в которых  $C$  является подструктурой  $C'$ :



сделать картинку в  $\text{\LaTeX}$

## 2 About 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 организовать поиск
- $substructure(M_1, M_2)$  — предикат для молекул  $M_1, M_2$ , который равен *true* тогда и только тогда, когда  $M_1$  является подструктурой  $M_2$

- Фingerprint — битовая строка константной длины. Зафиксируем константу  $fl$  равную длине fingerprintа.
- Для каждой молекулы можно построить fingerprint.  $fp : \mathbb{M} \rightarrow \mathbb{F}$  — функция, которая строит fingerprint по молекуле.
- $F[i]$  —  $i$ -й бит fingerprintа  $F$
- Будем писать  $F_1 \leq F_2$ , для fingerprintов  $F_1, F_2$  если:  $\forall i \in \{1, 2, \dots, fl\} F_1[i] \leq F_2[i]$
- $\forall M_1, M_2 \in \mathbb{M} \text{ substructure}(M_1, M_2) \Rightarrow fp(M_1) \leq fp(M_2)$  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 fingerprintов построенных по множеству  $\mathbb{M}$
- $\mathbb{F}_M = \{F' \in \mathbb{F} \mid fp(M) \leq F'\}$  — множество всех fingerprintов, являющихся надмаской fingerprintа  $fp(M)$ .
- $\mathbb{T}$  — бинарное дерево поиска для fingerprintов 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 соответственно поддеревья вершины  $v$  бинарного дерева
- $v.set$  — множество fingerprintов, хранящееся в вершине  $v$  бинарного дерева. При этом  $\bigsqcup_{l - \text{лист } \mathbb{T}} l.set = \mathbb{F}$  (каждый элемент  $\mathbb{F}$  лежит ровно в одном листе)
- $v.leaves$  — множество всех листов поддерева вершины  $v$
- $v.centroid = \bigvee_{l \in v.leaves} \bigvee_{F \in l.set} F$  — центроид записанный в вершине  $v$ .  
По смыслу  $v.centroid$  — это fingerprint  $F$ , у которого  $\forall i F[i] = 1$  тогда и только тогда, когда существует fingerprint  $F'$  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  $\{M' \in \bigcup_{F' \in \mathbb{F}_M} fp^{-1}(F') \mid M' \text{ is substructure of } M\}$ , 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 вершине  $v$  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**

### 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:  $\mathbb{F}_l, \mathbb{F}_r \leftarrow \text{SplitFingerprints}(l.set)$  ( $\mathbb{F}_l \sqcup \mathbb{F}_r = l.set$ ). 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  $\mathbb{T}.root \leftarrow \text{BuildTree}(\mathbb{F}, d)$**

---

**Algorithm 1** Поиск всех подходящих fingerprints в поддереве

---

**Require:**  $v$  — вершина в дереве.  $F$  — fingerprint

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

```
1: procedure FINDINSUBTREE( $v, F$ )
2:   if  $F \not\leq v.centroid$  then
3:     return  $\emptyset$ 
4:   else if  $v$  is leaf then
5:     return  $\{F' \in v.set \mid F \leq F'\}$ 
6:   else
7:      $left \leftarrow \text{FINDINSUBTREE}(v.left, F)$ 
8:      $right \leftarrow \text{FINDINSUBTREE}(v.right, F)$ 
9:     return  $\text{CONCATENATE}(left, right)$ 
10:  end if
11: end procedure
```

---

---

**Algorithm 2** Поиск всех надструктур заданной молекулы

---

**Require:**  $M$  — молекула

**Ensure:**  $\{M' \in \mathbb{M} \mid \text{substructure}(M, M')\}$

```
1: procedure FINDMETASTRUCTURES( $M$ )
2:    $F \leftarrow fp(M)$ 
3:    $F_M \leftarrow \text{FINDINSUBTREE}(\mathbb{T}.root, F)$ 
4:   return  $\bigcup_{F' \in F_M} fp^{-1}(F')$   $\triangleright fp^{-1}(F) = \{M' \in \mathbb{M} \mid fp(M) = F\}$ 
5: end procedure
```

---

---

**Algorithm 3** Построение дерева

---

**Require:**  $\mathbb{F}$  — множество fingerprints,  $d$  — глубина дерева

**Ensure:**  $\mathbb{T}$  — BallTree, для поиска надмасок fingerprints

```
1: procedure BUILDTREE( $\mathbb{F}, d$ )
2:    $v \leftarrow \text{new node}$ 
3:   if  $d = 1$  then  $\triangleright$  уточнить, остановка на  $d = 1$  или на  $d = 0$ 
4:      $v.set \leftarrow \mathbb{F}$ 
5:      $v.centroid \leftarrow \bigvee_{F \in \mathbb{F}} F$ 
6:     return  $v$ 
7:   else
8:      $\mathbb{F}_l, \mathbb{F}_r \leftarrow \text{SPLITFINGERPRINTS}(\mathbb{F})$ 
9:      $v.left \leftarrow \text{BUILDTREE}(\mathbb{F}_l, d - 1)$ 
10:     $v.right \leftarrow \text{BUILDTREE}(\mathbb{F}_r, d - 1)$ 
11:     $v.centroid \leftarrow v.left.centroid \vee v.right.centroid$ 
12:    return  $v$ 
13:  end if
14: end procedure
```

---

---

**Algorithm 4** Алгоритм разделение fingerprints попалам при построении дерева

---

**Require:**  $\mathbb{F}$  — множество fingerprints для разделения

**Ensure:**  $\mathbb{F}_l, \mathbb{F}_r$  — разделённое множество  $\mathbb{F}$

```

1: procedure SPLITFINGERPRINTS( $\mathbb{F}$ )
2:    $b \leftarrow \arg \min_i \{ |\mathbb{F}| - 2k \mid k = \#\{F \in \mathbb{F} \mid F_i = 1\} \}$  ▷ стоит ли пояснить формулу?
3:    $\mathbb{F}_l \leftarrow \{F \in \mathbb{F} \mid F[b] = 0\}$ 
4:    $\mathbb{F}_r \leftarrow \{F \in \mathbb{F} \mid F[b] = 1\}$ 
5:   if  $|\mathbb{F}_l| > \lfloor \frac{n}{2} \rfloor$  then
6:      $\mathbb{F}_r \leftarrow \mathbb{F}_r \cup \text{TAKELASTELEMENTS}(\mathbb{F}_l, |\mathbb{F}_l| - \lfloor \frac{n}{2} \rfloor)$ 
7:      $\mathbb{F}_l \leftarrow \text{DROPLASTELEMENTS}(\mathbb{F}_l, |\mathbb{F}_l| - \lfloor \frac{n}{2} \rfloor)$ 
8:   else if  $|\mathbb{F}_r| > \lceil \frac{n}{2} \rceil$  then
9:      $\mathbb{F}_l \leftarrow \mathbb{F}_l \cup \text{TAKELASTELEMENTS}(\mathbb{F}_r, |\mathbb{F}_r| - \lceil \frac{n}{2} \rceil)$ 
10:     $\mathbb{F}_r \leftarrow \text{DROPLASTELEMENTS}(\mathbb{F}_r, |\mathbb{F}_r| - \lceil \frac{n}{2} \rceil)$ 
11:   end if
12:   return  $\mathbb{F}_l, \mathbb{F}_r$ 
13: 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 fingerprints  $F$ , такие что  $F[i] = 0$ , а в правое поддерево те, у которых  $F[i] = 1$ . Тогда при поиске надструктур fingerprintа  $F'$ , 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 сбалансированное 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  $n$  fingerprints 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 единицей распределятся так, чтобы итоговый размер левой и правой долей были равны  $\lfloor \frac{n}{2} \rfloor, \lceil \frac{n}{2} \rceil$  соответственно. Если  $n_0 > n_1$ , то будем действовать симметрично. 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 «беспольных» 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