Mining Maximal Frequent Patterns in Molecular Data with Look-ahead Pruning

Arpit Gattani
gattani@cc.usu.edu

Abstract

Mining maximal frequent patterns in large database is now a well attended problem in data mining field. In this paper we present an algorithm which mines maximal frequent substructures in a molecular data. The algorithm uses bottom-up depth first search approach to scan data and find maximal frequent substructures. Algorithm works on the concept of look-ahead pruning to restrict the candidate generation and unnecessary scans of database. This algorithm also uses the concepts of dynamic reordering of candidates and support lower bound to efficiently find even the long patterns in molecular data. Initial results on anti HIV-1 data show that this algorithm works better than the simple Apriori algorithm in finding long frequent patterns.

Keywords

Frequent substructures, maximal frequent substructures, dynamic reordering, look-ahead pruning.

1. Introduction

Mining frequent patterns in databases is the most fundamental process in Data Mining. Many advanced data mining tasks like association rules mining and clustering requires frequent patterns to process. Since the introduction of Apriori algorithm [1] most of the algorithms proposed after were the versions of Apriori. Apriori works on the simple rule of generating all the candidates for frequent itemset at a level and then pursue with it till we either find its superset as frequent or infrequent. So it necessarily generates all the 2^n candidates where n is the length of frequent pattern. Apriori and its variant works well for smaller data and frequent

itemsets. But as the size of frequent itemsets increases, it becomes very inefficient to mine even moderate size itemsets. Many new approaches are recently proposed [2, 5, 6, 7, 8, 11] to find long patterns in large databases.

This paper proposes an algorithm which efficiently mines the maximal frequent substructures in molecular data. Mining frequent molecular fragments can useful in biological and pharmaceutical world. Some of its uses are to help in making more efficient drugs and identifying mutations in the disease causing microbes. Though our algorithm is a level vise algorithm, its efficiency is because of the reason that it restricts the generation of candidates by not expanding all the candidates and pruning all the generated candidates efficiently by look-ahead strategy. This algorithm works on bottom up depth first search on a lexicographical ordered tree. Also it make use of the dynamic reordering of the candidates [5] and generation of dynamic support lower bound [5] to mine maximal frequent substructure faster. Depth first approach is considered as more memory bound approach but the reason for the efficient working of this algorithm is that we now have the memory in the order of gigabytes. So the only concern we are left with is to limit the CPU time in our algorithm.

In this paper, we used a sample of anti HIV-1 data available on http://dtp.nic.nih.gov . Molecular structures are used in SMILES format. SMILES representation is used as it is very simple and easy to use and at the same time keep the compounds representation complete.

1.1 Related works

There are basically two types of strategies proposed to mine maximal frequent patterns in databases. First are the kinds of algorithms which produce a frequent itemset of length k in k^{th} pass and strictly not before [1, 9]. These algorithms suffer with the high complexity levels in candidate generation. MOLFEA [9] is one of the

algorithms which find frequent molecular substructures in anti HIV-1 database with defined constraints. Though it finds the accurate frequent substructures, it generates a lot of candidates at each level and also doesn't take the pruning information of previous step to next step. Simple Apriori has been tried to modify by Park at al. [11], by using hashing to store candidates and thus allowing them to be identified quicker. But again it doesn't generate the frequent pattern before considering all the subsets.

Another type of algorithm deals with the graph based methods. It represents each frequent itemset at a level as a node of the graph and generates the subsequent superset of that node. FP Tree structure [8] was the proposed to find the frequent patterns in large databases without candidate generation. Lately many algorithms are proposed to find maximal frequent patterns which are based on Lattice and Tree methods. MaxClique and MaxEclate [12] are two such Lattice based methods which works in bottom up fashion to find frequent patterns and uses vertical database representation. These strategies have takes into account the look-ahead property but have drawback like inefficient initialization. Max-Miner and Pincer-Search [5, 10] are two most talked about strategies. Though both works on same strategy of finding the maximal frequent patterns with limited candidate generation and look-ahead pruning, Max-Miner takes into account the depth first search while Pincer Search works on breadth firs search with top down pruning.

This paper is organized as follows:

In section 2 we described mining frequent pattern problem and maximal frequent patterns. Section 3 explains the SMILES representation of chemical compounds. In section 4 we described the lexicographical representation of data and discussed the benefits of reordering of candidates in tree. In section 5 we explained the algorithm

with example. In section 6 and 7 we described the implementation details of the algorithm and the results on anti viral screen data.

2. Mining Frequent Substructure

A substructure is defined as a subset of a molecular structure. From data mining perspective it is same as the itemset in transactional data or the pattern in any serial data. In this paper we have used the words substructure, itemsets, and patterns interchangeably. In a given dataset, a pattern is called frequent if it appears more times than a user defined support value. Association rules are found by a simple L => R form where each L and R represents a frequent pattern in the database which satisfies the minimum support and minimum confidence value.

2.1. Maximal Frequent Substructure

A frequent substructure, ***S*** can be said as maximal frequent substructure if there exist no superset of ***S*** which is frequent. In simple words maximal frequent substructure can be defined as the longest sequences of molecules which are frequent. Mining maximal frequent substructure limits the generation of redundant association rules and hence makes the rule generation more interesting and logical. Also it allows the frequent substructure mining process less complex by allowing candidate pruning at early stages.

3. SMILES Representation of Molecules

SMILES, Simplified Molecular Input Line Entry System, is a simple yet strong language to represent molecular structures. It gives a fairly understandable formats to even very complicated structures. A chemical compound representation in SMILES follows following notations:

- SMILES represent compounds in string format. Every element is represented by its typical chemical symbol, like C, N, O, H, Cl etc.

- Notation for single bond is '-', for double bond is '=', for triple bond id '#' and for aromatic bond is ':'.
- Aromatic elements are represented in lower case. Like aromatic carbon in benzene ring is shown as 'c'.
- In most general cases hydrogen atoms, single and aromatic bonds are not written.
- A cyclic structure can be represented by marking the first and the last element of the ring with same number.
- Elements inside the brackets denote the side-structures.

Figure below shows the complete representation of a compound in SMILES format.

Nc1ccc(O)cc1 ≡

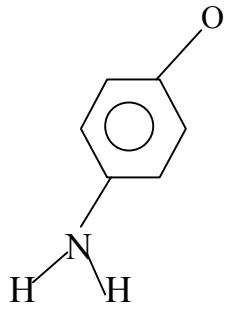

Figure 1: An example of SMILES representation of chemical compound.

4. Lexicographical Representation of Molecular Data.

This section explains the representation of the molecular data in lexicographical tree [3] format. Motive behind representing data in lexicographically ordered tree is that it helps in depth first searching of substructures. Each node of the tree consists of the frequent substructure. The root of the tree is NULL node. All children of root node are the frequent candidate at first level. This means all the single element in the data which are identified as frequent. The order of the nodes at a level is defined as $i \leq_L j$, where i is a node occurs before j in the tree.

4.1. Dynamic Node Ordering

Dynamic reordering [5] of the nodes at each level depends on the frequency of that node in previous level. Nodes are arranged in the order of the least frequent node first. This is done in order to find maximal frequent substructure early in our search. This happens because our search follows two basic principles of frequent pattern mining:

- Supersets of all infrequent itemsets are infrequent.
- Subsets of all frequent itemsets are frequent.

So we tried to find all the superset of less frequent substructures first as they generate maximal frequent substructures early and they restrict the expanding of frequent nodes.

5. Algorithm

This section shows the working of the algorithm to find maximal frequent substructures in molecular data. The lexicographically sorted tree is traversed in depth first order. To be more specific, our algorithm searches as follows:

To start with we have a frequent substructure set as empty. Minimum support is provided by the user and candidate set is also empty. After this data scan is done to find frequent substructure. Each frequent substructure is now a candidate and kept in candidate set. Then all candidates in candidate set are arranged in order of least frequent candidate first. Then we pick the first candidate in candidate set and expand it by adding next item to it. An item can be any one of these, an element, a bond, a number for cyclic elements or a bracket.

So if value of node a node N_i at i^{th} level is {c:c:c-O=} and its extensions are E_i = {H, O, C} than the successors of N_i are N_i + { i : i $\in$ Ei }. Figure 3 shows the expansion of tree. After expanding a node, pruning phase starts. Pruning can be done in three ways. First, all those nodes in the tree which are not frequent can be pruned.

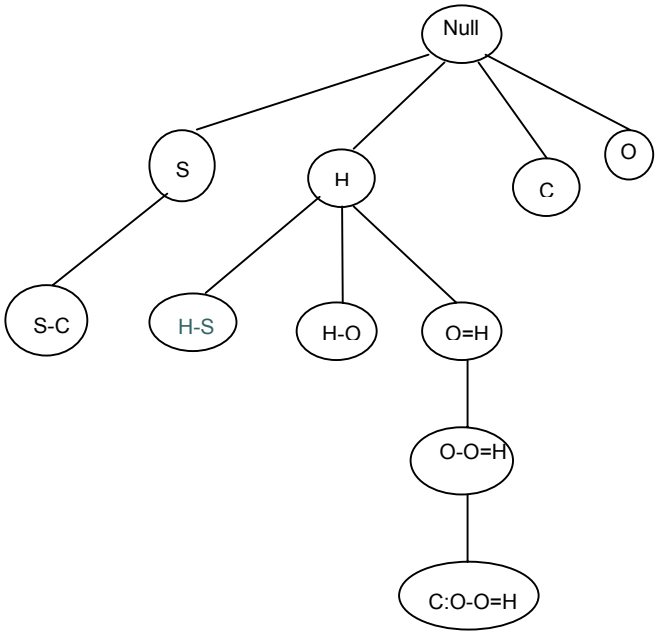

Figure 3: Expansion of the tree for compounds C:O-O=H-S,
C:O-O=H-O, S-C:O-O=H-S-C-H.

This is simple support based pruning. Second, all the nodes from the tree can be pruned if during previous levels we have found any of the superset of that node as frequent. But this type of pruning becomes inefficient when there exist long similar substructures which differ only at the end. Third type of pruning is actually the restriction phase in our algorithm. It deals with all the substructures which are found as infrequent. We need not to expand it as we know that none of its superset would be frequent. Pseudo code for the algorithm is described below in figure 4.

5.1. Example

In this example we consider the same compounds shown in figure 1. Let support is defined as 50%. This means a substructure has to be in at least 50% of the compounds to be said as frequent. To make this less complex we haven't considered bonds, brackets and ring numbers to be the part of the expanding node. We start with scanning data for the candidates consisting of single atom. Figure 4.1 shows the structure of the tree after level 1. All the nodes are then arranged in order of least

support value lower than defined support. Node *N* is pruned after this step as it occurs only once in the database.

```
Miner (database D, candidateset C, support S, frequentitemset FSS)
    Sup = minSup;
     k = 1;
    Ck-1 = { i | i ∈ dataset };
    FSS0 = NULL;
  while Ck != empty
    read the database and count the frequency of each i ∈ Ck;
    Ck = all i in Ck-1 having count ≥ minSup;
    arrange all i in Ck in increasing order of count, i.e. candidate with least
    count first;
    generate new Sup = genSup ();
    expand the first candidate in table. Let it be 'a';
    generate all of the supersets of a;
    prune all those supersets of a, which contain any infrequent item;
    candidates in Ck whose superset has been found frequent while
    expanding a, were not expanded;
  repeat till Ck != NULL;
```

Figure 4: Pseudo code for mining algorithm.

At level two, node *S* is expanded. Since the child of *S* is infrequent, it is pruned and not expanded further. Then next node in order is picked up and expanded. *H-S*, *H-O* and *O=H* are the children of *H*. Here couple of things is to be pointed. We have taken the canonical structure of a compound in to account by expanding the node in both directions. So any canonical structure will occur only once in our frequent fragment set at the end. Second, since we have expanded *H* with *O=*, we won't expand *O* node with the same suffix. This allows us to restrict our expansion and hence candidate generation. Here we keep *H-S*, *O=H* in our frequent substructure set and prunes the other children. We have got our first maximal pattern as there is no expansion

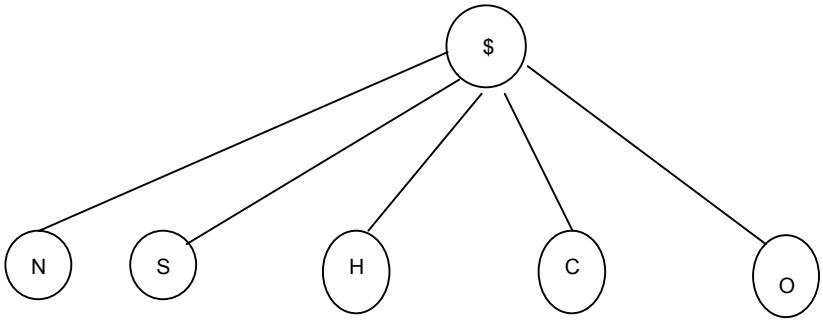

Figure 4.1: Search tree for level one.

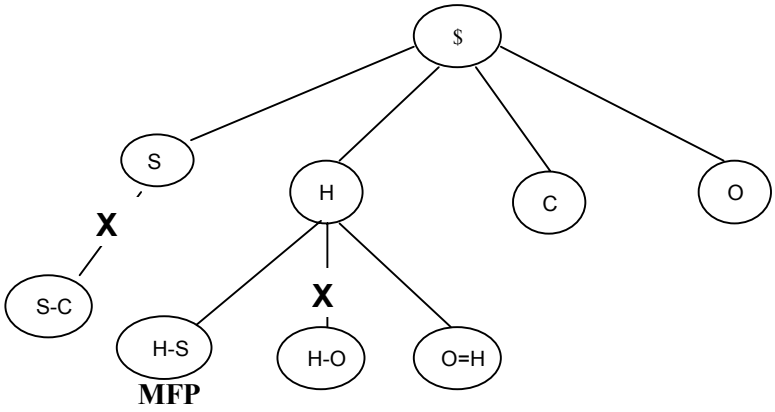

Figure 4.2: Search tree at level two. *S* is expanded and node *S-C* is pruned as it is infrequent. H is expanded to *H-S*, *H-O* and *O=H*. *H-O* is pruned. *H-S* is kept in FSS.

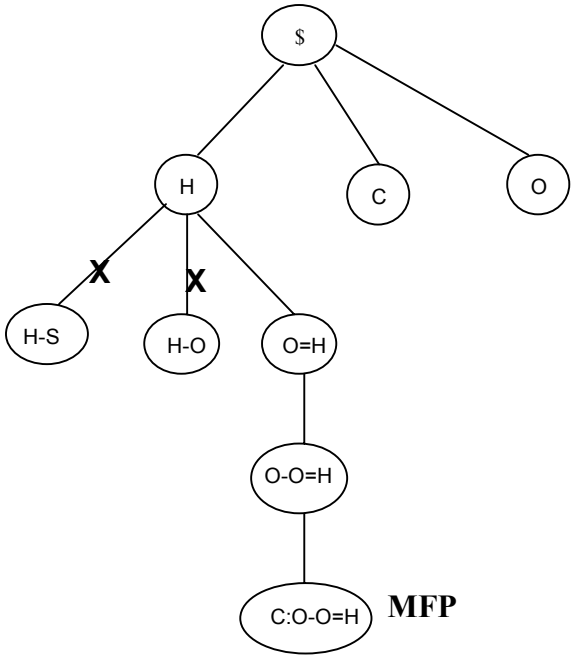

Figure 4.3: Complete expansion of the tree. Node C and O were not expanded as all the possible substructures were found in previous node's expansion.

possible with *H-S*. Then we expand O=H with its successors, getting *C:O-O=H* as the maximal frequent substructure. Now since we have expanded all the possible substructure in our database, we need not to go to any other node left to expand. This allows us to stop at the point we have mined all the possible substructures in the database.

6. Implementation

Data structure used to implement the lexicographical tree is priority stack with locator pattern. Reason behind using a stack is that it allows quick search of the node to be expanded and hence allows the depth first search to be more efficient. Also the reordering of nodes according to their respective count is efficient with priority queue implementation. A count array is maintained to store the count of each candidate in queue. Algorithm was implemented in C.

The data set was taken from the 43576 compounds from the Development Therapeutics Program's AIDS anti viral screen database. We have conducted the experiments on a sample of 300 compounds from this database. Sample compounds were taken irrespective of their classified nature of confirmed active (CA), confirmed moderately active (CM), confirmed inactive (CI). So the results were not very much accurate in terms of the usefulness.

7. Results

Results of initial implementation of the algorithm are given below in table 1 and table 2. Table 1 show the occurrence of single atom in number of compounds. Results shows that C atom occurs most in 100% of elements followed by O, N and S atom. We have not included bonds, brackets and numbers in our results as frequent item. Table 2 shows the 6 most frequent maximal substructures. Since the implementation is done on a sample data of 300 compounds from about 50000 compounds, results do

not stand with much usefulness from real world prospective. Initial support value was kept 10% for finding frequent substructure and for maximal frequent substructures lower bound was defined as 2%. The reason we have kept lower bound for support so low is that the data we have used is very skewed and only 1.3% of the whole data is supposed to be active.

Atom	Frequency
C	100%
O	91%
N	68%
S	53%

Table 1: Shows the frequency of individual atoms in the database. Frequency is defined with the occurrence of the atom in number of compounds.

Maximal Frequent Substructure	Frequency
S(Sc1ccc	3%
[N+](=O)([O-])c1ccc	3%
C(=O0(c1ccc	2.33%
C(=NNC(	2.33%
C(Occcc)O	2%

Table 2: Maximal frequent substructures found in the sample data consisting of 300 compounds from DTP AIDS anti viral screen database.

8. Conclusions

In this paper we tried to develop an algorithm to find maximal frequent substructure in molecular data with use of depth first traversal and concept like intelligent pruning.

The depth first search strategy helps in mining long patterns early in search and thus helps in restricting our search to few node expansions. We have also applied the concept of dynamic reordering of candidates at each level which helps us in pruning infrequent patterns efficiently.

Present algorithm can be enhanced by using better counting technique to count the frequency of candidates. Counting techniques like Bitvector can improve the performance of algorithm. Future work can also focus on modifying the algorithm according to different data sources like genetic data, transaction data and other chemical data.

9. Acknowledgements

We would like to thank Dr. SeungJin Lim for his helpful guidance. Also we would like to thank Cristoph Helma for providing me with data to perform experiments.

References

[1] R. Agrawal, T. Imielinski, A. Swami. Mining Association Rules between Sets of Items in Large Databases. *In Proc. of the 1993 ACM-SIGMOD Conf. on Management of Data, 207-216.*

[2] R. C. Agrawal, C. C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. In *Knowledge Discovery and Data Mining*, pages 108–118, 2000.

[3] R. C. Agrawal, C. C. Aggarwal, V. V. V. Prasad. A Tree Projection Algorithm for Finding Frequent Itemsets. *Journal on Parallel and Distributed Computing.*

[4] C. Aggarwal. Towards Long Pattern Generation in Dense Databases.

[5] R. J. Bayardo. Efficiently mining long patterns from databases. In *ACM SIGMOD Conf. Management of Data*, 1997.

[6] D. Burdick, M. Calimlim, J. Gehrke. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. *Proceedings of the ICDE Conference, 2001.*

[7] K. Gouda, M. J. Zaki. Efficiently Mining Maximal Frequent Itemsets. *Proc. Of the IEEE Int. Conference on Data Mining, San Jose, 2001*

[8] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00)*, pages 1–12, Dallas, TX, May 2000.

[9] S. Kramer, L. De Raedt, C. Helma. Molecular feature mining in HIV data. *Proc. 7th Int. Conf. on Knowledge Discovery and Data Mining (KDD-2001, San Francisco, CA), 136-143. ACM Press, New York, NY, USA 2001.*

[10] D. I. Lin and Z. M. Kedem. Pincer search: A new algorithm for discovering the maximum frequent set. In *Extending Database Technology*, pages 105–119, 1998.

[11] J. S. Park, M. Chen, P. S. Yu. An Effective Hash Based Algorithm for Mining Association Rules. *In Proc. ACM-SOGMOD, Intl. Conf. Management of Data, May, 1995.*

[12] M. J. Zaki, S. Parthasarthy, M. Ogihara, W. Li. New Algorithms for Fast Discovery of Association Rules. *In Proc. Of the Third Int'l Conf. on Knowledge Discovery and Data Mining, 1997, 283-286.*

[13] Q. Zou, W. W. Chu, B. Lu. SmartMiner: A Depth First Algorithm Guided by Tail Information for Mining Maximal Frequent Itemsets.