

Алгоритмы поиска максимальной общей подструктуры набора молекул

© А.С. Савельев

SciTouch LLC

E-mail: savelyev.alex@gmail.com

Реферат.

Диссертация посвящена оптимизации поиска общей подструктуры в химической базе данных. Задача поиска «максимальной общей подструктуры» очевидным образом сводится к задаче поиска «максимального общего подграфа» для помеченных (раскрашенных) графов. Для структур среднего размера предлагается использовать точный алгоритм поиска. В работе предлагаются различные оптимизации для точного алгоритма поиска. Кроме того, в работе представлен точный метод нахождения ядра для множества структур.

Для графов большого размера предлагается использовать оптимизированный приближенный алгоритм, позволяющий искать решения за линейное от размера входных данных время.

Ключевые слова: максимальная общая подструктура, максимальный общий подграф, поиск изоморфизма подграфу

Содержание

1	<i>Введение.</i>	3
2	<i>Определения и терминология.</i>	4
3	<i>Точный метод поиска. Алгоритм Хансера.</i>	8
3.1	Описание алгоритма.	8
3.1.1	Построение ассоциативного графа.	9
3.1.2	Алгоритм Хансера.	10
3.1.3	Псевдокод алгоритма Хансера.	12
3.2	Модификации алгоритма.	13
4	<i>Приближенный метод поиска. Алгоритм 2DOM.</i>	18
4.1	Формулировка и описание алгоритма.	19
4.1.1	Стадия построения: жадный метод.	20
4.1.2	Стадия улучшения: случайный дискретный итерационный метод.	21
4.2	Модификации алгоритма.	23
5	<i>Тестирование алгоритмов поиска максимальной общей подструктуры.</i>	25
6	<i>Задача поиска множественной наибольшей общей подструктуры.</i>	31
6.1	Определение и постановка задачи	31
6.2	Поиск изоморфизма подграфу	31
6.3	Алгоритм определения ядра “The basket”	33
6.4	Тестирование алгоритма нахождения ядра	37
7	<i>Выводы.</i>	40
8	<i>Используемая литература.</i>	41

1 Введение.

В настоящее время практически ни одно исследование не обходится без привлечения вычислительной техники. Не являются исключением и исследования в области разработки лекарств. Нередко возникает потребность в поиске тех или иных химических соединений, обладающих теми или иными свойствами. Процессы, происходящие в живых клетках, слишком запутаны, для того, чтобы быть рассчитанными с помощью какого-либо алгоритма разумной сложности. Тем не менее, информационные технологии со времени своего появления играют немалую роль в облегчении труда химиков и биологов, беря на себя выполнение рутинных операций.

В данной работе речь пойдет о задаче определения общей химической структуры, которая может возникать при поиске по молекулярным базам данных. Поскольку самые современные из этих баз могут насчитывать миллионы записей, необходимо использовать наиболее эффективные методы поиска. Задача поиска максимальной общей подструктуры (Maximum Common Substructure) будет сведена к известной задаче из теории графов, а именно, поиску клики. Далее будут рассмотрены различные подходы к решению этой задачи, предложено усовершенствование одного из них, продемонстрированы преимущества предлагаемых алгоритмов и рассмотрено применение этих алгоритмов для решения более общей задачи – задачи определения ядра у молекул (scaffold detection).

В данной работе задача нахождения MCS определяется и рассматривается в контексте химического и биологического применения. Рассматриваемая задача является важной во многих других приложениях, таких как машинное зрение и распознавание образов [12],[13],[14]. В интересах краткости автор концентрирует свое внимание и исследует поставленную задачу применительно к химическому использованию, но также учитывается возможность использования данной работы для других областей знаний,

например, высшей математики и теории вычислительных машин (computer science).

Задача определения максимального общего подграфа для двух графов - одна из комбинаторно сложных NP-полных проблем. Для задачи не известно полиномиального алгоритма в общем случае [15].

Существует множество статей и алгоритмов, посвященных этой проблеме [8]. Все алгоритмы классифицируются на точные и приближенные. Точные алгоритмы работают в худшем случае за экспоненциальное от размера входных данных время. Среди точных алгоритмов выделяют алгоритмы поиска клики (clique-based algorithms), например, алгоритм Cone [16], Burstall [17] Raymond [18] и других; также алгоритмы поиска с возвратом (backtracking algorithms), например алгоритм McGregor [19] и Wong [20]. Среди приближенных алгоритмов выделяют генетические алгоритмы (genetic algorithms) [21] и алгоритмы комбинаторной оптимизации (combinatorial optimization) [22].

В данной работе в качестве основных алгоритмов были выбраны точный алгоритм Хансера поиска клики и приближенный алгоритм Фунабики и Китамичи комбинаторной оптимизации.

2 Определения и терминология.

Графом $G = (V, E)$ будем называть совокупность двух множеств – множества вершин V и множества ребер $E \subset V \times V$. Через $V(G)$ и $E(G)$ обозначим множество вершин и множество ребер соответственно. Граф называется *неориентированным*, если $\forall e = (a, b) \in E, (b, a) \in E$. Граф называется *графом без петель*, если $\forall u \in V : (u, u) \notin E$. Граф называется *помеченным* с метками на вершинах и ребрах, если на множествах V и E заданы функции $f_V : V \rightarrow N$ и $f_E : E \rightarrow N$

Далее под графом будем понимать помеченный неориентированный граф без петель.

Химические структуры (органические молекулы) представляются в виде помеченных графов; вершины обозначают атомы, а ребра – химические связи между ними. Метки на вершинах и ребрах обозначают соответственно химические элементы и типы химических связей. Молекула, как правило, содержит от десятков до сотен атомов (рисунок 1).

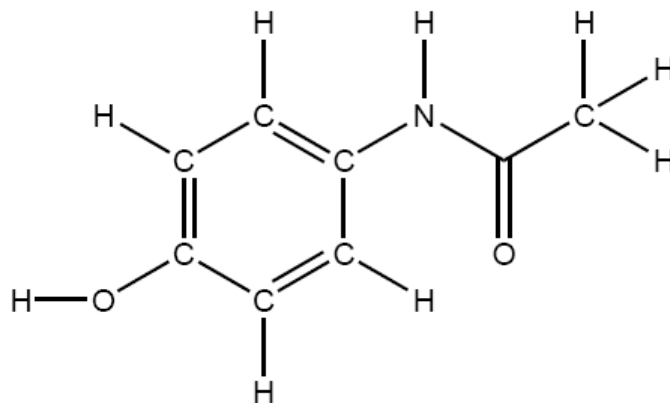


Рисунок 1 Молекула парацетамола. Содержит основные органические элементы (C, N, O, H). Видны «одинарные» и «двойные» связи. В химической нотации атомы водорода (H) обычно не пишутся, а подразумеваются.

Задача распознавания «Изоморфизм подграфу» заключается в том, чтобы дать ответ на вопрос, существует ли подграф графа $G_1(V_1, E_1)$, изоморфный графу $G_2(V_2, E_2)$. Иначе говоря, существуют ли подмножества $V \subseteq V_1$ и $E \subseteq E_1$, такие, что $|V| = |V_2|$ и $|E| = |E_2|$, и такая взаимнооднозначная функция $f_1: V_2 \rightarrow V$, что $\{u, v\} \in E_2$ тогда и только тогда, когда $\{f_1(u), f_1(v)\} \in E$.

Соответствующая задача разрешимости для приведенной задачи изоморфизма подграфу является NP-полной и поэтому не имеет полиномиального решения (в предположении $P \neq NP$) [23]

Максимальным общим подграфом двух графов $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$, где V – набор вершин, E – набор ребер, называется граф G_{12} , такой, что G_{12} является одновременно изоморфным подграфом для G_1 и G_2 , и содержит максимальное количество ребер.¹

¹ Дано определение Maximum Common Edge Subgraph (MCES). В терминологии существует также определение для Maximum Common Induced Subgraph (MCIS), для графа G_{12} с максимальным количеством вершин.

Задача нахождения MCS двух графов также является NP-полной. Один из методов ее точного решения основан на переходе от этой задачи к задаче нахождения *максимального полного подграфа (клики)*.

Граф называется *полным*, если для любых двух его вершин существует, хотя бы одно ребро, соединяющее их.

Клика (максимальный полный подграф) графа $G = (V, E)$ есть порожденный подграф, построенный на подмножестве вершин графа $S \subset V$ и являющийся полным и максимальным в том смысле, что любой другой подграф графа G , построенный на множестве вершин H , содержащим в себе S ($H \supset S$) не является полным. Следовательно, в противоположность *максимальному независимому множеству*, в котором не могут встретиться две смежные вершины, в клике все вершины попарно смежные. Совершенно очевидно, что *максимальное независимое множество* графа G соответствует клике графа \bar{G} и наоборот, где \bar{G} - дополнение графа G .

Линейным графом $L(G)$ графа $G = (V, E)$ называется граф, у которого набор вершин соответствует набору ребер E , то есть если ребро $(v_i, v_j) \in E$, то оно является вершиной $L(G)$. Пара вершин в линейном графе $L(G)$ является смежной, если отвечающие им ребра в графе G инцидентны друг другу.

Ядром для множества графов $\{G_1, G_2, \dots, G_n\}$ называется граф S , который является одновременно изоморфным подграфом для каждого из графов, и содержит максимальное количество ребер.

Реакцией будем называть совокупность двух множеств молекул: молекул-реактантов (вступающих в реакцию) и молекул-продуктов (получающихся в процессе химической реакции).

Задача “Atom-atom mapping” заключается в нахождении такой функции, которая позволяет каждому атому в молекуле-реактанте сопоставить один или несколько атомов в молекулах-продуктах. Пример реакции на рисунке 2

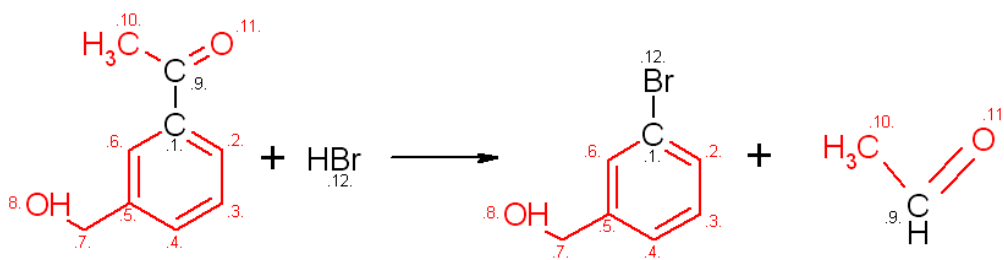


Рисунок 2 Пример реакции с размеченными атомами.

Особенности исходных данных.

Приведенные в данной работе методы могут быть применены к широкому классу графов, но практические результаты были получены на основе базы графов, удовлетворяющих следующим условиям:

- 1) Для графов в используемых базах $|V(G)| < 600$. Доля графов с $|V(G)| > 100$ составляет около 2%.
- 2) Большинство графов планарны.
- 3) Графы связаны и разрежены: $|V(G)| \approx |E(G)|$
- 4) Степень вершин ограничена: $\deg(v) < 12$

3 Точный метод поиска. Алгоритм Хансера.

3.1 Описание алгоритма.

Этот алгоритм был разработан для приложений, использующих поиск химического подобия (chemical similarity) молекул, где молекулярные структуры представляются в виде графов таких, что атомы соответствуют окрашенным вершинам и химические связи соответствуют окрашенным ребрам. Более детальное описание графических представлений для химических структур можно найти в [9].

Нахождение подобия между молекулярными графами играет важную роль во многих аспектах химии и биологии. Например, для предсказания биологической активности, интерпретации молекулярного спектра, при поиске в базе данных. Одним из важных применений является поиск в молекулярных базах данных. Химические базы содержат в себе десятки тысяч различных молекул, что убирает возможность стандартизации, и, соответственно, существенно осложняет задачу. Предложенные в этом обзоре алгоритмы поиска максимальной общей подструктуры были разработаны в том числе и для таких применений.

Алгоритм Хансера основан на переходе от задачи нахождения MCES к задаче нахождения клики. Для этого на первом этапе строится специальный граф, порожденный из двух первоначальных молекул. Этот граф называется ассоциативным графом (association graph). На втором этапе ищется клика построенного ассоциативного графа. По своей сути, алгоритм Хансера является модифицированным *систематическим методом перебора Брона и Кербоша*. Этот метод является существенно упрощенным перебором, использующим дерево поиска. Модификации метода сделаны с целью увеличения скорости поиска, и основаны на использовании основных свойств молекулярных графов.

3.1.1 Построение ассоциативного графа.

Ассоциативный граф (А-граф) строится по следующему правилу. Пусть нам даны два графа $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$, где V – набор вершин, E – набор ребер. Вершины А-графа соответствует парам вершин данных графов, имеющих одинаковую окраску. Ребра А-графа соединяют вершины (u_1, v_1) и (u_2, v_2) в том случае, если выполняется следующее условие:

$$((u_1, u_2) \in E_1 \text{ и } (v_1, v_2) \in E_2) \text{ или } ((u_1, u_2) \notin E_1 \text{ и } (v_1, v_2) \notin E_2). \quad (1)$$

Таким образом, из вышеуказанного определения можно показать, что максимальный полный подграф для А-графа будет соответствовать Maximum Common Induced Subgraph (MCIS) для первоначальных графов. Для перехода от MCIS к проблеме нахождения Maximum Common Edge Subgraph (MCES), А-граф строится для *линейных графов*.

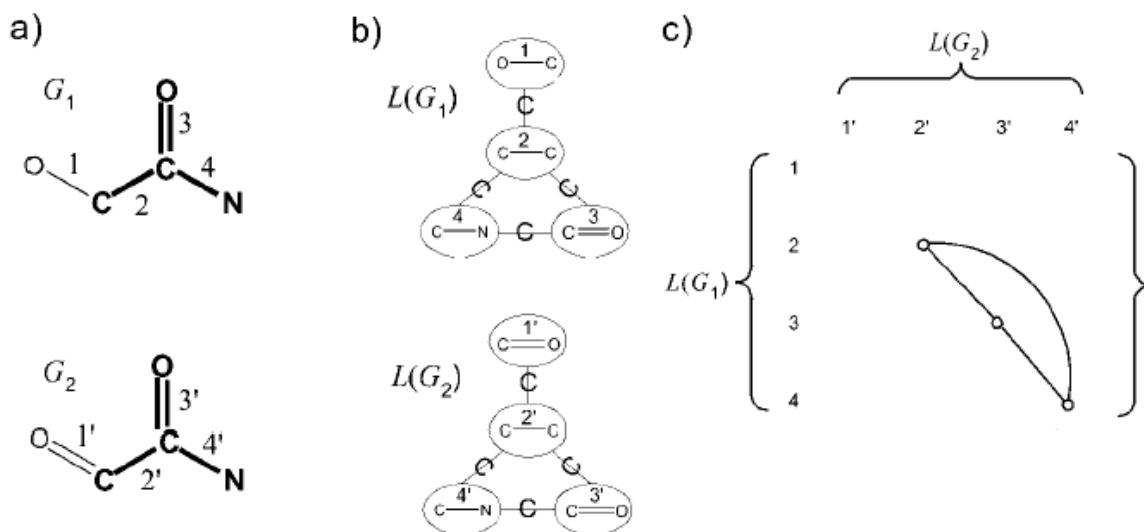


Рисунок 3. а) Исходные молекулярные графы. б) Линейные графы с) Ассоциативный граф для линейных графов.

На рисунке 3 показан пример построения А-графа для двух линейных графов. Понятно, что при вышеописанном методе построения, *полные* подграфы А-графа могут соответствовать несвязным подграфам исходных молекул, а значит им может соответствовать и искомая клика. Поэтому, метод Хансера использует отличающуюся процедуру построения ассоциативного графа, а именно, строятся два вида ребер. Так называемые «запрещенные» ребра

соединяют вершины (u_1, v_1) и (u_2, v_2) A-графа в том случае, если НЕ выполняется условие (1). Таким образом, для каждой вершины x A-графа существует множество вершин $Fbn(x)$ (Forbidden). Вершины $Fbn(x)$ не могут принадлежать независимому множеству (см. пункт 3.1.2), в котором содержится вершина x , так как для них не выполняется условие (1). Одновременно с запрещенным множеством, для каждой вершины x A-графа строится множество $Ext(x)$ (Extension) вершин, которые будут использоваться для расширения независимого множества. Вершины (u_2, v_2) принадлежат множеству $Ext(x)$, где $x = (u_1, v_1)$, если выполняется условие:

$$((u_1, u_2) \in E_1 \text{ и } (v_1, v_2) \in E_2) \quad (2)$$

3.1.2 Алгоритм Хансера.

По своей сути, алгоритм Хансера является модифицированным *систематическим методом перебора Брона и Кербоша*. Этот метод осуществляет поиск максимальных независимых множеств. При этом применяется поиск с возвратами. На каждом этапе к текущему множеству добавляется вершина, и проверяется условие на сохранение независимости множества. Если условие не выполнилось, то метод возвращается в исходную ситуацию. Максимальность независимого множества в методе Брона и Кербоша проверяется условием невозможности дальнейшего расширения, а также условием «пустоты» множества вершин, которые уже использовались [1].

Алгоритм Хансера осуществляет перебор независимых множеств A-графа, а максимальность множества определяется лишь условием невозможности дальнейшего расширения. В процессе поиска – на некотором этапе k – независимое множество вершин S_k расширяется путем добавления к нему подходящим образом выбранной вершины (чтобы получилось новое независимое множество S_{k+1}) на этапе $k+1$, и так делается до тех пор, пока

добавление вершин не станет невозможным, а порождаемое множество не станет максимальным независимым множеством.

В некоторый произвольный момент работы алгоритм хранит два типа подмножеств множества вершин A -графа: подмножество Q_k^- тех вершин, которые нельзя использовать в процессе поиска для расширения множества S_k , и подмножество Q_k^+ таких вершин, которые будут использоваться для расширения множества S_k . Тогда дальнейшее ветвление в дереве поиска включает процедуру выбора вершин $x_{i_k} \in Q_k^+$, добавление ее к S_k для построения множества

$$S_{k+1} = S_k \cup \{x_{i_k}\}$$

и порождение новых множеств:

$$Q_{k+1}^- = Q_k^- \cup Fbn(x_{i_k}) \quad \text{и} \quad Q_{k+1}^+ = (Q_k^+ \cup Ext(x_{i_k})) - Q_{k+1}^-$$

Как видно из определений, множество Q_k^- расширяется за счет «запрещенных» вершин, что определяет независимость для множества S_k в последующих шагах ветвления. А множество Q_k^+ расширяется путем добавления вершин, которые определяют «связность» множества S_k , то есть в качестве решений рассматриваются только связные подграфы.

Необходимым и достаточным условием в методе Хансера того, что S_k – максимальное независимое множество, является выполнение равенства:

$$Q_k^+ = \emptyset$$

В методе Брона и Кербоша достаточным условием для осуществления шага возвращения является существование вершины $x \in Q_k^-$, для которой $\overline{Fbn(x)} \cap Q_k^+ = \emptyset$ (см. [1]). Алгоритм Хансера, как уже упоминалось ранее,

является несколько модифицированным методом. Одна из его модификаций связана с изменением условия для осуществления шага возвращения. Очевидно, что на очередном этапе работы алгоритма можно построить множество вершин P_k , которое, потенциально, может стать максимальным независимым множеством в последующих шагах ветвления:

$$P_k = (X - Q_k^-) \cup S_k$$

Условие возвращения заключается в проверке, является ли текущее «потенциальное» независимое множество P_k подмножеством к какому-либо из уже найденных решений. Таким образом, условие

$$\exists S \in \mathfrak{S} : \text{proj}^{G_1}(P_k) \subset \text{proj}^{G_1}(S) \text{ или } \text{proj}^{G_2}(P_k) \subset \text{proj}^{G_2}(S), \quad (3)$$

где proj^G - оператор проекции вершин A-графа, устанавливающий соответствие между вершинами A-графа и ребрами исходных графов G_1 и G_2 , \mathfrak{S} - множество найденных решений на текущем этапе работы алгоритма, является достаточным для осуществления шага возвращения, поскольку из S_k при всяком дальнейшем ветвлении уже не получится максимальное независимое множество.

3.1.3 Псевдокод алгоритма Хансера.

function $MCS(G_1, G_2)$

$X \leftarrow \text{constructAssociationGraph}(G_1, G_2)$

return $\text{searchClique}(X)$

end function

function $\text{searchClique}(X)$

$S_0 \leftarrow \emptyset$

$Q_0^+ \leftarrow X$

$Q_0^- \leftarrow \emptyset$

return $\text{parseRecursive}(S_0, Q_0^+, Q_0^-)$

end function

```

function parseRecursive( $S_k, Q_k^+, Q_k^-$ )
  if  $Q_k^+ = \emptyset$ 
     $\mathfrak{S} \leftarrow \mathfrak{S} \cup S_k$ 
  else
     $P_k \leftarrow (X - Q_k^-) \cup S_k$ 
    if mustContinue( $P_k$ )
      for  $x_{i_k} \in Q_k^+$  do
         $Q_{k+1}^- \leftarrow Q_k^- \cup Fbn(x_{i_k})$ 
        if  $S_k = \emptyset$ 
           $Q_{k+1}^+ \leftarrow (Ext(x_{i_k})) - Q_{k+1}^-$ 
        else
           $Q_{k+1}^+ \leftarrow (Q_k^+ \cup Ext(x_{i_k})) - Q_{k+1}^-$ 
        end if
         $S_{k+1} \leftarrow S_k \cup x_{i_k}$ 
         $Q_k^- \leftarrow Q_k^- \cup x_{i_k}$ 
        parseRecursive( $S_{k+1}, Q_{k+1}^+, Q_{k+1}^-$ )
      end for
    end if
  end if
end function

```

```

function mustContinue( $P$ )
  for  $S_k \in \mathfrak{S}$ 
    if ( $proj^{G1}(P) \subset proj^{G1}(S_k)$ ) or ( $proj^{G2}(P) \subset proj^{G2}(S_k)$ )
      return false
    end if
  end for
  retrun true
end function

```

3.2 Модификации алгоритма.

Как и во всяком методе, использующем дерево поиска, в методе Хансера выгодно стремиться начать шаги возвращения как можно раньше, поскольку это ограничит «размеры» «ненужной» части дерева поиска. Для перехода на

шаг возвращения алгоритм Хансера делает проверку условия (3). Это условие оказывается более эффективным, чем условие, проверяемое в методе Брона и Кербоша (см. [1]).

В алгоритме Хансера автором были сделаны модификации, позволяющие существенно ускорить процесс поиска. Во-первых, была изменена медленная функция *mustContinue()*, проверяющая условие возврата. Основная часть времени при вызове этой функции тратилась на вычисление проекций для вершин потенциального независимого множества и уже найденных решений. Алгоритм был переделан таким образом, что необходимые для условия возврата проекции вычисляются сразу, в основном цикле. Проекции уже найденных решений сохраняются вместе с решениями (все решения имеют структуру дерева для быстрого доступа). Во-вторых, рекурсивный алгоритм был переделан в итеративный за счет самостоятельной организации стека вычислений (максимальная глубина стека известна – минимальный размер входных графов). Псевдокод модифицированного алгоритма приведен ниже:

```
function MCSi( $G_1, G_2$ )  
   $X \leftarrow \text{constructAssociationGraph}(G_1, G_2)$   
  return searchCliqueIterative( $X$ )  
end function
```

function *searchCliqueIterative*(X, n)

$$\left. \begin{array}{l} S_j \leftarrow \emptyset \\ Q_j^+ \leftarrow \emptyset \\ Q_j^- \leftarrow \emptyset \\ Q_j^{G1} \leftarrow \emptyset \\ Q_j^{G2} \leftarrow \emptyset \\ S_j^{G1} \leftarrow \emptyset \\ S_j^{G2} \leftarrow \emptyset \\ x_j \leftarrow -1 \end{array} \right\} j = 1..n$$

$Q_0^+ \leftarrow X$

$Q_0^{G1} \leftarrow \text{proj}^{G1}(X)$

$Q_0^{G2} \leftarrow \text{proj}^{G2}(X)$

parseIterative($S, Q^+, Q^-, Q^{G1}, Q^{G2}, S^{G1}, S^{G2}$)

end function

function *parseIterative*($S, Q^+, Q^-, Q^{G1}, Q^{G2}, S^{G1}, S^{G2}$)

$level \leftarrow 0$

while *true* **do**

for $x_{level}^i \in Q_{level}^+$ **do**

$$Q_{level+1}^- \leftarrow Q_{level}^- \cup Fbn(x_{level}^i)$$

$$Q_{level+1}^{G1} \leftarrow Q_{level}^{G1} \cap Ext^{G1}(x_{level}^i)$$

$$Q_{level+1}^{G2} \leftarrow Q_{level}^{G2} \cap Ext^{G2}(x_{level}^i)$$

$$P_k \leftarrow (X - Q_k^-) \cup S_k$$

if $S_{level} = \emptyset$

$$Q_{level+1}^+ \leftarrow (Ext(x_{level}^i)) - Q_{level+1}^-$$

else

$$Q_{level+1}^+ \leftarrow (Q_{level}^+ \cup Ext(x_{level}^i)) - Q_{level+1}^-$$

end if

$$S_{level+1} \leftarrow S_{level} \cup x_{level}^i$$

$$S_{level+1}^{G1} \leftarrow S_{level}^{G1} \cup proj^{G1}(x_{level}^i)$$

$$S_{level+1}^{G2} \leftarrow S_{level}^{G2} \cup proj^{G2}(x_{level}^i)$$

$$Q_{level}^- \leftarrow Q_{level}^- \cup x_{level}^i$$

$level \leftarrow level + 1$

if $Q_{level}^+ = \emptyset$

$$\mathfrak{S} \leftarrow \mathfrak{S} \cup \{S_{level}, S_{level}^{G1}, S_{level}^{G2}\}$$

$$x_{level} \leftarrow -1$$

$level \leftarrow level - 1$

if $level < 0$ **break**

else

if $!continue(S_{level}^{G1}, S_{level}^{G2}, Q_{level}^{G1}, Q_{level}^{G2})$

$$x_{level} \leftarrow -1$$

$level \leftarrow level - 1$

if $level < 0$ **break**

end if

end if

end for

$level \leftarrow level - 1$

if $level < 0$ **break**

end

end function

```

function continue( $S_{level}^{G1}, S_{level}^{G2}, Q_{level}^{G1}, Q_{level}^{G2}$ )
   $P^{G1} \leftarrow S_{level}^{G1} \cup Q_{level}^{G1}$ 
   $P^{G2} \leftarrow S_{level}^{G2} \cup Q_{level}^{G2}$ 
  for  $S_k^{G1}, S_k^{G2} \in \mathfrak{S}$ 
    if ( $P^{G1} \subset S_k^{G1}$ ) or ( $P^{G2} \subset S_k^{G2}$ )
      return false
    end if
  end for
  retrun true
end function

```

Еще одна модификация алгоритма связана с использованием основного свойства молекулярных графов – их сильной разреженности. При реализации алгоритма используются битовые поля, позволяющие перейти от операций над множествами к быстрым бинарным операциям. Битовые поля моделируют массивы битов (логических величин) произвольного размера. Их удобно использовать для различных представлений графов. Например, каждая строка матрицы смежности это массив булевских значений, который может быть задан с помощью битового поля. Основное свойство молекулярных графов – сильная разреженность, то есть в каждой строке матрицы очень мало значений «true» и много «false». Для нахождения следующего значения «true», а также для выполнения бинарных операций между двумя массивами («и», «или», «или не» и т.д.) в битовом поле затрачивается порядка $O(1)$ действий; в то время как при поиске в обычном булевском массиве следующего значения «true», или сравнения попарно элементов двух массивов требуется порядка $O(N)$ действий.

4 Приближенный метод поиска. Алгоритм 2DOM.

Задача нахождения максимальной общей подструктуры является NP-полной. Это означает, что не существует точного алгоритма, который бы гарантировал нахождение решения за полиномиальное время. Скорее всего, время поиска будет возрастать экспоненциально с ростом характерного размера. Таким образом, задача приближенного алгоритма – найти приближенное решение, но за полиномиальное время. Обычно на практике, для молекул большого размера, не требуется точного решения, а затраты времени, при использовании точных алгоритмов, получаются колоссальными.

В этой части будет описан новый приближенный алгоритм, имеющий название 2DOM (2-stage discrete optimization method). 2DOM, как ясно из названия, состоит из 2 стадий: построения и улучшения (корректировки). На стадии построения создается возможное начальное решение, удовлетворяющее заданным условиям. Алгоритм основан на эвристической стратегии поиска, а именно на «жадном» методе, создающим оптимальное решение шаг за шагом, постоянно делая выбор лучшего на данный момент варианта, исходя из функции стоимости. На стадии улучшения постепенно совершенствуется начальное решение, построенное на первом этапе. Алгоритм основан на дискретном методе, понижающем ошибку. Этот итерационный метод делает переход текущего состояния в случайно выбранное близкое состояние и оставляет последнее в случае понижения значения функции ошибки. Такие переходы делаются до тех пор, пока не будет достигнуто точное решение, или итерации не достигнут заданного предела.

4.1 Формулировка и описание алгоритма.

Пусть даны исходно два неориентированных графа $G = (V_1, E_1)$ и $H = (V_2, E_2)$.

Решением задачи в контексте рассматриваемого алгоритма является нахождение подграфа $G' = (V_1', E_2')$ для G и подграфа $H' = (V_2', E_2')$ для H , таких, что: G' и H' изоморфны и количество ребер в них максимально.

Два исходных графа считаются неориентированными и непомеченными, но 2DOM легко расширяется и на графы с окрашенными вершинами и взвешенными ребрами, то есть на молекулы. Конфигурация ребер в каждом графе представляется в виде матрицы смежности. Пусть N и M количество вершин в G и H соответственно ($|V_1| = N, |V_2| = M$). Элемент матрицы C для графа G $c(i, j) = 1$ ($i = 1..N, j = 1..N$), если вершины $\#i$ и $\#j$ смежные, и равен «0» в противоположном случае. Аналогично для элемента матрицы D графа H $d(i, j)$ ($i = 1..M, j = 1..M$).

Без потери общности считаем, что количество вершин N всегда меньше либо равно количеству вершин M . Итоговой целью алгоритма является сопоставить каждую из вершин в графе G определенной вершине графа H . Пусть x_i - вершина в графе H , которая была сопоставлена вершине $\#i$ в графе G . Таким образом, $1 \leq x_i \leq M$, если $i \neq j$, то $x_i \neq x_j$ для $i = 1..N$ и $j = 1..N$.

Функция ошибки представляет собой число *несовпадающих* ребер в G , которое необходимо минимизировать. Ребро $\{i, j\} \in E_1$ называется *несовпадающим*, если $\{x_i, x_j\} \notin E_2$, *совпадающим* – в противоположном случае. Количество *несовпадающих* ребер E дается формулой:

$$E = \frac{1}{2} \sum_{i=2}^N \left(\sum_{\substack{j=1 \\ j \neq i}}^N c(i, j)(1 - d(x_i, x_j)) \right)$$

4.1.1 Стадия построения: жадный метод.

Жадный эвристический метод используется на стадии построения с целью создания начального вероятного решения с «приемлемым качеством» за короткое время. Метод последовательно сопоставляет каждую вершину в G индивидуальной вершине в H , одновременно минимизируя функцию ошибки E . Вначале выбирается несопоставленная вершина в графе G , имеющая максимальное количество инцидентных ребер с сопоставленными вершинами. Далее, выбирается несопоставленная вершина в графе H , такая, что при сопоставлении ее с первой, количество совпадающих ребер будет максимальным. Если при выборе вершин две или более имеют одинаковые максимальные числа, преимущество отдается той, у которой степень вершины выше.

Нижеописанная процедура изображает жадный метод в 2DOM. L_G, L_H – списки несопоставленных вершин в G и H соответственно, сортированные в убывающем порядке степеней вершин (степень вершин дается формулой: $\sum_{\substack{j=1 \\ j \neq i}}^N c(i, j)$). x_i это вершина в H , сопоставленная вершине $\#i$ в G .

y_i вершина в G , сопоставленная вершине $\#i$ в H . s_i – статус смежности: $s_i = 1$ если вершина $\#i$ в H является смежной для сопоставленной вершины, 0 если нет.

- 1) Создать L_G и L_H , где, если две или более вершины имеют одинаковую степень, порядок среди них случаен.
- 2) Инициализировать x_i, y_j, s_j нулями для $i = 1..N, j = 1..M$
- 3) Выбрать первые элементы в L_G и L_H для начального сопоставления. Пусть вершина $\#p$ и вершина $\#q$ были выбраны здесь для G и H соответственно.
- 4) Сопоставить вершину $\#p$ в G вершине $\#q$ в H , положив $x_p = q$ и $y_q = p$. Удалить их из L_G и L_H .

- 5) Завершить процедуру в случае, если список L_G пуст.
- 6) Обновить статус смежности вершин в L_H , которые смежные для вершины $\#q$: если $y_i = 0$ и $d(i, q) = 1$, то $s_i = 1$ для $i = 1..M$
- 7) Выбрать первую вершину из L_G , которая имеет максимальное количество инцидентных ребер с сопоставленными вершинами в G . Выбрать первую вершину в L_H , такую, что при сопоставлении ее с выбранной в L_G вершиной, количество совпадающих ребер будет максимальным. Здесь, вершина с $s_i = 1$ имеет приоритет в L_H . Аналогично шагу 3, пусть вершина $\#p$ и вершина $\#q$ были выбраны для G и H соответственно. Перейти на шаг 4.

4.1.2 Стадия улучшения: случайный дискретный итерационный метод.

Случайный дискретный итерационный метод используется на второй стадии алгоритма 2DOM, стадии улучшения. Этот метод итерационно уменьшает функцию ошибки. Метод производит переход текущего состояния в случайно выбранное близкое состояние и оставляет последнее в случае понижения значения функции ошибки. Переход состояний реализуется с помощью комбинации операций замены и сдвига в сопоставлениях вершин. Если число вершин N в графе G равно числу вершин M в графе H , то производятся только операции замены сопоставлений вершин H между двумя случайно выбранными вершинами в графе G . С другой стороны, когда $N < M$, производятся также операции сдвига. Для того чтобы достичь решения задачи как можно быстрее, используется список L_E вершин G инцидентных несовпадающим ребрам: если $E_i > 0$, то вершина $\#i$

включается в L_E . $E_i = \sum_{\substack{j=1 \\ j \neq i}}^N c(i, j)(1 - d(x_i, x_j))$ - функция ошибки для вершины

$\#i$. Соответственно

$$E = \frac{1}{2} \sum_{i=1}^N E_i$$

Переход в другое состояние происходит путем сдвига или замены одной из случайно выбранных вершин в L_E . Нижеописанная процедура изображает случайный итерационный метод. Пусть t – количество итерационных шагов, T – максимальное значение t .

- 1) Положить $t = 0$, $T = 300 \cdot N$. Инициализировать решение, применив вышеописанный жадный метод: задать x_i, y_j , $i = 1..N, j = 1..M$, создать L_E и вычислить функцию ошибки E .
- 2) Случайно выбрать одну вершину из L_E . Пусть $\#p$ была выбрана.
- 3) Случайно выбрать одну вершину из H , отличную от x_p для смены сопоставления. Пусть вершина $\#q$ была выбрана.
- 4) Проверить сопоставлена ли вершина $\#q$ с какой-либо вершиной из G или нет: если $y_q > 0$, то перейти к шагу 5 для операции замены, иначе, перейти к шагу 6 для сдвига.
- 5) Заменить сопоставления между вершиной $\#p$ и вершиной $\#y_q$ в G и вычислить новое значение функции ошибки E_i' для $i = 1..N$ после операции замены:

$$x_{y_q} = x_p, x_p = q, y_{x_{u_q}} = y_q, y_q = p \text{ и } E_i' = \sum_{\substack{j=1 \\ j \neq i}}^N c(i, j)(1 - d(x_i, x_j))$$

перейти к шагу 7

- 6) Сдвинуть сопоставление вершины $\#p$ в G к вершине $\#q$, вычислить новую функцию ошибки E_i' для $i = 1..N$: $y_{x_p} = 0, x_p = q, \text{ и } y_q = p$
- 7) Вычислить новую функцию суммарной ошибки: $E' = \frac{1}{2} \sum_{i=1}^N E_i'$
- 8) Если $E' < E$, то оставить данное состояние, обновив E и L_E .

Завершить процедуру в случае если $E = 0$ или $t = T$. Иначе увеличить значение t на 1 и перейти к шагу 2.

4.2 Модификации алгоритма.

По своей сути 2DOM является алгоритмом поиска минимума функции многих переменных. Вышеописанный метод не исключает возможности попадания в локальные минимумы функции ошибки E . Поэтому, одна из главных модификаций алгоритма разработана для решения проблемы попадания в локальные минимумы. Во-первых, для избегания ситуации застревания на плато (в контексте данного алгоритма, совокупность близкорасположенных состояний, для которых значения функции E равны) на шаге 8, в итерационном методе, условие для принятия текущего состояния записывается как: если $E' \leq E$, то оставить данное состояние.

Во-вторых, вводится дополнительный счетчик, который подсчитывает время пребывания в одном состоянии. Если значение счетчика превысило заданный предел, есть подозрение на то, что алгоритм застрял в локальном минимуме. Текущее состояние сохраняется, и системе сообщается дополнительная «энергия» для преодоления минимума, то есть происходит переход в состояние с более высоким значением функции ошибки. Счетчик обнуляется, и алгоритм продолжает работу. После завершения процедуры поиска, найденное решение сравнивается с сохраненным. Алгоритм оставляет решение с меньшим значением функции ошибки.

Еще одна модификация связана с уменьшением вычислительного времени. На шаге 5 и 6 подсчитывается новые функции ошибки E_i' не для всех $i = 1..N$, а лишь для выбранных вершин, то есть для $i = p$ и для $i = y_q$ в случае замены, и для $i = p$ в случае сдвига. Проверка на шаге 7 и 8 также осуществляется для выбранных вершин. При этом, алгоритм осуществляет порядка $O(2 \cdot N)$ действий, вместо $O(N \cdot N)$, которые он осуществлял ранее. Кроме того, используется основное свойство молекулярных графов

– сильная разреженность. При реализации алгоритма используются битовые поля, позволяющие перейти от операций над множествами к быстрым бинарным операциям.

5 Тестирование алгоритмов поиска максимальной общей подструктуры.

Модифицированный алгоритм Хансера сравнивался при тестировании с 4-мя алгоритмами: с методом Брона и Кербоша, а также с модифицированным методом Брона и Кербоша (с использованием битовых полей), алгоритмом Хансера (без модификаций), а также с алгоритмом Мак Грэгора [19]. Задача нахождения максимальной клики, а также проблемы MCES/MCIS не имеют стандартного набора графов, на которых данные алгоритмы будут работать наиболее эффективно. Одним из важных практических применений алгоритма поиска наибольшей общей подструктуры является нахождение Atom-Atom Mapping в химических реакциях [10]. Поэтому, тестирование алгоритма Хансера и его экспериментальная оценка проводилась на молекулярных графах, взятых из реакционной базы данных. Как известно, в реакциях присутствуют два вида молекул – реагенты (молекулы, вступающие в реакцию) и продукты (молекулы, образующиеся в результате) реакции. При поиске Atom-Atom Mapping возникает необходимость искать наибольшую общую подструктуру для молекул-реагентов и молекул-продуктов реакции. Были проведены тесты для более 5000 реакций. С целью проверки надежности и эффективности предложенных алгоритмов для каждой реакции сравнивались все возможные пары молекул. В итоге было выполнено более 10000 MCES сопоставлений для каждого из методов. Среднее количество вершин в молекулярных графах 23.86 со стандартной девиацией 6.32. Для сохранения единообразия все алгоритмы были скомпилированы C++ MS Visual Studio 9.0. Тесты проводились на одинаковых структурах и были запущены на процессоре Mobile Intel Core 2 Duo 1666 MHz с 2 GB RAM в системе Windows XP. В таблице 1 приведены результаты некоторых тестов.

Таблица 1 Результаты тестирования 5 точных алгоритмов поиска MCS

Количество вершин				Время тестирования алгоритма (мс)				
Молекула 1	Молекула 2	MCS	A-граф	McGregor	BronKerbosh	BK_bitset	Hanser	Modified Hanser
11	15	11	55	136	375	16	2	7
14	17	13	57	1714	1328	47	6	8
14	15	12	70	5936	4062	172	11	11
14	15	14	82	7941	8547	406	15	14
15	15	14	96	16060	17703	578	20	18
16	18	16	106	110588	107336	1609	22	23
14	17	12	126	255454	292963	2844	40	33
11	27	11	154	499650	559781	5813	67	46
27	27	18	286	0.9h	0.5h	178794	297	164
31	31	30	504	7h	9h	0.8h	1328	506
42	53	24	784	>24h	>24h	7h	3516	981

В первых 4-х колонках таблицы содержится информация о размерах тестируемых молекул, а именно: количество атомов (вершин молекулярных графов) в 2-х молекулах, количество атомов в максимальной общей подструктуре, а также количество вершин в ассоциативном графе. В последних 5 колонках таблицы содержится информации о времени поиска решения одним из 5 тестируемых алгоритмов соответственно. Принцип работы почти всех алгоритмов основан на поиске клики у ассоциативного графа (всех, кроме метода поиска Мак Грэгора). Размер A-графа является неплохим критерием определения сложности задачи, поэтому размер A-графа используется везде ниже для удобства сравнения алгоритмов. Из таблицы видно, что время нахождения решения для каждого алгоритма растет с увеличением количества вершин у A-графа. Для методов Брона и Кербоша и Мак Грэгора, в отличие от метода Хансера, вид зависимости времени от количества вершин в графе имеет более выраженный растущий характер. На рисунке 4 представлены экспериментальные зависимости для тестируемых алгоритмов, а также кривые аппроксимации. Каждая

экспериментальная точка на графике представляет собой вычисленное среднее значение за большое количество измерений.

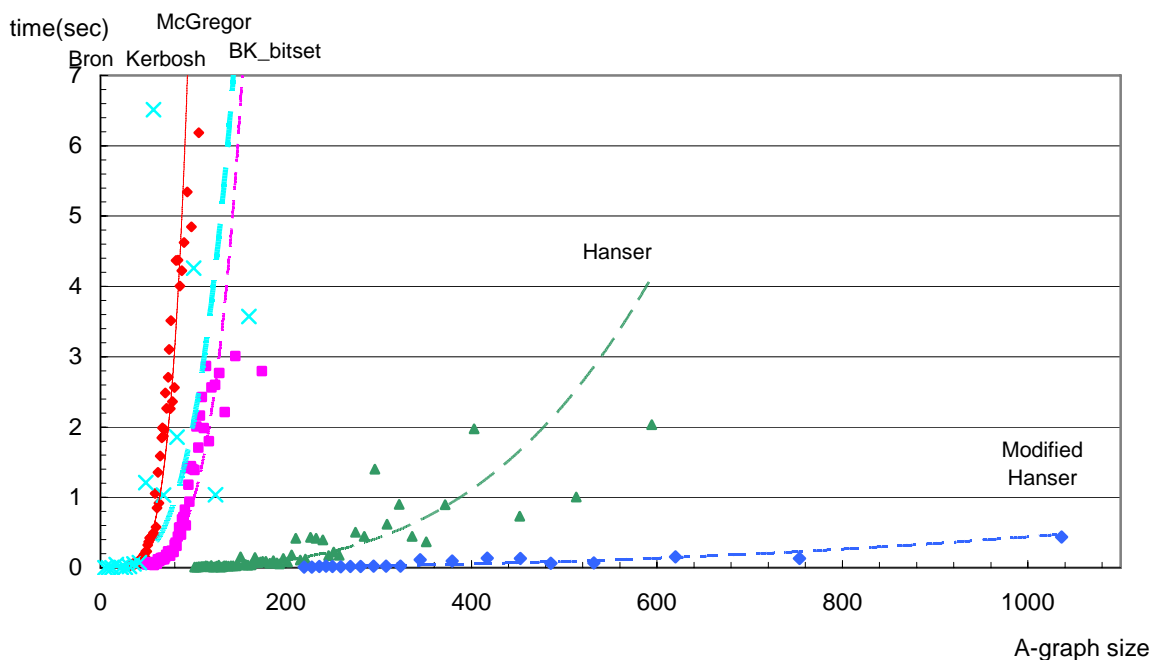


Рисунок 4 . Зависимость времени поиска решения от количества вершин в A-графе.

- × – экспериментальные точки для алгоритма Мак Грегора.
- ▲ – экспериментальные точки для алгоритма Хансера
- ◆ – экспериментальные точки для модифицированного алгоритма Хансера
- – экспериментальные точки для метода Брона и Кербоша
- ◆ – экспериментальные точки для модифицированного метода Брона и Кербоша (с использованием bitset)

Как видно из рисунка 4, модифицированный алгоритм Хансера для молекул среднего размера (при размере A-графа от 100 до 900 вершин, количество атомов в молекулах колеблется в среднем от 30 до 50) выполняет поиск MCS за время порядка секунды. Это доказывает, что данный алгоритм может использоваться в качестве эффективного инструмента при вычислениях в химических базах данных.

Для тестирования приближенного алгоритма использовалась та же база данных, что и для тестирования точных методов. В таблице 2 приведены результаты некоторых тестов для алгоритма 2DOM.

Таблица 2 Результаты тестирования приближенного алгоритма 2DOM

Количество вершин					<i>Quality ratio</i>	Время тестирования алгоритма (мс)
Molecule 1	Molecule 2	MCS approximate	MCS exact	A-graph		
24	22	20	20	103	1	0
17	17	14	15	205	0.93	15
25	36	13	13	302	1	47
31	30	22	28	431	0.79	47
22	70	19	22	644	0.86	63
37	38	11	15	716	0.73	62
49	51	41	42	827	0.98	78
35	35	27	34	877	0.79	46
54	57	19	19	1038	1	94
42	44	29	40	1078	0.73	78
47	65	35	40	1320	0.88	94

Для приближенного алгоритма большой интерес представляет «качество» найденных решений. Приведенные в таблице размеры MCS, найденные приближенным методом, позволяют найти коэффициент *Quality ratio*, который вычисляется по следующей формуле:

$$Quality\ ratio = \frac{N_{MCS}^{approximate}}{N_{MCS}^{exact}}$$

На рисунке 5 представлена зависимость коэффициента «качества» решения от размера А-графа.

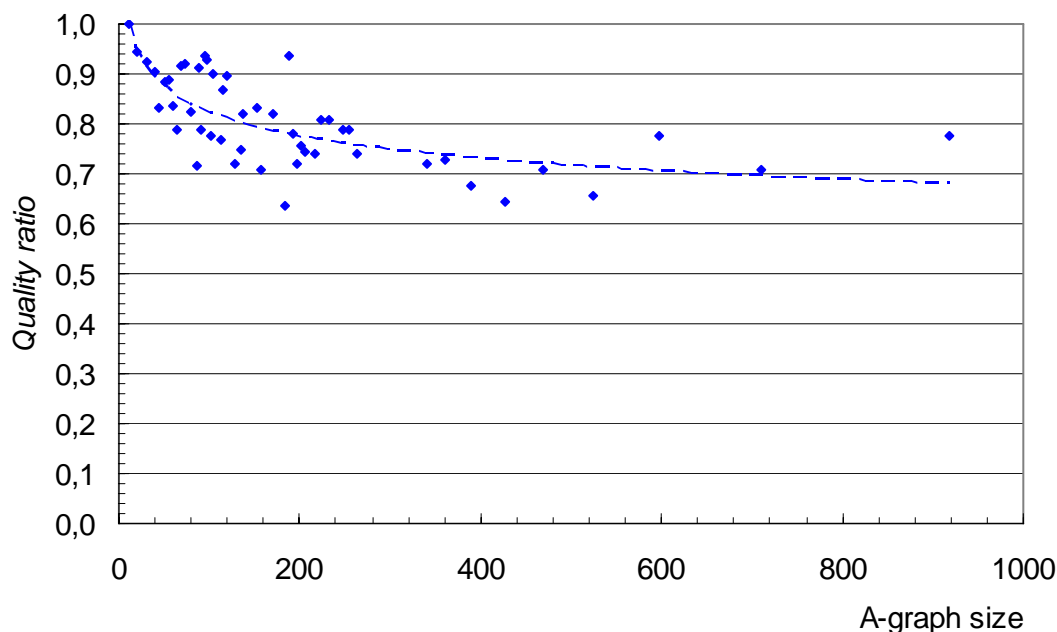


Рисунок 5 Зависимость коэффициента Quality ratio от количества вершин в A-графе.

В [11] 2DOM тестируется на 2 видах графов: р-случайных и геометрических. В статье показывается, что алгоритм находит наиболее «качественные» решения для насыщенных графов, в то время как для разреженных графов найденные решения оставляют желать лучшего. Такое положение дел связано с тем, что при нахождении максимального общего подграфа в разреженных графах любая ошибка в сопоставлении вершин может привести к «отсечению» большей части решения. Как видно из рисунка, качество решений для молекул большого размера приближается к 0.7. Однако, в основном благодаря модификации алгоритма, позволяющей выходить из локальных минимумов, качество решений для молекул среднего размера меняется в пределах 0.8 - 1

Главное преимущество приближенного алгоритма перед точными методами поиска MCS заключается в линейной зависимости времени поиска от размера молекул. На рисунке 6 показана зависимость времени поиска приближенного алгоритма от количества вершин в A-графе. Приближенный алгоритм не требует построения A-графа. Данная зависимость приведена для удобства сравнения с точными алгоритмами поиска. На рисунке 6, для сравнения результатов поиска, показаны

экспериментальные точки для алгоритма Хансера (зависимость, абсолютно аналогичная приведенной на рисунке 4).

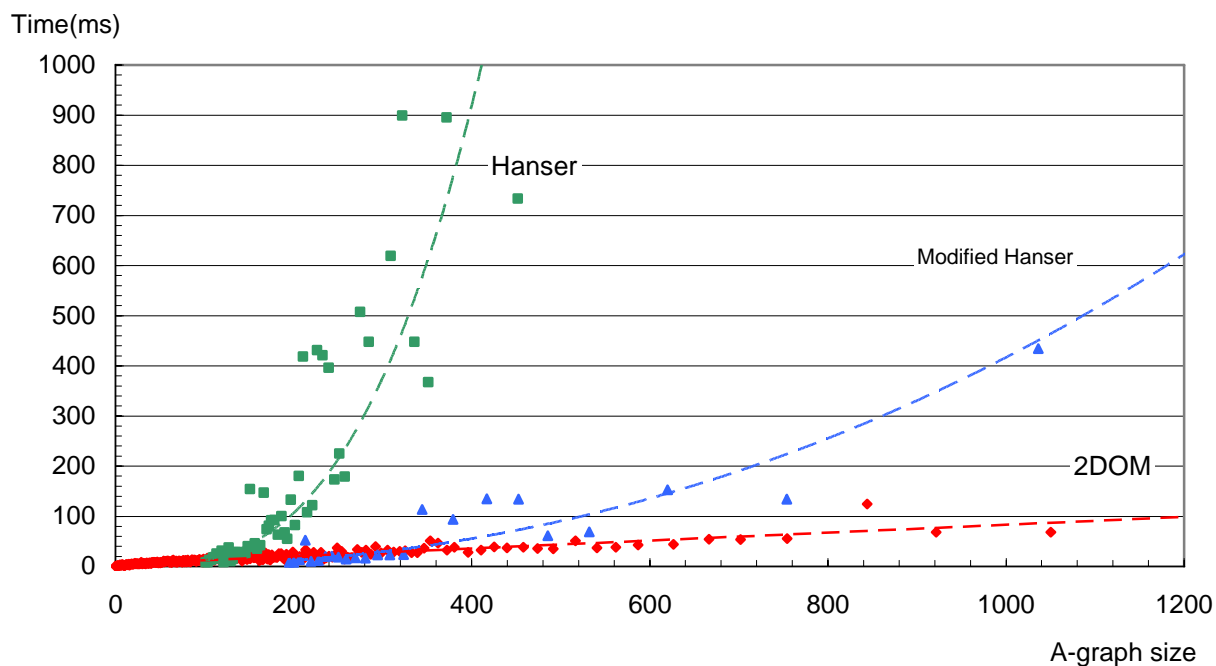


Рисунок 6 зависимость времени поиска от количества вершин А-графа для приближенного алгоритма

Как видно из рисунка, время поиска линейно зависит от размера А-графа. Что обуславливает большой интерес приближенного алгоритма 2DOM с точки зрения применения его в тандеме с точными алгоритмами в поисковых системах молекулярных баз данных.

6 Задача поиска множественной наибольшей общей подструктуры

6.1 Определение и постановка задачи

Задача нахождения наибольшего общего подграфа для множества графов или задача нахождения ядра (scaffold detection) для множества структур является широко известной проблемой в различных областях знаний, включая обработку изображений, распознавание образов, семантический анализ и биоинформатику. Например, определенные приложения в химии и биологии, которые включают в себя сравнение множественных 3-х мерных химических или протеиновых структур, изучение множественного протеин-протеинового взаимодействия или интерпретацию молекулярного спектра.

Большинство известных алгоритмов решают поставленную задачу путем сведения ее к задаче нахождения наибольшего общего подграфа для двух графов [8],[24],[25]. Приведенный ниже метод не является исключением. В нем будет использован точный алгоритм Хансера нахождения наибольшей общей подструктуры. Кроме того, в этом методе будет использоваться алгоритм для решения задачи изоморфизма подграфу.

6.2 Поиск изоморфизма подграфу

В данной работе алгоритмы поиска изоморфизма подграфу подробно рассматриваться не будут. В качестве описания будет представлен общий обзор существующих методов поиска, а также краткое описание используемого в работе алгоритма.

В 1950-х годах, с возникновением первых компьютерных баз данных, справочники химических соединений стали переноситься в электронный вид; появились химические базы данных (chemical structure databases). Тогда же впервые была сформулирована задача поиска подструктуры (substructure search). В 1957 году был разработан первый алгоритм, имеющий к этой задаче непосредственное отношение [26]. Тогда же сформировались

основные области приложения вычислительных технологий к химии: спектральный анализ, молекулярное моделирование и базы химических соединений. Задача изоморфизма графу подграфу, кроме поиска химических структур, имеет широкое применение во многих практических приложениях, таких как распознавание образов, САD-системы, сжатие видеоданных.

Дальнейшее развитие методов поиска можно проследить в [27],[28],[29]. Более оптимальный алгоритм определения изоморфизма подграфу, который до сих пор используется во многих программных продуктах, предложил Дж. Ульман [30].

В статье [31] приводится модификация Корделла алгоритма Ульмана, позволяющая существенно сократить число шагов алгоритма. В данной работе за основу был взят алгоритм Ульмана-Корделла, учитывающий автоморфизмы графов. Детальное описание и целесообразность выбора данного алгоритма приведены в [32].

Алгоритм Ульмана представляет собой рекурсивный перебор с возвратом множества допустимых состояний (под состояниями понимается множество пар вершин, задающих биективное отображение подмножеств множества вершин двух графов), начинающийся с пустого состояния. На каждом шаге перебираются пары вершин из множества пар-кандидатов, добавление которых к состоянию оставляет его допустимым, и производится рекурсивный вызов для каждого пополненного состояния. В [33] рассмотрена модификация алгоритма, использующая группы автоморфизмов графов и позволяющая существенно увеличить скорость поиска за счет сокращения перебора допустимых состояний.

Как уже отмечалось выше, задача поиска изоморфизма подграфу является NP-полной. Для молекулярных графов в этой задаче удалось существенно уменьшить сложность за счет описанных выше топологических ограничений на графы, а также за счет реализованных в алгоритме модификаций, позволяющих существенно сократить пространство поиска. Модификации

позволили достичь скорости работы для алгоритма Ульмана-Корделла на порядок быстрее скорости работы точного алгоритма Хансера.

6.3 Алгоритм определения ядра “The basket”

Рассматриваемый метод поиска множественной наибольшей общей подструктуры носит название «Корзина». Этот метод основан на двух алгоритмах: точном алгоритме Хансера нахождения наибольшей общей подструктуры, и алгоритме Ульмана-Корделла поиска изоморфизма подграфу. Итак, нам необходимо найти ядро для набора графов – максимальный подграф, который будет изоморфен каждому из входных графов.

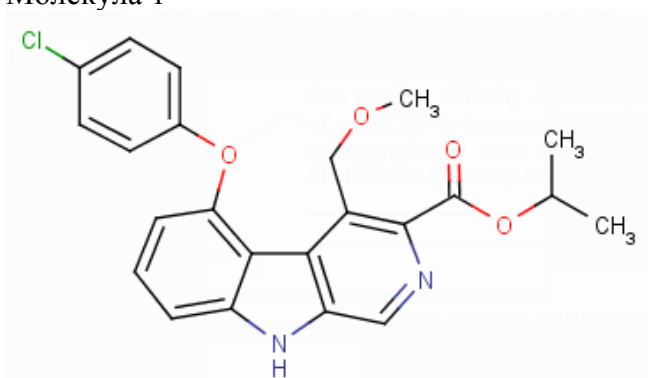
Алгоритм Хансера ищет максимальный общий подграф двух графов (MCS). Задача сводится к поиску клики у специального ассоциативного графа², построенного из входных графов. Для осуществления шага возвращения в процессе поиска алгоритм сохраняет найденные решения. Эти решения представляют собой полные подграфы в ассоциативном графе. На конечном этапе алгоритм выдает в качестве ответа максимальный из этих подграфов, то есть максимальный полный подграф (или клику) в ассоциативном графе. Рассмотрим подробнее структуру этих решений. Как уже отмечалось в определении ассоциативного графа, можно установить взаимнооднозначное соответствие между полными подграфами A-графа и общими подграфами двух исходных графов. При этом решение добавляется в том случае, если независимое множество вершин больше невозможно расширить. Для исходных графов это означает следующее: в каждом графе найден подграф, который является общим в том смысле, что подграф первого графа изоморфен второму графу и наоборот. Невозможность расширения независимого множества означает невозможность расширения этого

² Под ассоциативным графом понимается набор множеств, которые строит алгоритм Хансера на первом этапе. Всюду ниже автор использует понятие максимальное независимое множество в A-графе, а также максимальный полный подграф (клика) в A-графе. Два этих понятия считаются абсолютно аналогичными, и обозначают максимальное независимое подмножество в запрещенном множестве вершин или максимальный полный подграф в дополнении к запрещенному множеству вершин.

подграфа за счет добавления каких-либо ребер в каждом графе с сохранением изоморфизма. Таким образом, в конце работы алгоритма Хансера сохраняются все «возможные максимальные» подграфы, являющиеся изоморфными подграфами для исходных графов.

Для большей ясности, ниже будет приведен пример, в котором будут показаны все найденные решения. Пусть на вход алгоритму Хансера даны две молекулы, показанные на рисунке 7:

Молекула 1



Молекула 2:

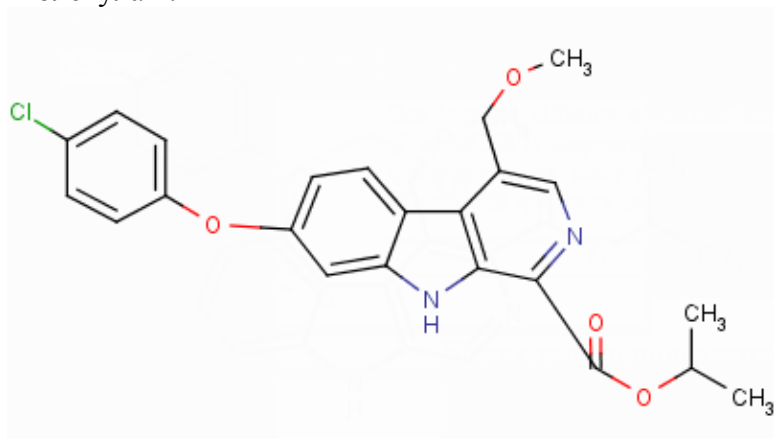


Рисунок 7 Пример молекул для поиска наибольшей общей подструктуры.

Решениями для поставленной задачи нахождения наибольшей общей подструктуры будут молекулы, показанные на рисунке 8:

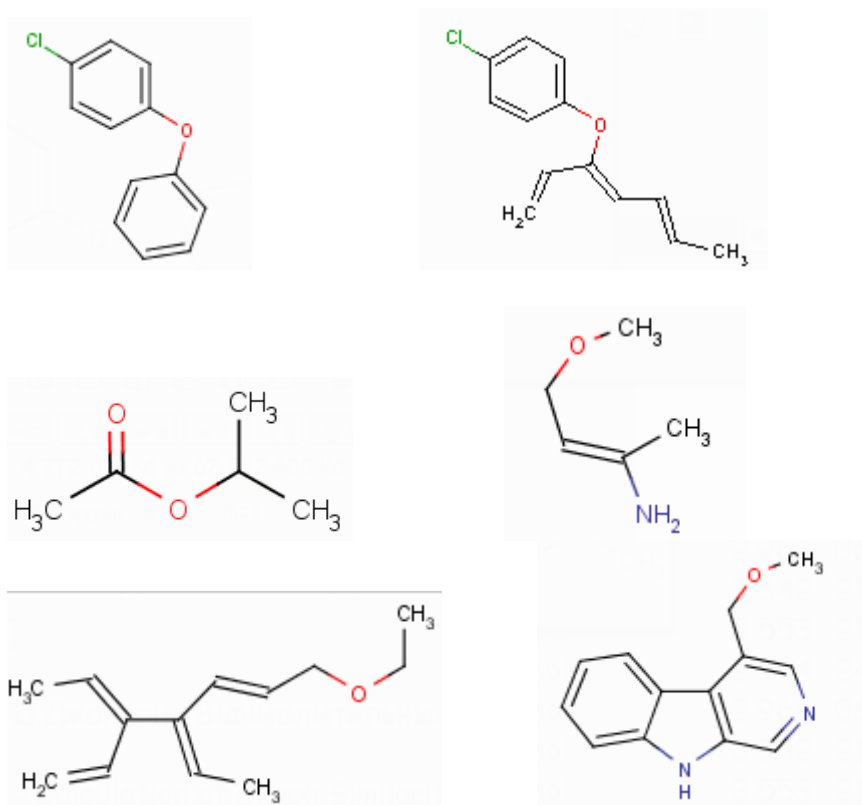


Рисунок 8 Найденные методом Хансера структуры являются изоморфными подструктурами, для молекул, приведенных на предыдущем рисунке

Следует отметить, что во множестве решений каждый из найденных графов может быть также изоморфен какому-либо графу из этого же множества.

Метод нахождения ядра “The basket” использует все решения, найденные алгоритмом Хансера, складывая их в так называемую «корзину», для дальнейшего поиска.

Для этого, в методе определяется понятие «корзина» - множество графов β , удовлетворяющее следующим свойствам:

- Во множестве β все графы попарно неизоморфны.
- При добавлении графа в корзину, производится поиск изоморфных ему графов. Граф добавляется только в том случае, если в корзине нет графов, для которых он является изоморфным подграфом.
- При добавлении графа в корзину производится поиск изоморфных ему подграфов. Если какой-то из графов в корзине является изоморфным подграфом для добавляемого графа, то он удаляется из корзины, и на его место становится добавляемый граф. При этом, если несколько

графов в корзине были изоморфными подграфами для входного графа, то все они удаляются, а на их место становится только один граф (чтобы выполнить первое условие).

Проверка изоморфизма графа подграфу в корзине осуществляется при помощи алгоритма Ульмана-Корделла.

На каждом этапе метод “The basket” ищет MCS для каждого элемента из корзины и для каждого элемента из множества входных графов, добавляя все полученные решения в корзину. Процесс повторяется до тех пор, пока множество входных графов не окажется пустым. В конечном итоге в корзине будут сложены все возможные ядра. Максимальный элемент в корзине является искомым ответом.

Алгоритм по шагам.

Пусть на вход дано множество графов $\Psi = \{G_1, G_2, \dots, G_n\}$, пустая корзина $\beta = \emptyset$

Шаг 0 Добавляем в корзину первый элемент из Ψ

Шаг 1 Берем следующий элемент из Ψ . Если все элементы перебрали, переходим к шагу 5.

Шаг 2 Берем следующий элемент из корзины. Если все элементы перебрали, переходим к шагу 1.

Шаг 3 Осуществляем проверку алгоритмом Ульмана-Корделла, является ли текущий элемент из корзины подструктурой к текущему элементу Ψ . Если да, то переходим к шагу 2, иначе к шагу 4.

Шаг 4 Ищем все решения для текущих элементов алгоритмом Хансера. Удаляем из корзины текущий элемент (взятый из корзины) и добавляем туда все решения, выполняя свойства корзины. Текущий элемент в корзине больше не нужен, так как он будет сохранен там как совокупность частей.

Шаг 5 Ищем максимальный элемент в корзине – ядро для множества исходных молекул Ψ

Очевидно, что время поиска ядра будет соизмеримо со временем поиска MCS, поэтому, выгодно отсортировать по размеру множество Ψ , чтобы искомое решение «отфильтровалось» уже на первых этапах работы алгоритма.

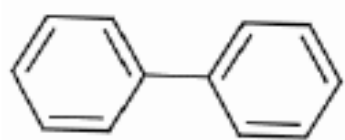
6.4 Тестирование алгоритма нахождения ядра.

Тестирование проводилось следующим образом. Определенным образом выбирались молекулы – ядра. Алгоритмом Ульмана-Корделла в базе данных молекул производился поиск молекул, для которых выбранные ядра являлись подструктурами. Неформальная постановка задачи заключалась в поиске множественной наибольшей общей подструктуры для тысячи молекул за время, порядка секунды. Поэтому, ядра выбирались таким образом, чтобы в базе данных нашлось структур для тестирования более 1000.

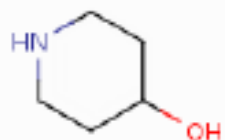
Метод нахождения ядра “The basket”, при поиске производит предварительную сортировку молекул по размеру. Для каждого из ядер приводится по два теста. Первый тест – поиск ядра для всех найденных в базе молекул. Второй тест – поиск ядра для 1000 молекул, которые расположены в конце предварительно отсортированного множества. Таким образом, тестировался «самый худший» для алгоритма вариант.

Тестировались два алгоритма поиска ядра. Алгоритм, использующий в качестве основы модифицированный алгоритм Хансера, сравнивался с тем же алгоритмом, но использующим метод Хансера без модификаций.

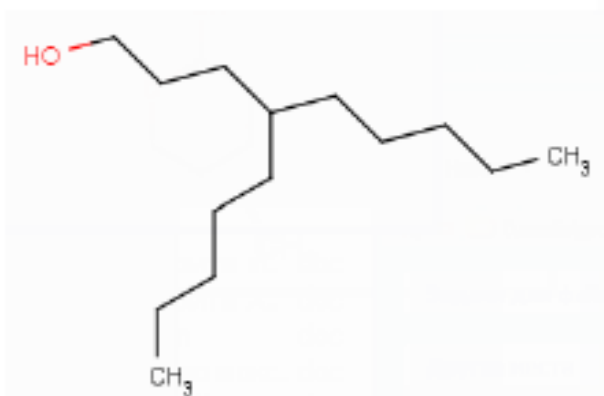
На рисунке 9 показаны молекулы-ядра, предложенные для поиска.



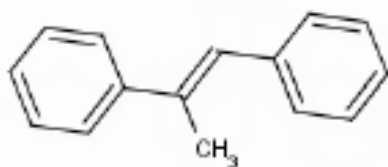
Scaffold 01 $N^{SUB} = 11407$



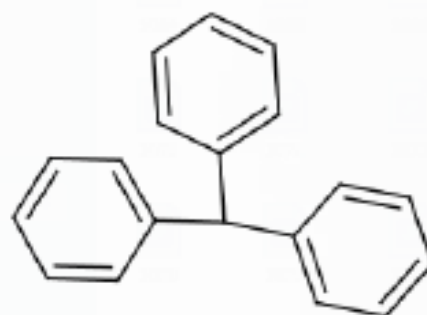
Scaffold 04 $N^{SUB} = 1090$



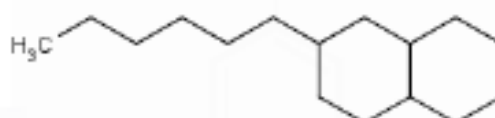
Scaffold 05 $N^{SUB} = 2350$



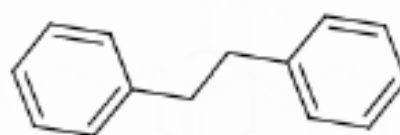
Scaffold 08 $N^{SUB} = 1854$



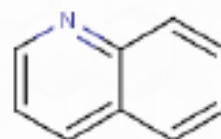
Scaffold 02 $N^{SUB} = 1279$



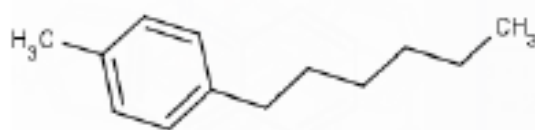
Scaffold 03 $N^{SUB} = 1680$



Scaffold 06 $N^{SUB} = 3107$



Scaffold 07 $N^{SUB} = 1931$



Scaffold 09 $N^{SUB} = 1386$

Рисунке 9 Молекулы-ядра, предложенные для поиска в базе данных. N^{SUB} - количество молекул, найденных в базе данных

Тестирование проводилось на процессоре Mobile Intel Core 2 Duo 1666 MHz с 2 GB RAM в системе Windows XP. Таблица результатов тестирования приведена ниже.

Таблица 3 Результаты тестирования алгоритма поиска ядра для множества молекул.

№ ядра	Тест №1 (время мс)			Тест №2 (время мс)		
	Кол-во молекул	Hanser	Modified Hanser	Кол-во молекул	Hanser	Modified Hanser
01	11407	875	447	1000	797	109
02	1279	110	55	1000	922	57
03	1680	26579	9780	1000	>1h	40218
04	1090	63	22	1000	47	21
05	2350	9219	2241	1000	>1h	2081
06	3107	265	109	1000	594	58
07	1931	156	98	1000	406	102
08	1854	109	53	1000	250	43
09	1386	125	90	1000	141	65

Объяснение результатов.

Из структуры метода поиска очевидно, что конечный результат каждого тестирования будет зависеть от размера первой и второй молекулы во множестве, так как ими будет определяться количество подграфов в корзине. Важно также отметить, что время поиска точного алгоритма полностью определяется первым сравнением. Дальнейшее увеличение молекул в корзине мало влияет на скорость поиска, так как их размеры уменьшаются пропорционально количеству. Рассмотрим для примера 3-ое ядро в таблице. Время поиска в тесте №1 для точного алгоритма Хансера равняется 26.6 секунды. При этом 2.5 секунды тратится на нахождение MCS для первых двух молекул и, соответственно, $26.6 - 2.5 = 24.1$ секунд на нахождение ядра в оставшихся 1678 молекулах. Время поиска для теста №2 больше часа. Это время полностью определено первым сравнением. (Молекулы для 3-го и 5-го ядра обладают свойством – при поиске MCS они имеют большое количество решений). Как видно из таблицы 3, модификация алгоритма Хансера позволила существенно увеличить скорость поиска.

7 Выводы

Достигнутые результаты.

В работе проведено исследование различных алгоритмов, имеющих отношение к задаче поиска наибольшей подструктуры. Были проанализированы недостатки имеющихся методов и предложены модификации для достижения большей эффективности.

Разработан новый, составленный из уже известных алгоритмов на графах, эффективный метод решения задачи нахождения ядра для множества структур.

В целях сравнения, были реализованы и ранее известные подходы. Проведен анализ времени работы для статистических данных, подтверждающий преимущество разработанных методов.

Все приведенные в работе алгоритмы были имплементированы и включены в многофункциональный картридж для СУБД Oracle. Картридж протестирован на соответствие нормам, предъявляемых к коммерческим программным продуктам.

8 Используемая литература.

- [1] Н. Кристофидес. Теория графов. Алгоритмический подход. Издательство «Мир» 1978 г.
- [2] М. Гери Д. Джонсон. Вычислительные машины и труднорешаемые задачи
- [3] А. Ахо Дж. Хопкрофт, Дж. Ульман. Построение и анализ вычислительных алгоритмов.
- [4] Берж К. Теория графов и ее применение.
- [5] Л.Е. Захарова. Алгоритмы дискретной математики 2002 г.
- [6] Т. Кормен, Ч. Лейзерсон. Алгоритмы. Построение и анализ.
- [7] H. Bunke, P. Foggia, A comparison of algorithms for maximum common subgraph on randomly connected graphs.
- [8] John W. Raymond and Peter Willett. Maximum Common Subgraph ISomorphism Algorithms Matching Chemical Structures *Journal of Computer-Aided Molecular Design*, 16: 521–533, 2002
- [9] Trinajstic, N. (1992) *Chemical Graph Theory*. CRC Press, Boca Raton, FL
- [10] Bernhard Rohde. Novartis Pharma AG, Basel. Reaction Atom-Atom Mapping. US Daylight Users' Meeting 1999
- [11] Nobuo FUNABIKI and Junji KITAMICHI. A Two-Stage Discrete Optimization Method for Largest Common Subgraph Problems. *IEICE TRANS. INF. & SYST.*, VOL. E82-D, NO. 8 AUGUST 1999
- [12] Horaud, R. and Skordas, T., *IEEE Trans. Pattern Anal. Mach. Intell.*, 11 (1989) 1168.
- [13] Pelillo, M., Siddiqi, K. and Zucker, S.W., *IEEE Trans. Pattern Anal. Mach. Intell.*, 21 (1999) 1105.
- [14] Shearer, K., Bunke, H. and Venkatesh, S., Video Indexing and Similarity Retrieval by Largest Common Subgraph Detection Using Decision Trees, No. IDIAP-RR 00-15, Dalle Molle Institute for Perceptual Artificial Intelligence, Martigny, Valais, Switzerland, 2000.
- [15] Kann, V., On the Approximability of NP-Complete Optimization Problems, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden, 1992.
- [16] Cone, M., Venkataraghavan, R. and McLafferty, F., *J. Am. Chem. Soc.*, 99 (1977) 7668.
- [17] Barrow, H. and Burstall, R., *Inf. Proc. Lett.*, 4 (1976) 83.
- [18] Raymond, J., Gardiner, E. and Willett, P., *Comput. J.*, in the press.
- [19] McGregor, J., *Software Pract. Exper.*, 12 (1982) 23.
- [20] Wong, A. and Akinniyi, F., *Proc. Int. Conf. Systems, Man and Cybern.*, Bombay & New Delhi, India, 1983, pp. 197.
- [21] Brown, R.D., Jones, G., Willett, P. and Glen, R., *J. Chem. Inf. Comput. Sci.*, 34 (1994) 63.
- [22] Funabiki, N. and Kitamichi, J., *IEICE Trans. Inf. & Syst.*, E82-D (1999) 1145.
- [23] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.
- [24] R. Horaud, T. Skordas, Stereo correspondence through feature grouping and maximal cliques, *IEEE Trans. Pattern Anal. Mach. Intell.* 11 (11) (1989) 1168_1180.

- [25] K. Shearer, H. Bunke, S. Venkatesh, Video indexing and similarity retrieval by largest common subgraph detection using decision trees, No. IDIAP-RR 00-15, Dalle Molle Institute for Perceptual Artificial Intelligence, Martigny, Valais, Switzerland, 2000.
- [26] Ray, L. C., Kirsch, R. A. Finding Chemical Records by Digital Computers. *Science* 1957, 126, 814-819
- [27] Crowe, J. E., Lynch, M. F., Town, W. G. Analysis of Structural Characteristics of Chemical Compounds in a Large Computer-Based File. I: Non-Cyclic Fragments.. *J. Chem. Soc* 1970, (C), 990-996
- [28] Graf, W., Kaindle, H. K., Kniess, H., Schmidt, B., Warszawski, R. Substructure Retrieval by Means of the BASIC Fragment Search Dictionary Based on the Chemical Abstracts Service Chemical Registry III System. *J. Chem. Inf. Comput. Sci* 1979, 19, 51-55
- [29] Evans, L. A., Lynch, M. F., Willett, P. Structural Search Codes for On-line Compound Registration. *J. Chem. Inf. Comput. Sci.* 1978, 18, 146-149
- [30] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, 1976.
- [31] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1367–1372, 2004.
- [32] Карулин Б. Выпускная работа бакалавра. Поиск изоморфизма графа подграфу 2006г.
- [33] Рыбалкин М. А. Выпускная работа бакалавра: Симметрии в задаче изоморфизма графа подграфу, 2008