Research on Parallel Frequent Pattern Discovery Based on Ontology and Rules

Yuhan Gu
Undergraduate, Computer Science and Technology College
University of Electronic Science and Technology, UESTC
Chengdu, China
guyuhns@163.com

Ming Sun
Associate Professor, Computer Science and Technology College
University of Electronic Science and Technology, UESTC
Chengdu, China
sunm@uestc.edu.cn

Nardi, et al, 1998) and the $DL + log$ (R. Rosati, 2006) isomerism system have been brought out.

II. DEFINITIONS:

The knowledge base of the frequent pattern discovery in this paper is heterogeneous system $Data\log^{SHIQ}$, which is combined by $SHIQ$ and $Data\log$ rules.

**Definition 1:** The $SHIQ$ knowledge base $G$ can be defined as: $G = (T, A)$ and it consists of $TBox$ $T$ and $ABox$ $A$.

In this definition, the $TBox$ $T$ is the term set of description field and consists of concept set and rule set. The $ABox$ $A$ is the assertion set of description field and consists of conceptual assertion and role assertion. And the basis of the establishment of the $Data\log^{SHIQ}$ is heterogeneous $Data\log$ rules, which can be defined as the follows:

**Definition 2:** the representation of heterogeneous $Data\log$ is:

$$h \models a_1, a_2, ..., a_n \& b_1, b_2, ..., b_m$$  \hspace{1cm} (1)

In the formula (1), $h$, $a_1, a_2, ..., a_n$ is the rule atom, $b_1, b_2, ..., b_m$ is conceptual atom or rule atom which consists of concept of description logics or role.

The traditional knowledge base, which combines the ontology and rule, includes $TBox$, $ABox$ and rule set, but the observation set which contains application rules and consistent facts is a independent conception and has no intersection with traditional knowledge set. In order to guarantee the consistency of the following frequent pattern discovery, the knowledge base defined in this paper is based on the $Data\log^{SHIQ}$. It contains $ABox$, $TBox$, rule set and observation set, which is called hybrid knowledge base.

**Definition 3:** The definition of $Data\log^{SHIQ}$ is as follows: hybrid knowledge base $B$ consists of $Term\textit{ Set}$ ($TBox$), assertion set $A$($ABox$) of ontology and rule set $R$, observation set $O$ of rule. Its framework is shown in the “Fig.1”:

![Figure 1: Hybrid Knowledge Base Framework](image)

---

**Abstract—** Frequent pattern discovery is a hotspot of knowledge discovery, which is based on the hybrid knowledge base that consists of ontology and rule logic. However, the mining efficiency of it is too low and unsatisfying, since the reasoning process needs large quantities of computing. Thus, we need to develop the parallel algorithm and architectures to solve this problem. In this paper, we come up with a parallel Inductive Logic Programming (ILP) frequent pattern discovery method based on MapReduce Framework, and then establish the Trie Tree by node expansion algorithm. Finally, we distribute the various computational pattern detection processes to each parallel computing nodes in the Frequent Pattern Tree (FPT) and get the frequent pattern. This way can also eliminate the semantic redundancy, which is judged by support degrees. Consequently, by the verification in experiments, we find this frequent pattern discovery algorithm can improve the efficiency of pattern discovery in a large degree.

Keywords-component: ontology, rules, parallel computing, frequent pattern discovery

I. INTRODUCTION

The parallel computing of Frequent Pattern Discovery is an important branch of the data mining, it can select the potential and valuable pattern and rules from various servers and databases. The classical frequent pattern discovery algorithm is the Apriori Algorithm (R. Agrawal & H. Mannila & R. Srikant, 1996) and its varieties. It uses the fundamental method based on the key-value attributes, which aims at processing single relations. Moreover, for the researches concerning the frequent pattern discovery of multiple relationships, the ILP (H. Blockeel & J. Shavlik & P. Tadepalli, 2008) is the first algorithm that is relatively mature in this field. At the same time, the studies concerning the hybrid logical knowledge base was on its way. In general, there are two common methods to combine the ontology logic and reasoning logic: one is called isomorphism method, another is called isomerism method. The isomorphism method combines the ontology and rule into a specific logical language and share the same model to build up coherent knowledge base and reasoning mechanism. In this field, MeiJing uses the former method to bring about $DL + log$ that is the semantical security of isomorphism security, which is the basis of decidable language $ALC^2$. On the other hand, the isomerism method views the ontology and rule as two separately components which own their independent logical reasoning. The $Data\log^{SHIQ}$ (B. Motik & U. Sattler & R. Studer, 2005) based on the AL-log (F. M. Donini & M. Lenzerini & D.
As it shown in the Fig.1, hybrid knowledge base mainly contains the conceptual level and case level. And the conceptual level consists of term set T and rule set R, while the case level consists of assertion set A and observation set O. The conceptual pattern P is the mining result of term set and rule set. Under this circumstance, we can judge whether this pattern is frequent pattern or not according to the cover tests of observation set and assertion set.

**Definition 4:** Data log^SHIQ^ hybrid knowledge base is B = (L,R), the L is description logic knowledge base, and R is the set composed by Data log^SHIQ^ rules, like:

\[
Q(X) = A_1(Y_1), A_2(Y_2), ..., A_n(Y_n) \\
& B_1(Z_1), B_2(Z_2), ..., B_m(Z_m)
\]  

(2)

is called the Data log^SHIQ^ mode. Q, A_1, A_2, ..., A_n, is the random meta-predicate of rules, B_1, B_2, ..., B_m is the concept(one-predicate) or role(two-predicate) of description rules. Q(X) and A_i(Y_i) is the Data log atom, while B_i(Z_i) is DL atom (1 ≤ i ≤ n, 1 ≤ j ≤ m).

However, due to the different mathematical foundations of description logic and rule logic, there would be undesirable halting problems of Turing Engine with combination of ontology and rules if there is no secure constriction. It would stop the reasoning. To avoid this case, we should define the three securities as follows:

**Definition 5:**

Data log Security: [X] ⊑ U_{i=1}^m [Y_i] \cup U_{j=1}^n [Z_j]  

(3)  

DL + log Security: [X] ⊑ U_{i=1}^m [Y_i] \cup U_{j=1}^n [Z_j]  

(4)  

DL Security: [X] ⊑ U_{i=1}^m [Y_i] \cup U_{j=1}^n [Z_j]  

(5)

The [X] means sets of all the variables of sequence X.

By these secure constrictions, the reasoning can regain the decidability. Thus during the computing process of frequent pattern discovery, we would use support degree as an important reference.

**Definition 6:** The definition of support degree is:

\[
\text{support}(P, C_{ref}, B) = \frac{|\text{answerset}(P, C_{ref}, B)|}{|\text{answerset}(P, C_{ref}, B)|}
\]

(6)

\text{answerset}(P, C_{ref}, B) is the answer set of patter P, while the \text{answerset}(P, C_{ref}, B) is that of trivial pattern P_t.

The support degree is used not only to test the validation of parallel computing results but also in the expansion of Trie Tree. On the other hand, we also need the semantic equivalent pattern to judge whether there exists semantic redundancy.

Semantically Equivalent Pattern means that two patterns of the Web access pattern space share the same semantics which express the meaning of users’ accessing behavior. In general, when there exist semantic redundancies, there would be equivalence of two patterns. If there are two given Web access pattern H_1 and H_2, and the role atom of pattern H_1 could determine the events types of its codomain. Under this circumstance if we introduce the new access pattern H_2 consisted by this domain’s event atom, it would result in semantic redundancies of the two patterns. This process is called Semantically Redundant Pattern. The introduced atom is Semantically Redundant Atom. Based on the universal relationships of pattern semantics, by defining Semantically Free Pattern, it can avoid the presence of semantically redundant atoms of the new created patterns.

**Definition 7:** The definition of semantically free pattern is:

on the hybrid knowledge base Data log^SHIQ^ based on the DL + log, for given Web access patterns H, H’ is the pattern coming from H that deletes a random atom. If there is no such H’ that makes H → BH’, then H is semantically free pattern.

**III. ALGORITHM:**

**A. Making changes in the knowledge base**

In order to realize the parallel computing method, we change the traditional bidirectional reasoning process (2) to one-way reasoning process. Under this case, we simplify representation of rule header and only use the reasoning process starting from rule body to rule header. Thus the representation is like:

\[
Q(X) \leftarrow A_1(Y_1), A_2(Y_2), ..., A_n(Y_n) \\
& B_1(Z_1), B_2(Z_2), ..., B_m(Z_m)
\]

(7)

According to a large amount of researches, during the reasoning process of frequent pattern mining in this paper, we find that in majority cases, we use the reasoning process starting from rule body to rule header. Only in few cases, there exists the reasoning from rule header to rule body. Although, by doing this, the descriptive abilities would be weakened, we would realize the parallel computing. Thus, the data mining system can bear such sacrifices. Also, a large quantities of experiments have verified that the mining efficiencies would be increased in a large degree while there is no obvious signal of reducing reasoning qualities in the parallel computing in the multi-computers environment.

**B. MapReduce algorithm:**

After we define the disjunctive form Data log^SHIQ^ and the hybrid knowledge base Data log^SHIQ^, we can give the MapReduce parallel frequent pattern discovery algorithm based on ILP algorithm: firstly, we do case detection by using map function and then get the answer sets; secondly, we do parallel computing on each node to get the partial support degrees; finally, we use the reduce function to sum them. In this framework, the reasoning process is done by ILP engine, while the MapReduce algorithm realizes parallel calculations. The framework is shown in the “Fig.2”
The fundamental idea of the MapReduce parallel computing is to divide the observation sets, and the MapReduce Computing process would run with every iteration of the algorithm. The map function runs in its own observation section and then the function reduce would sum all the results of each function map. The detailed description is: firstly, the main process use MapReduce framework to divide the observation sets into N observation subsets whose size is the same. Secondly, the process distributes every k-candidate pattern depending on the (k-1)-candidate patterns to node to get result of function map. Then function reduce collects result of each node and gets separately support degrees of k-candidate sets. It computes the total support degrees of k-candidate patter and verify whether the pattern is frequent pattern or not. Finally, the main process determines whether to run the next step or not. The framework of function Map and Reduce is shown in “Fig.3”.

The framework of candidate pattern P is shown in “Fig.3-1”. Observation sets (O) are input, candidate pattern P is input, and the knowledge base B is input. The output is support degree S. The input of the map function is observation set O and the output is support degree S.' 

The MapReduce algorithm is represented as the follows:

Input: candidate pattern P, observation set fragmentation O, data except observation set of hybrid knowledge base

foreach cᵢ ∈ Cref
    If there exits oᵢ, that is a case detection of cᵢ and Pᵢ then
        Put the cᵢ to answer set(Pᵢ, Cref, B)
    According to formula (6), we can compute the partial support degree sᵢ of Pᵢ.

In this algorithm, cᵢ is an individual of core concept Cref, oᵢ is an observation of observation set fragmentation.

If cᵢ and Pᵢ logics cover oᵢ, then cᵢ is correct answer to Pᵢ. The answer set(Pᵢ, Cref, B) is the set that consists of all the cᵢ based on hybrid knowledge base B.

After getting all the partial support degrees, we use the function reduce to sum up to get global support degrees to select the frequent answer sets. Then we use the verified parallel computing results to build up Trie Tree via ExpandNode algorithm.

C. ExpandNode algorithm:

During the process of expanding pattern, the Trie Tree ExpandNode algorithm is to expand the next level frequent pattern set via broad-first method, according to the single pattern or the last level frequent pattern sets. The algorithm would stop when there is no frequent pattern in a level or the Trie Tree has reached the highest level.

In this algorithm, the first key algorithm is to get the candidate pattern. The traditional method of this procedure is to use a concept or rule atom to connect to a pattern of the (k-1)-frequent pattern sets, which is the last result, and all the k-candidate pattern consists of the k-candidate pattern sets. Although this method is very simple and complete, there are too many candidate patterns which would contain a large amount of patterns that have the same semantics that is useless. That would cause a lower efficiency of the system. Thus, in this paper, we would get candidate pattern by expanding atom and at the same time compute support degrees and verify the validities rather than get candidate pattern sets directly. Only when we find the pattern is effective and frequent, we would add it to the Trie Tree. In detail, the method to expand nodes is determined by current node and right sibling node.

The Frequent Pattern Tree can be represented by a tree which begins with key concept Cref. Thus a pattern is the way from root node to random child node. The algorithm is:

Input: parent node to be extended N(x,yᵣ), node level L

If L < maximum depth then
    compute available predicate set APN of N(x,yᵣ)
    foreach A ∈ APN do
        construct corresponding node of A, A(x,yᵣ)
        construct pattern Pᵢ which is from root node to A(x,yᵣ)
        if Pᵢ is not a semantic equivalence pattern and pass verification of redundant schism then
            number of nodes to compute the partial support degrees
            get all the map results according to pattern Pᵢ as well as the data fragment of each nodes
            use reduce function to compute the support degree s of pattern Pᵢ
            if s > minsup then
                add A(x,yᵣ) to N(x,yᵣ) as child node
            foreach B(x,yᵣ) the child node of N(x,yᵣ) do
                ExpandNode (B(x,yᵣ), nodeLevel+1)
        output: frequent pattern tree FPT
IV. EXPERIMENT RESULTS:

In this paper, the Hadoop Framework is used, and it is the most representative parallel computing framework of master-slave mode. For this framework, Master is responsible for task assignments, while Slave accomplishes the task. The configuration parameters of the nodes are shown in “Fig.4”:

![Figure 4: Configuration parameters of the experiment](image)

The observation sets of the experiment come from the financial data of http://www.owl-ontologies.com/. And the result of the serial algorithm mining based on the 128k observation sets is shown at “Fig.5”:

<table>
<thead>
<tr>
<th>Level</th>
<th>The number of candidate modes</th>
<th>The number of frequent modes</th>
<th>Running time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.12</td>
</tr>
<tr>
<td>2</td>
<td>63</td>
<td>5</td>
<td>2.85</td>
</tr>
<tr>
<td>3</td>
<td>406</td>
<td>66</td>
<td>18.12</td>
</tr>
<tr>
<td>4</td>
<td>1782</td>
<td>389</td>
<td>788.31</td>
</tr>
<tr>
<td>5</td>
<td>9920</td>
<td>2351</td>
<td>875.02</td>
</tr>
</tbody>
</table>

![Figure 5: The result of serial algorithm mining basing on 128k observation sets](image)

The result of the serial algorithm mining based on the 1M observation sets is shown in “Fig.6”:

<table>
<thead>
<tr>
<th>Level</th>
<th>The number of candidate pattern</th>
<th>The number of frequent pattern</th>
<th>Running time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.17</td>
</tr>
<tr>
<td>2</td>
<td>71</td>
<td>7</td>
<td>7.25</td>
</tr>
<tr>
<td>3</td>
<td>426</td>
<td>67</td>
<td>63.98</td>
</tr>
<tr>
<td>4</td>
<td>3083</td>
<td>427</td>
<td>787.11</td>
</tr>
<tr>
<td>5</td>
<td>19794</td>
<td>3013</td>
<td>222911.14</td>
</tr>
</tbody>
</table>

![Figure 6: The result of serial algorithm mining basing on 1M observation sets](image)

As the results shown, the running time is increasing exponentially with the rise of level. There are two reasons for this phenomenon: firstly, the atoms of the pattern increase with rising of the level; secondly, the range of reasoning and the number of frequent pattern increases with the rising number of observation sets. When we use the parallel computing algorithm, the changes of the speed ratio with the change of level represent in the “Fig.7”:

![Figure 7: The running time of the serial algorithm basing on different scale of observation sets](image)

In this experiment, we also get results of the speed ratio according to the max number of level when the number of the mapper is 4 or 12. The results show in the “Fig.8”:

![Figure 8-1: The speed ratio of the 4 mappers](image)
![Figure 8-2: The speed ratio of the 12 mappers](image)

When the max level is five, we can get the following results of speed ratio according to the different number of map in the “Fig.9”:

![Figure 9: The speed ratio of the different number of maps](image)
If the maximum level is too small, such as the level is 3, then the speed ratio would less than 1. If the observation set is too small, such as the observation set is only 128k, and then the speed ratio would not increase, although we add more processors.

If the number of processors and the maximum level were fixed, then with the observation set increases, the speed ratio would grow with near-linear efficiency in a certain degree. When the observation grows into a certain scale, the speed ratio would be stable.

If the maximum number of level and the scale (larger than 128K) of observation sets were fixed, the speed ratio would increase in a linear degree with the rise of number of the processors.

The analysis are as follows based on the conclusions:

1. Firstly, we can get the time of parallel process \( t_p(P, M) \) by using the M mappers from P processors. The \( r_1, r_2, ..., r_M \) represent processing time of each mapper that is from M mappers. Given that \( r_i \) is not a descending sequence (which means that \( r_M = \max(r_1, r_2, ..., r_M) \)) and the consistent and serial processing time \( t_s \) is not affected by single or multi processors, then according to \( t_p(P, M) = r_M \), the speed ratio is:

   \[
   S = \frac{t_s + \sum_{i=1}^{M} r_i}{t_s + t_p(P, M)} = \frac{t_s + \sum_{i=1}^{M} r_i}{t_s + r_M}
   \]

   If \( t_s \ll t_M \), the above formula can be simplified as:

   \[
   S = 1 + \frac{\sum_{i=1}^{M-1} r_i}{r_M}
   \]

   Since \( \sum_{i=1}^{M-1} r_i \leq (M - 1)r_M \), when the values of \( r_1, r_2, ..., r_M \) are much the same, we can get the maximum ratio speed M. It will increase linealy along with the rise of the number of mapper. In contrast, if some processors last for longer time without increasing in certain ratio, such as \( r_M = 2r_{M-1} \), then \( S < M/2 \). Consequently, the following cases are reasonable:

   1. The time of function reduce and other serial operating can be neglected. \( t_s \ll t_p(P, M) \)
   2. The time of function map is much the same. \( t_p(P, M) \) would not be affected by single value.

Thus, based on the improved parallel computing algorithm of this paper, the pattern discovery efficiency is higher than before.

V. CONCLUSION:

The paper uses the MapReduce Framework based on the ILP algorithm to get parallel computing results and verify whether the result is frequent and effective simultaneously. Then the resulting data help establish Trie Tree. Finally, the paper outlines how we get frequent pattern discovery with a higher speed. With the improvement of both qualities and efficiencies, the algorithm of this paper is very meaningful in many research fields as well as business applications.

REFERENCE: