High Utility Patterns: Algorithms, Generalizations And A Graph Theoretic Approach

1st Farhad Mirzapour
Department of Computer Science
University of Manitoba
Winnipeg, MB, Canada
mirzapom@myumanitoba.ca

2nd Supervisor: Dr. Carson K. Leung
Department of Computer Science
University of Manitoba
Winnipeg, MB, Canada
kleung@cs.umanitoba.ca

***Abstract*—In this paper we provide a review of the well-studied problem of high frequency pattern mining and its most canonical generalization, the problem of high utility pattern mining. The classical formulation of a high frequency pattern captures a notion of importance of a pattern but also fails to encapsulate the significance of patterns in more complex systems. This has led to the study of high utility patterns which capture data about the external and internal utility of items present in a transaction and not just the binary state of existence. This inclusion of the notion of weights in the study of high utility patterns, turns out to be rather useful when dealing with situations that warrant them. We study some proposed algorithms for the problem of high frequency pattern mining and the problem of high utility pattern mining. We then examine some rather interesting generalizations of the formulation of the problem of high utility pattern mining that add new complexity to our study and also examine a proposed framework of mining high utility subgraphs from a graph data structure which views the problem from a different perspective.**

***Index Terms*—High Frequency Pattern Mining: High Utility Pattern Mining, Graph Mining**

I. Introduction

A. High Frequency Pattern Mining

We start by defining the problem of high frequency pattern mining which is a baseline of all pattern mining algorithms. We define a total list of items $I = \{i_0, i_2, ..i_n\}$ as any finite set, elements of which we call **Items**. A **Transaction** $T_{ID} \subseteq I$ is defined as any subset of the list of items and is also assigned a unique identifier ID. A **Transactional Database** $D = \{T_0, T_1, ..T_m\}$ is defined as a finite set of such transactions. We define an **Itemset** $X \subseteq I$ as any subset of our total list of items. **The Support Measure** of any itemset X in transactional database D is defined as the number of transactions in D that contain X as a subset. Formally $sup(X) = |\{T \in D | X \subseteq T\}|$. **A Frequent Itemset** is any itemset X whose support measure $sup(X) >= MINSUP$ where $MINSUP$ is an arbitrarily defined support threshold. Now we are ready to define the problem of **High Frequency Pattern Mining** as the task of finding all frequent itemsets given some $MINSUP$ threshold. The following example lets us see all our definitions in action:
Let $I = \{a, b, c, d, e\}$ be our list of 5 items. The following table shows 5 transactions in a transactional database.

transaction identifier	transaction
0	$\{a, b, c, d, e\}$
1	$\{b, c, d, e\}$
2	$\{c, d, e\}$
3	$\{d, e\}$
4	$\{e\}$

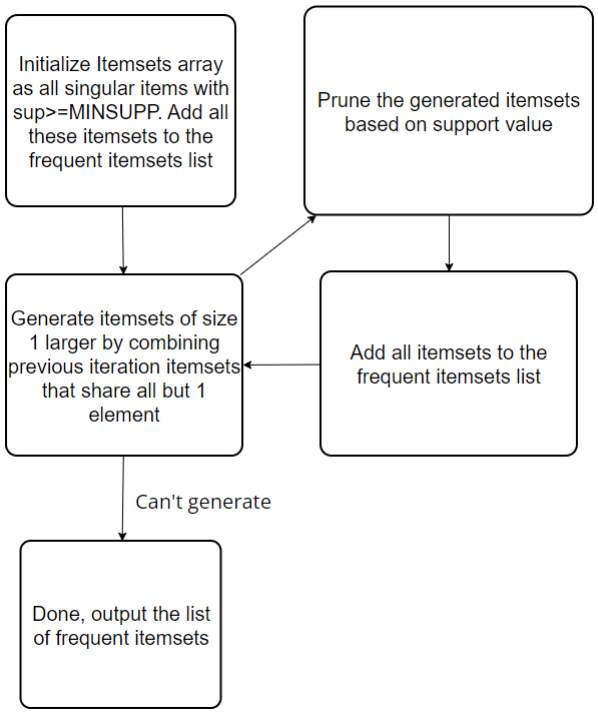
The following table shows all itemsets and their support values with respect to the transactional database defined above. Note that the number of possible itemsets is $2^{|I|} = 2^5 = 32$. We usually do not consider the empty set as an itemset yielding 31 itemsets.

support	itemset	support	itemset
2	$\{b\}$	1	$\{a, b\}$
3	$\{c\}$	0	$\{a, c\}$
2	$\{b, c\}$	1	$\{a, b, c\}$
4	$\{d\}$	0	$\{a, d\}$
0	$\{b, d\}$	0	$\{a, b, d\}$
3	$\{c, d\}$	0	$\{a, c, d\}$
2	$\{b, c, d\}$	1	$\{a, b, c, d\}$
5	$\{e\}$	0	$\{a, e\}$
0	$\{b, e\}$	0	$\{a, b, e\}$
0	$\{c, e\}$	0	$\{a, c, e\}$
0	$\{b, c, e\}$	0	$\{a, b, c, e\}$
4	$\{d, e\}$	0	$\{a, d, e\}$
0	$\{b, d, e\}$	0	$\{a, b, d, e\}$
3	$\{c, d, e\}$	0	$\{a, c, d, e\}$
2	$\{b, c, d, e\}$	1	$\{a, b, c, d, e\}$
1	$\{a\}$		

We can now see that if we are given $MINSUP = 0$, then every itemset listed (all 31) are considered frequent, but this is not very interesting. If we consider $MINSUP = 2$ We will find that only the following itemsets are frequent :

support	frequent itemset
2	$\{b\}$
3	$\{c\}$
2	$\{b, c\}$
4	$\{d\}$
3	$\{c, d\}$
2	$\{b, c, d\}$
5	$\{e\}$
4	$\{d, e\}$
3	$\{c, d, e\}$
2	$\{b, c, d, e\}$

We observe that increasing the $MINSUP$ value results in less and less frequent itemsets which is apparent from the definitions. Now that we have seen what a list of frequent itemsets might look like, it is natural to go back to the problem of finding all such frequent itemsets. Certainly one could generate all possible itemsets and then scan through the database and check if each one of the generated itemsets are frequent. This approach is referred to as the **Brute Force** approach and while it works and solves the problem, it is very inefficient as it involves scanning through the database for every single possible itemset which is at least $O(2^{|I|})$ and exponential with respect to the number of items which makes this approach basically infeasible when we are dealing with hundreds of items. The heart of many algorithms that aim to solve the this inefficiency problem is the **Anti-Monotonicity** of the support measure. This means that for any itemsets $X_1, X_2 \in I$ where $X_1 \subseteq X_2$ We have $sup(X_1) > sup(X_2)$. This is very intuitive to see since any transaction that contains X_2 certainly also contains X_1. Now that we have established this property, we can use it to prune the search space as we look for frequent itemsets. This solves our inefficiency problem since if we take a bottom up approach for generation of the itemsets (start with an empty set and at each step generate itemsets with size 1 more than the previous step), we can stop looking at all supersets of any infrequent itemset. This greatly improves the efficiency of our algorithm. This general approach called the Apriori algorithm turns out to be efficient for solving the problem of high frequency pattern mining [1]. The following is a flowchart of the Apriori algorithm:



The Apriori algorithm has a time complexity of $O(n \times m^2)$ where $n = |D|$ is the number of transactions in D and $m = |I|$ is the number of items [3].

B. High Utility Pattern Mining

Often times a transaction in a database contains more useful information than just the list of items present in a transaction. A general store might be more interested in transactions containing computers than transactions containing gums since computers yield more profit than gums. The number of copies of an item in a transaction might also be of interest. For example a transaction containing 10 bars of candy might be more profitable than a transaction containing 3 apples. Such examples of transactions in the real world have led the study to include the notion of **Internal** and **External** utility for items in a **Quantitative Transactional Database**. We alter our definitions to fit our needs the following way: Let each item $i \in I$ have an external utility $p(i)$ associated to it. Also let each transaction $T_{ID} \in D$ assign an internal utility $q(i, T_{ID})$ to every item inside of it. A transactional database whose transactions are defined as above is called a quantitative transactional database. We define the utility of item in a transaction $i \in T_{ID}$ as the product of its internal and external utilities $u(i, T_{ID}) = p(i) \times q(i, T_{ID})$. The utility of an itemset X in a transaction T_{ID} is defined as the sum of the utilities of the items inside of it with respect to T_{ID} if X is contained inside of T_{ID}, $u(X, T_{ID}) = \sum_{i \in X} u(i, T_{ID})$, and 0 otherwise $u(X, T_{ID}) = 0$. The **utility measure** of an itemset X in a quantitative transactional database D is then defined to be the sum of the utility of X with respect to every transaction in D, $u(X) = \sum_{T_{ID} \in D} u(X, T_{ID})$. In a sense this notion of utility

measure aims to capture the total importance/profit generated by an itemset in the database. Any itemset X whose utility in database D, $u(X) >= MINUTIL$ where $MINUTIL$ is an arbitrarily user defined threshold is considered **High Utility**. We are now able to define the problem of **High Utility Itemset Mining** as the task of finding all high utility itemsets given a quantitative transactional database D. This quantitative transactional database with 5 transactions highlights the difference between a utility situation that allows for internal and external utilities and the frequency situation that we looked at before.

transaction identifier	transaction
0	$\{(a,1),(b,2),(c,3),(d,3),(e,4)\}$
1	$\{(b,2),(c,3),(d,1),(e,1)\}$
2	$\{(c,2),(d,1),(e,3)\}$
3	$\{(d,3),(e,2)\}$
4	$\{(e,2)\}$

Note that each item is mapped to its internal utility in each transaction (internal utility of each item is the second parameter of the tuple). The following table provides external utilities for all items in our database.

item	external utility
a	1
b	2
c	3
d	4
e	5

Imagine we are given $MINUTIL = 30$. We can calculate the utility measure of itemsets the following way:
$u(\{a,b,c,d,e\}) = (1\times1)+(2\times2)+(3\times3)+(3\times4)+(4\times5) = 46 \geq 30$.
We find that $\{a,b,c,d,e\}$ is high utility.
$u(\{c,d,e\}) = (2\times3)+(1\times4)+(3\times5)+(3\times3)+(1\times4)+(1\times5)+(3\times3)+(3\times4)+(4\times5) = 84 \geq 30$.
We find that $\{c,d,e\}$ is high utility.
$u(a,b) = (1\times1)+(2\times2) = 4 \leq 30$ so $\{a,b\}$ is not high utility. It is also important to note that any algorithm capable of finding all high utility patterns is also capable of finding all high frequency patterns by setting all external weights equal to 1 and all internal weight of items that are present in a transaction to 1. For example, finding all high utility itemsets with respect to $MINUTIL$ in the following table is equivalent to finding all high frequency itemsets with respect to $MINSUP = MINUTIL$ in the example provided in section A.

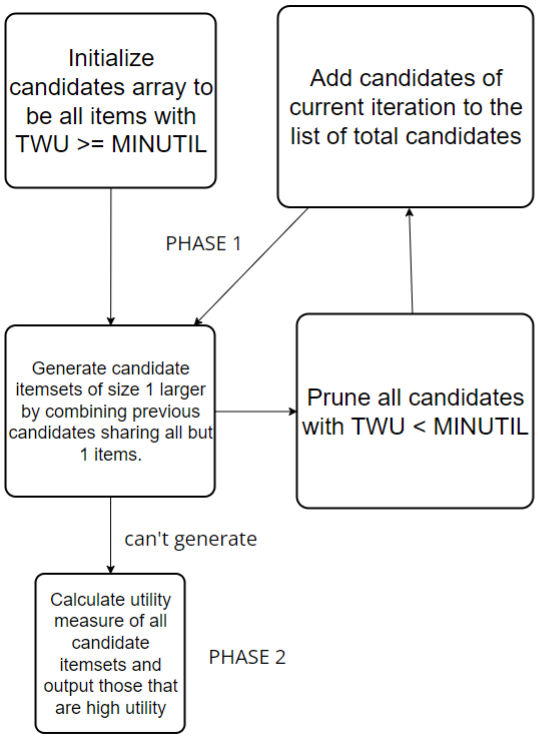
transaction identifier	transaction
0	$\{(a,1),(b,1),(c,1),(d,1),(e,1)\}$
1	$\{(b,1),(c,1),(d,1),(e,1)\}$
2	$\{(c,1),(d,1),(e,1)\}$
3	$\{(d,1),(e,1)\}$
4	$\{(e,1)\}$

With the following the external utilities :

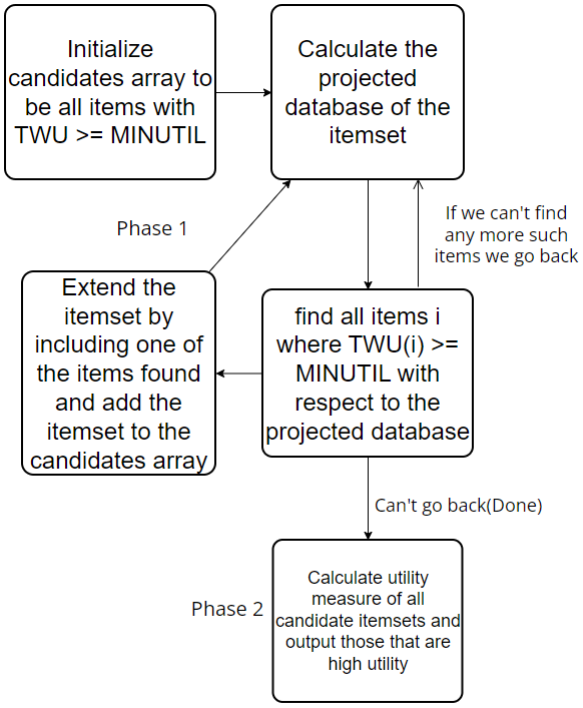
item	external utility
a	1
b	1
c	1
d	1
e	1

Any high utility itemset mining algorithm could be used to find all high frequency patterns but this is often not the most efficient approach as algorithms that are developed solely to find high frequency itemsets, capitalize on the simplicity of the situation and are more efficient. This is one of the reason why we can't generalize them to solve for high utility itemsets as we will discuss later. In order to come up with an algorithm that finds all high utility itemsets given a database D, we can again start with the brute force approach. Generate all possible itemsets, calculate the utility of each itemset by scanning the database, output the itemsets whose utility is above a given $MINUTIL$ value. This brute force approach is again correct but very inefficient and costly. In order to find an optimization, given all the similarity that we have established between the problem of high utility pattern mining and the problem of high frequency pattern mining, it is very tempting to try to prune the search space as we did before (Apriori algorithm). This time using the utility measure instead of the support measure. This turns out to be incorrect since the utility measure lacks the key property of anti-monotonicity that the support measure possessed. To see this we can look at the example we have provided and the two itemsets $\{a\} \subseteq \{a,e\}$. $u(\{a\}) = (1\times1) = 1$ but $u(\{a,e\}) = (1\times1)+(4\times5) = 21$ so $u(\{a\}) \leq u(\{a,e\})$ which means the utility measure is not anti-monotone. This means that we can not use the utility measure to prune the search space. In order to keep the heart of the technique we try to come up with other measures that are anti-monotone and are upper bounds on the utility measure. It is easy to see that any measure that is anti-monotone and is an upper bound on the utility measure can be used to prune the search space. The new problem that this approach introduces is the fact that after pruning the search space using said measure, the remaining itemsets are not necessarily high utility since the said measure is only an upper bound on the utility measure. This means that we still need to go through the **candidate** itemsets that survive the pruning phase and calculate the exact utility of each one to see if they are actually high utility with respect to the specified $MINUTIL$ value. It turns out that this line of thinking is very useful and is exactly what we need. The first branch of algorithms that were invented to solve the problem of high utility itemset mining, are called **Two Phase Algorithms** and do exactly what we discussed. One of the canonical measures that experts have come up with is called the **Transaction-Weighted Utilization(TWU) Measure** and is defined to be the sum of transaction utilities for transaction than contain an itemset X. Formally $TWU(X) = \sum_{T_{ID}\in D, X\in T_{ID}} u(T_{ID}, T_{ID})$.

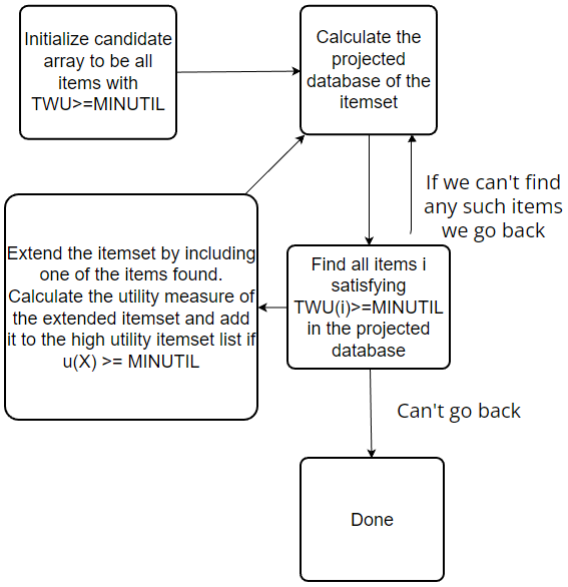
It is easy prove that the TWU measure is an upper bound on the utility measure but is also anti-monotone which allows us to prune the search space. Most two phase algorithms have the same structure. In phase 1, start from itemsets of size 1 (containing a single item). At step k, the algorithm combines pairs of itemsets that share all items but one (k-1 items are shared). The TWU measure of these generated itemsets are then calculated and if $TWU(X) \leq MINUTIL$ for any generated itemset X, X is pruned. The remaining itemsets (that were not pruned) are added to a candidate list. The algorithm then moves on to the next step and generates Itemsets of size k+1. If at any step we find that we can no longer generate itemsets of bigger size, we finish phase 1. Phase 2 is simple, the utility measure of each candidate itemset is calculated and the high utility itemsets are found. The following is a flowchart of the described two phase algorithm.



The proposed two phase algorithm is indeed a step in the right direction. It correctly finds all high utility itemsets with respect to a quantitative transactional database D and a $MINUTIL$ threshold and is way more efficient than a brute force approach. There are still several major problems with the two phase algorithm. The first problem is that although TWU pruning the search space is better than a brute force approach, the TWU measure turns out to be a not so tight upper bound on the utility measure. This means that the two phase algorithm generates way too many candidate itemsets that are not high utility and have to be eliminated in the second phase of the algorithm. In the worst case, all itemsets could be chosen as candidates which yields exponential runtime. The second big problem with the two phase algorithm is the fact that it generates the itemsets using a BFS (breadth first search) approach which means that at the k+1 iteration, it keeps all generated itemsets of length k in the memory. This is very costly in terms of space complexity. The two phase algorithm also generates the candidate itemsets without utilization of the database which means that there is a high chance that many itemsets that are not present in the database are generated [1]. Various papers have proposed a number of solutions that tackle some of these problems. The first big improvement is the idea of **Pattern Growth Two Phase Algorithms**. There are many variations of pattern growth algorithms (IHUP, MU-Growth, UP-Growth,etc) but they all utilize the same core concept. Pattern growth algorithms introduce the idea of a **Projected Database**. The core idea is to eliminate generation of itemsets that are not present in the database and also avoid scanning through the entire database while calculating the TWU measure of each itemset. This is achieved by fundamentally altering the way we generate candidate itemsets. The candidate itemsets are generated using a BFS approach in the Apriori algorithm and the two phase algorithm, but here we utilize a total ordering on the items (e.g. lexicographic) and generate the itemsets using a DFS (depth first search) approach. The candidate generation algorithm works the following way: We start with all itemsets consisting of all singular items that have $TWU >= MINUTIL$. At each depth of the candidate generation algorithm, we pick an itemset from the current iteration with respect to some total ordering. We then try to extend that itemset by looking at the projected database of that itemset. The projected database of an itemset is the list of all transaction from the quantitative database D that contain the itemset where all the items that come before elements of the itemset according to the total order are removed. The TWU measure of all the items appearing in the projected database are then calculated and if the TWU of an item is higher than MINUTIL, it is added to the itemset, the itemset is added to the list of candidate itemsets, and we move to the next depth. Projected database at each depth is calculated using the projected database at the previous depth. If we find that we can not extend the current itemset, we backtrack to a previous depth. We continue the DFS approach until all candidate itemsets have been found. Phase 2 of the algorithm is then conducted the exact same way as before. We calculate the utility measure of all candidates and see which candidates are high utility. Note that pattern growth algorithms also consist of two distinct phases but are usually discussed separately than the two phase algorithm. A flowchart of a typical pattern growth algorithm is provided:



Generating candidate itemsets in this way turns out to be a huge improvement to the BFS approach. UP-Growth was found to be up to 1000 times faster than the two phase algorithm [4]. It is important to note that we still have to store the projected databases in memory. These duplicates of the database take up alot of space and there has been much work done to optimize this approach even more in terms of space utilization. Space complexity can be reduced even more by constructing the projected database using pointers to the original database [5]. This way we don't have to store duplicates of the database in the memory. There are also other memory optimization that can be performed such as using prefix-tree data structures to store projected databases. Such a prefix-tree structure was used in the UP-Growth algorithm discussed before [4]. A big problem with pattern growth two phase algorithms and the two phase algorithm is the need to store all candidate itemsets in memory in the first phase, until they are sorted out in the second phase of the algorithm. It is natural to think of a way to merge the two phases and eliminate this need of storing all candidates in memory. This exactly the idea behind **One Phase Algorithms**. One phase algorithms have the same structure as pattern growth two phase algorithms except for the fact that they calculate the utility measure of all candidate itemsets as they are discovered and immediately decide whether or not they are high utility on the go. It is important to note that the candidate itemsets are still generated and pruned based on the upper bound, anti-monotone measure (TWU or variations of it). This means that if an itemset X is found to be not high utility but it satisfies the $TWU(X) >= MINUTIL$ property, its extensions/supersets are still considered in the next depth/step of the algorithm. This approach does indeed circumvent the need of storing candidate itemsets in memory. The following is a flow chart of a one phase algorithm.



A large class of one phase algorithms utilize a structure known as a **Utility List**, a data structure that stores information about the utility of an itemset and its remaining utility (utility of extensions of the itemset with respect to an ordering) which is used to calculate a new anti-monotone upper bound measure called **Remaining Utility Upper Bound** which is supposedly a tighter upper bound than the TWU measure and more efficient when used to prune the search space. The FHM algorithm is one such example [6]. There are other one phase algorithms that offer slight variations in terms of the measure used to prune the candidates and how the candidates are generated, but they all use the main ideas we have discussed.

II. Genralizations

A. Motivation

The main goal of our study is to find important patterns in a database. One of the biggest problems that we face is that this notion of importance is not necessarily shared by everyone. In the real world, different transactional databases are designed to hold different information about different things and this introduces lots of complexity in our setting. The inclusion of internal and external utilities in a transactional database turns out to be a great way of generalizing frequent itemsets. This is because, as we discussed before, high frequency itemsets are not necessarily the most important when we look at the full context of a situation (e.g. a general store). But there are also many situations where the classical formulation of a high utility itemset fails to encapsulate the measure of significance of an itemset, that is required in the specific situation. This had led to many generalizations/extensions of the problem of finding high utility itemsets which examine practical situations whose representation requires a modified structure. In this pa-

per, we examine some of the more interesting generalizations and the big ideas behind some solutions.

B. Top k High Utility Itemset Mining

The formulation of the problem of high utility itemset mining relies heavily on the MINUTIL threshold specified by the user. This may pose a problem since the user might not know what MINUTIL value to set to get a moderate list of high utility itemsets. Setting the MINUTIL too low (close to 0) will result in too many high utility itemsets that may not be of use and also slows down the algorithm significantly since more candidate itemsets will be considered. Setting the MINUTIL too high will result in too few high utility patterns. If the user does not have a rough idea of what MINUTIL threshold will achieve their desired effects, the high utility itemset mining algorithms that we have discussed may not be useful. The problem of **Top k High Utility Itemset Mining** is introduced to address this issue. Instead of specifying a MINUTIL threshold, the user specifies a positive integer k and the task is to find the top k high utility itemsets with the largest utility. Most algorithms that aim to solve the problem of top k high utility itemset mining, utilize algorithms for the problem of high utility itemset mining the following way. The MINUTIL threshold is started at 0 until k high utility itemsets are found. The MINUTIL is then set to the lowest utility among the k itemsets found. The search space is explored as the list of top k high utility itemsets are updated and the MINUTIL is changed to to be the lowest utility in the list at each step. An example is the TKU (mining Top-k Utility itemsets) [7] algorithm which is basically an extension of the UP-Growth algorithm.

C. High Utility Itemset Mining with Negative Utilities

Positivity of internal and external utilities in a quantitative transactional database turns out to be very central in the correctness of proposed algorithms. There are many real world application that might require the external utility of an item to be negative. As an example, many stores sell items with negative net profit in order to attract customers. The external utility of said items in a quantitative transnational database must be negative. The problem of **High Utility Itemset Mining with Negative Utilities** is described as this exact situation. What would happen if we allow the external utilities to take on negative values? It turns out that many measures that we discussed before that were used for pruning, are no longer upper bounds on the utility of an itemset (e.g. TWU). The solution that experts have come up with is using more clever upper bound measures to prune the search space. One such measure is the **Upper Bound on Utility Using Positive Items** $u(up(X))$. $up(X) \subseteq X$ is defined to be the set of all items in X with positive external utility and $un(X) \subseteq X$ is defined to be the set of all items in X with negative external utility. The FHN [8] algorithm utilizes $u(up(X))$ to prune the search space. Any itemset that only has extensions (with respect to some total ordering) that include negative items and also has $u(up(X)) \leq MINUTIL$ is pruned. There are other algorithms with other novel modified upper bounds that aim to extend various algorithms used for the problem of high utility itemset mining.

D. Mining High Utility Itemsets that Are Correlated

It is important to note that items in a high utility itemset are not always correlated. Suppose we have an item $i \in I$ where the external utility of i, $p(i)$ is very large. It is clear that any itemset containing i will be deemed high utility by the traditional algorithms. This might pose a problem because if high utility itemsets are used to, for example, market items together, we also want the items in a high utility itemset to be correlated. For example, we don't want to market gums with computers just because the itemset $\{computer, gum\}$ is high utility (result of the high price of computers). In order to solve this problem we would like to find all high utility itemsets that are also correlated. There are many measures of correlation that have been used to address this problem. One of the most recent and widely accepted measures is the **Bond Measure** [9]. The **Conjunctive Support** of an itemset X in a quantitative transactional database D is defined to be the number of transactions that contain X.
$conj_{supp} = |\{T \in D | X \in T\}|$. The **Disjunctive Support** of X is defined to be the number of transactions in D that contain any item of X. $disj_{supp} = |\{T \in D | \exists i \in I, i \in T\}|$. The bond measure of X is then define to be the ratio $bond(X) = \frac{conj_{supp}}{disj_{supp}}$. Now we are ready to define the problem of **Mining High Utility Itemsets that Are Correlated** as the task of finding all high utility itemsets X where the extra condition $bond(X) \geq MINBOND$ also holds. Algorithms (e.g. FCHM [9]) that solve this problem utilize the methods that we have studied but also prune using the bond measure since the bond measure is monotone. It turns out that the solutions to this generalization is faster than the solutions to the original high utility pattern mining problem since a large number of weakly correlated itemsets are pruned using the bond measure. There are other measures such as the **All-Confidence** measure that aim to capture correlation in an itemset but all algorithms that solve this proposed problem use the foundations that we have established.

III. Mining High Utility Subgraphs [2]

A. Motivation

The importance of internal and external weights were established from the usefulness and applicability of high utility patterns. It is natural to attempt to lay the foundations for the same generalization with respect to the problem of high frequency subgraph mining. There are many situations (road networks, social networks,etc) that warrant the use of a graph data structure. Mining high utility subgraphs may be of more importance than mining high frequency subgraphs when the utility of a subgraph represents the importance (revenue generated, traffic, etc) of that subgraph in any sort of network. This is enough motivation to lay the foundations for mining high utility subgraphs from a graph data structure.

B. Definitions

Let $L = \{l_0, l_1, ..l_n\}$ be any finite set, whose elements are called **Labels**. A **Quantitative Labeled Graph** $G = (V, E, l, p)$ is a tuple where V is a set of vertices. $E \subseteq [v]^2$ is a set of edges between those vertices. $l : E \cup V \rightarrow L$ is a function that assigns a label to every vertex and edge of G. $p : E \rightarrow R_+$ is a function assigning an internal utility to each edge of G. A **Quantitative Labeled Graph Database** is a set of such quantitative labeled graphs equipped with $q : L \times L \times L \rightarrow R_+$ that is, a function assigning external utility to each edge between two vertices. Any edge e between two vertices v_1 and v_2 in G has the internal utility $p(e)$ and the external utility $q(l(v_1), l(v_2), l(e))$. The utility of such an edge is then defined as the product $u_e(e) = p(e) \times q(l(v_1), l(v_2), l(e))$. The utility of a quantitative labeled subgraph $g = (V_g, E_g, l, p)$ is $u_g(g) = \sum_{e \in E_g} u_e(e)$. The utility of a labeled subgraph g in a quantitative labeled graph G is defined as:

$u_G(g, G) = \max_{g' \in \phi(g,G)} u_g(g')$ where $\phi(g, G)$ is the set of all quantitative labeled subgraphs of G that are isomorphic to g. Intuitively we are assigning the utility of g in G to be the highest utility of any "copy" of g in G. The utility of a labeled subgraph g in a quantitative labeled graph Database D is defined as: $u_D(g, D) = \sum_{G \in D} u_G(g, G)$. We are now ready to define the problem of **Mining High Utility Subgraphs** as the task of finding all labeled subgraphs g, such that $u_D(g, D) \geq MINUTIL$ where $MINUTIL$ is defined as: $MINUTIL = \sum_{G \in D} u_g(G) \times \delta$ for a given threshold δ.

C. Algorithm

The **GWU (graph Weighted Utility)** measure of a labeled subgraph g in a quantitative labeled graph G is defined to be $GWU(g, G) = u_g(G)$ if any copy(isomorphic graph) of g is a subgraph of G and 0 otherwise. The GWU measure of a labeled subgraph g in a quantitative labeled graph database D is $GWU(g, D) = \sum_{G \in D} GWU(g, G)$

It turns out that the GWU measure can be used to prune the search space. Now that we have a way to prune the search space, it is natural to think of a way to generate all candidate subgraphs and develop an algorithm that is similar to algorithms used for the problem of high utility itemset mining. This turns out to be exactly the approach and yields UGMINE [2]. One problem that a candidate generation algorithm will need to address is that of creating many isomorphic graphs. In order to avoid considering isomorphic subgraphs an inspired DFS code approach to representing subgraphs is used. This approach is taken from the gSpan algorithm that mines high frequency subgraphs [10]. In order to not consider and extend isomorphic graphs, only the canonical code (minimal with respect to some ordering on the labels) for a subgraph is considered. This turns out to be a complete framework for mining high utility subgraphs from a quantitative labeled graph database. Another measure called the **RMU(Right Most Utility)** is defined and used to prune the database and performance of GWU pruning and RMU pruning are compared. RMU pruning turns out to be faster in most cases [2]

IV. Conclusion

In conclusion, we have reviewed the baseline problem of high frequency pattern mining and its most important generalization which is the problem of high utility pattern mining. The consideration of external and internal utilities add quite a bit of complexity to the problem and some core ideas and algorithms that have been developed over the years to solve these problems were discussed. Even more genralizations of the problem of high utility pattern mining were touched on which introduce new complications. Some novel ideas that are elemental for developement of solutions to any such problems are considered. We also look at a theoretical foundation laid in [2] to address the problem from the viewpoint of graphs and look at an algorithm that finds high utility subgraphs in a quantitative labeled graph database.

Acknowledgment

This paper was written as a summary of my research done in COMP 4520 (Undergraduate Honours Project) class at the University of Manitoba. I would like to thank Dr. Leung for guiding me and introducing me to this interesting research area.

References

[1] Fournier-Viger, P., Chun-Wei Lin, J., Truong-Chi, T., Nkambou, R. (2019). A Survey of High Utility Itemset Mining. In: Fournier-Viger, P., Lin, JW., Nkambou, R., Vo, B., Tseng, V. (eds) High-Utility Pattern Mining. Studies in Big Data, vol 51. Springer, Cham.

[2] Md. Tanvir Alam, Amit Roy, Chowdhury Farhan Ahmed, Md. Ashraful Islam, Carson K. Leung: UGMINE: utility-based graph mining. Applied Intelligence 53: 49-68 (2023)

[3] Hegland, M.: The apriori algorithm–a tutorial. In: Mathematics and Computation in Imaging Science and Information Processing, vol. 11, pp. 209–62 (2005)

[4] Tseng, V.S., Shie, B.-E., Wu, C.-W., Yu., P.S.: Efficient algorithms for mining high utility itemsets from transactional databases. IEEE Trans. Knowl. Data Eng. 25(8), 1772–1786 (2013)

[5] Pei, J.,Han, J., Lu,H., Nishio, S.,Tang, S.,Yang, D.:H-mine: hyper-structure mining of frequent patterns in large databases. In: Proceedings of the 2001 IEEE International Conference Data Mining, pp. 441–448. IEEE (2001)

[6] Fournier-Viger, P., Wu, C.W., Zida, S., Tseng, V.S.: FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Proceedings of the 21st International Symposium Methodologies for Intelligent Systems, pp. 83–92. Springer (2014)

[7] Tseng,V.,Wu, C., Fournier-Viger, P.,Yu, P.S.: Efficient algorithms for mining top-k high utility itemsets. IEEE Trans. Knowl. Data Eng. 28(1), 54–67 (2016)

[8] Lin, J.C.-W., Fournier-Viger, P., Gan, W.: FHN: an efficient algorithm for mining high-utility itemsets with negative unit profits. Knowl. Based Syst. 111(1), 283–298 (2016)

[9] Fournier-Viger, P., Lin, J.C.-W., Dinh, T., Le, H.B.: Mining correlated high-utility itemsets using the bond measure. In: Proceedings of the International Conference Hybrid Artificial Intelligence Systems, pp. 53–65. Springer (2016)

[10] X. Yan and J. Han, "gSpan: graph-based substructure pattern mining," in 2002 IEEE International Conference on Data Mining, 2002. Proceedings., Dec 2002, pp. 721–724.