A Paradigm for Detecting Cycles in Large Data Sets via Fuzzy Mining

James P. Buckley and Jennifer Seitzer[*]
Computer Science Department
University of Dayton
300 College Park
Dayton, Ohio 45469-2160
{buckley, seitzer}@cps.udayton.edu

Abstract

Traditional data mining algorithms identify associations in data that are not explicit. Cycle mining algorithms identify meta-patterns of these associations depicting inferences forming chains of positive and negative rule dependencies. This paper describes a formal paradigm for cycle mining using fuzzy techniques. To handle cycle mining of large data sets, which are inherently noisy, we present the α-cycle and β-cycle, the underlying formalism of the paradigm. Specifically, we show how α-cycles, desirable cycles, can be reinforced such that complete positive cycles are created, and how β-cycles can be identified and weakened. To accomplish this, we introduce the concept of Ω nodes that employ an alterability quantification, as well as use standard rule and node weighting (with associated thresholds).

1. Introduction

1.1. Traditional data mining

With the present state of technology, we have the capability to store extremely large amounts of data in organized and automated systems. The preponderance of data warehouses and datamarts [3] [6] are concrete evidence that this is not only possible, but of great interest to researchers, government agencies, and large corporations. But what is the meaning and usefulness of these large repositories of data?

Most small to medium-sized databases can be understood by a single developer, and languages (SQL) and OLAP exist to provide a useful purpose for having such a storage. However, extremely large databases present a new set of problems: they cannot be understood by a single human and traditional querying techniques seldom produce enough useful information to justify maintaining such a large database. The latter is partly due to a lack of comprehension, but additionally they fail due to being cumbersome and tedious. Even with today's high-speed processing capabilities, the answer lies not in speeding up current query techniques, rather in a new and radical approach to finding answers and important associations in the data.

Data mining is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data [4]. We are no longer looking for tabular answers or aggregations of the data; rather, we are looking for *patterns* within the data that reveal knowledge previously unknown. One of the most common applications of data mining is to generate all significant association rules between items in a data set. We can employ efficient algorithms to mine a large collection of basket data type transactions for association rules between sets of items with some minimum specified confidence [1] [2] [10]. The data now has both meaning and usefulness.

1.2. Why meta-patterns are important

The patterns we discover in our data sets through data mining may not be necessarily isolated. There may be chains of rules forming patterns of patterns, or *meta-patterns*, where the head of one rule is the body of another rule. In particular, the chain of rules may form a *cycle*. This form of meta-pattern is the focus of our work in this paper.

[*] Partially supported under Grant 9806184 of the National Science Foundation.

Cycles exist commonly in everyday life. The changing seasons, human behavior patterns, and consumer purchasing patterns are all examples of cycles. As would be expected, the evidence of these cycles can be found in our data. It is important to identify such patterns because, as for data mining, this identification assists us in a better understanding of the data itself. And more importantly, because events comprising a cycle are interdependent, it allows us to focus on specific events that may perpetuate or break a cycle. Another powerful aspect of a cycle is its inherent implication of continuity. Extraction of a cyclic pattern alerts a system that targeting any of the cycle participant activities assures continuous attainment of the goal (at least until the cycle is broken). Likewise, an undesirable cycle can be broken by failing to fire any of the constituent rules.

2. Cycle Mining

In previous work, the authors developed a methodology for discovering cycles as well as a formal data set model [8]. We now present an improved methodology that uses the individual rule supports and confidences to detect and categorize different types of cycles. We also present an enhanced cycle detection algorithm that uses a metric τ computed from constituent rule support and confidence factors. This metric is used to characterize the strength of the encompassing cycle. A user-specified threshold, U, is utilized as a global variable to denote how strong a rule must be to be included in any potential cycle. This allows the user to select how sensitive the system will be and is dependent upon the particular enterprise being mined. The following definition was used and presented in the original paper.

Definition 2.1 (support and confidence) *The* support *for a rule, $C \leftarrow B$, is the percentage of data tuples that satisfy $C \wedge B$; the* confidence *for rule $C \leftarrow B$ is the percentage of tuples that satisfy C given all tuples that satisfy B* [7]. *The* support of a cycle *is the minimum support value of any of the constituent rules forming the cycle; the* confidence of a cycle *is the minimum confidence of any of the constituent rules forming the cycle.*

We define τ, the aggregation of the above two metrics, to be our threshold measurement for any specific rule. No rule with τ less than a user-specified threshold *U* will be considered meaningful enough to be placed in the system knowledge base. Hence, it will not be detected as part of any cycle.

Definition 2.2 (τ)

$$\tau = \begin{cases} (\text{support} + \text{confidence}) / 2, & \text{when support} \geq .5 \\ \text{MAX}(\text{support}, \text{confidence}), & \text{otherwise.} \end{cases}$$

Cycles are composed of *n* individual rules, so we define the strength metric τ applied to cycles as

$T = \min(\tau_1, \ldots, \tau_n)$, where τ_i is the strength measurement of rule *i.*

The revised methodology and cycle detection algorithm do not consider rules with τ less than user-specified threshold *U*. Thus, any detected cycle has *T* at least U.

2.1. Cycle mining methodology

Discovery of cycles has as much to do with the knowledge representation structure as it does with the rule discovery algorithm itself. We assume a hypergraph representation of the knowledge base where logical predicates appearing as head of rules are represented as vertices, and sets of (conjuncted) predicates, appearing as rule bodies, are represented as hyperedges. For example, a rule $P(X) \leftarrow R(X), \ldots, Q(Y)$ with head $P(X)$ and body $R(X), \ldots, Q(Y)$ is internally represented as vertex $P(X)$ with incoming hyperedge $R(X), \ldots, Q(Y)$.

Our methodology consists of four steps:

1. Discover constituent rules in the form of *if-then* patterns
2. Insert discovered rules, the supports of which are greater than threshold ***U***, into the program hypergraph.
3. Run Cycle Mining Algorithm 2.1 (below)
4. Determine the new state of the knowledge base. As the dependency graph grows with each newly discovered pattern, the current state of the combined collection is reevaluated.

Algorithm 2.1 (Cycle Mining)

```
Input:   -Hypergraph representation P,
         -discovered rule $h \leftarrow b$ with $\tau$ value
         -acceptable strength threshold U
Output:  -Set of nodes forming a cycle C
              where C has $T \geq U$
         -Precise T of newly found cycle C
BEGIN ALGORITHM 2.1
If $\tau$ of rule $(h \leftarrow b) < U$ then
    discard rule
else
```

```
insert rule into P
Perform a postorder numbering (via
    simple dfs) on the program hypergraph
    with head h as starting node
If postorder number of head is ≥ any of
     of its incoming neighbor postorder
     numbers, then there exists a causal cycle
     formed by the edge emanating from the
     incoming neighbor to the head

     Traverse the (cyclic) path C backwards --
       node to incoming neighbor -- to h,
       tracking the minimum $\tau$
            $T = \min(\tau_1, \ldots, \tau_n)$
            Return C and T
       END ALGORITHM 2.1
```

2.2. A brief example

Assume the following rules reside in a dependency hypergraph indicating causal relationships within a system. The user specified acceptability threshold is set to 0.85.

costly_auto_repair $\rightarrow$ *low_funds*
low_funds $\rightarrow$ *car_not_maintained*
car_not_maintained $\rightarrow$ *dirty_engine*

with τ values of 0.92, 0.88, and 0.97, respectively. A new pattern with τ value of .86 is discovered.

dirty_engine $\rightarrow$ *costly_auto_repair*

The Cycle Mining Algorithm detects the following cycle with T = 0.86, as shown in Figure 1.

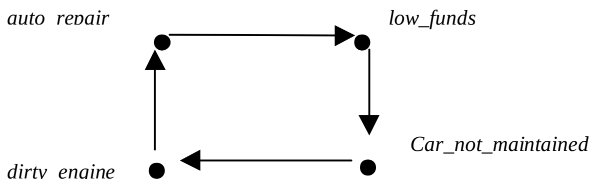

Figure 1. Example cycle

3. Fuzzy Cycle Paradigm

3.1. α-cycles and β-cycles

Our Cycle Mining algorithm is able to detect all cycles in a particular data set that meet a specific confidence and support threshold. However, cycles differ to the extent that they meet external system goals or semantic domain criteria.

Definition 3.1 (Cycle Types)

A complete cycle *is any cycle where* $T \geq U$; *(where U is a user-specified value). A* partial cycle *is any cycle where* $0 < T < U$. *An* α-cycle *is a complete cycle that results in a positive goal. A* partial α-cycle *is a partial cycle that, if complete, would result in an* α-cycle. *A* β-cycle *is a complete cycle that contains a proposition that is counter to one or more external system goals. A* partial β-cycle *is a partial cycle that, if complete, would result in a β-cycle.*

As indicated above, both α-cycles and β-cycles are determined semantically. That is, at this point in our work, there is no automated method to classify cycles as desirable (α-cycles) or undesirable (β-cycles). These assessments are made externally by humans of the enterprise owning the computer system. The assessments are useful to the enterprise because they indicate whether the cycle should be perpetuated as in the case of α-cycles, or broken as in the case of β-cycles.

An example of an α-cycle is as follows:

attends_class $\rightarrow$ takes_notes
takes_notes $\rightarrow$ good_test_grade
good_test_grade $\rightarrow$ feels_good
feels_good $\rightarrow$ attends_class

An example of a β-cycle is as follows:

misses_class $\rightarrow$ no_notes
no_notes $\rightarrow$ poor_test_grade
poor_test_grade $\rightarrow$ low_self_esteem
low_self_esteem $\rightarrow$ misses_class

Of primary interest in this paper are partial cycles, because, in the case of partial α-cycles, they can be useful in moving an enterprise to attain its goals. We wish to reinforce or complete any α-cycle or partial α-cycle. Partial β-cycles can alert an enterprise of a vulnerable and possibly dangerous situation. To handle these detections, we wish to diminish or break any β-cycle or partial β-cycle. Algorithm 2.1, however, gives no indication that these chains of dependencies, that are almost cyclical, exist. We now present a formalism that facilitates identification and handling of partial α and β-cycles.

3.2. Ω nodes

As will be shown later in the paper, once we have identified partial α-cycles and β-cycles, we want to strengthen or weaken them. This involves the identification of a node or nodes that will effect a change in a cycle.

Some nodes, such as Student_SSN or Employee_Age, are generally considered unchangeable in terms of a particular data set. We call these *static nodes*. Other nodes, such as Cost_of_Repair or Quantity_Eaten, are changeable. We call these *Ω-nodes*.

Associated with each Ω-node is an alterability factor, Ψ, that expresses how "changeable" a particular node is. Trivially, for static nodes, Ψ would have a value of 0. The remaining nodes would have a Ψ value such that $0 < \Psi \leq 1$.

For example, we may have the following nodes in our cycle:

Node	Ψ
Age	0
Weight	.7
SSN	0
Salary	.9
Received_mailing	1.0

Age and SSN are static nodes, whereas Weight, Salary, and Received_mailing are alterable to some degree as indicated by Ψ.

3.3. Fuzzy mining of cycles

Suppose we wish to find all partial cycles in our data set that match a particular criteria. Up to this point, we group all partial cycles together, that is, all cycles with $T \leq$ U, where U is a user-specified threshold. We can partition this group into many different types of cycles by using many intermediate thresholds.

A *near-cycle* would be a cycle that is very close to being a crisp (complete) cycle. A relatively small modification in a future data set would move this cycle into the complete category. A *mid-cycle* is a cycle that is not as close to being a complete cycle as a near-cycle, but it is still relatively strong in its manifestation. A *weak-cycle* is not very strong, but is evident enough in the data set to be significant. The identification of these various strata of cycles in the data set are important. Near-cycles tell us that this cycle is on the verge of being complete. If this is a negative cycle, it is an indication that it may manifest itself soon if left unchecked. Mid- and weak-cycles are indications that patterns are there, although weakly. They give us a precursor to the possible formation of complete cycles in the future. Over time, the strengthening of such cycles may indicate specific action(s) to be taken depending upon the nature of the cycle.

We wish to quantify the linguistic concepts *near-cycle*, *mid-cycle*, and *weak-cycle*. In order to accomplish this, we define a *fuzzy cycle* to be a partial cycle that is characterized by a membership function in the following format [9]:

$$\Phi = \mu_1 / T_1, \mu_2 / T_2, \ldots, \mu_n / T_n$$

where, μ_i is the grade of membership and
T_i is the cycle threshold value

Figure 2 illustrates this containment relationship.

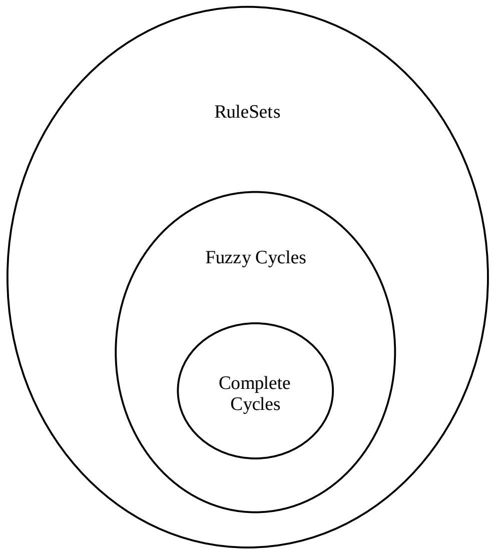

Figure 2. Cycle containment

For example, to express a near-cycle, we can use the following function:

$$\text{Near-cycle} = \{ .4/.74, .6/.76, .8/.78, .9/.80, .9/.82, 1/.84\}$$

We specify the minimum membership value to be .8. It is assumed that all $T_I > .84$ have a membership value of 1.

The membership function allows us to express various degrees of membership based upon a specific application, as well as associated domain knowledge. It is similar to the idea of expressing the concept of being tall. The membership function for "tall" would be quite different in many countries around the world. Crisp (non-fuzzy) classification would be limited to only one definition of "tallness". Therefore, it would not be an accurate imterpretation of the linguistic term "tall" in many countries where people's heights are different on the average. The fuzzy term classification is more responsive to these forms of differences and can be adjusted accordingly, via the membership function.

This allows us to use a wide range of fuzzy operators [5] to express such fuzzy concepts as an extremely-near-cycle by using the concentration operation CON(A):

$$\mu_{CON(A)}(x) = (\mu_A(x))^2$$

We can now define extremely-near-cycle as:

Extremely-Near-cycle = { .16/.74, .36/.76, .64/.78, .81/.80, .81/.82, 1/.84}

This allows us to express mid-cycle, weak-cycle, and extremely-weak-cycle in a similar manner.

3.4. Fuzzy mining algorithm

The following algorithm produces a stratification of all cycles existent in a given knowledge base. By repeatedly calling Algorithm 2.1, we find all types of cycles in the program dependency hypergraph. This algorithm uses a sequence of thresholds to identify differing strength cycles such as those defined above: complete cycles as well as near, mid, and weak cycles. Because the stratification of cycles desired by the user could contain an arbitrary number of strata, we pass in a sequence *S* of threshold values as a parameter. For each threshold U_i in the sequence, all cycles with threshold T such that $U_{i-1} \leq T \leq U_i$ will be identified. Thus, Algorithm 3.1 returns *m* classes of fuzzy cycles where *m* is the number of strata, or equivalently, the number of thresholds provided in sequence *S*.

```
Algorithm 3.1   (Fuzzy Mining of Cycles)
Input:    -Hypergraph representation P,
          - threshold sequence $S = <U_1, \ldots, U_m>$
Output:   -Sequence of m classes $<P_1, \ldots, P_m>$ of
           cycles where for each cycle $C \in P_i$ ,
           $U_{i-1} \leq T_c \leq U_i$

BEGIN ALGORITHM 3.1
For each threshold $U_i \in S$   {form cycle class $P_i$ )
    For each node h in program hypergraph P
        For each incoming neighbor b of node h
            $P'$ := Cycle Mining(P, $h \leftarrow b$, $U_i$)
            Let T be T returned by Cycle Mining
                If   $T > U_i$  then
                    discard cycle $P'$
                else
                    $P_i = P_i \cup P'$
Return $<P_1, \ldots, P_m>$
END ALGORITHM 3.1
```

To produce the above mentioned fuzzy cycle sets including complete, near, mid, and weak cycles, the following call to Algorithm 3.1 could be made.

*sets_of_cycles = Fuzzy Mining(***P***, <.85, .8, .75, .7>).*

3.5. Reinforcing α-cycles and diminishing β-cycles

It is important to note that any cycles we discover using our algorithm for a given data set are static. Only a change of data can cause a cycle to be modified. The enterprise associated with the data set must incorporate change. This will produce different data which in turn may alter the cycles already discovered.

α-cycles and β-cycles are cycles we want to perpetuate or remove respectively. In order to accomplish this we do the following:

```
For (each new data set)

    execute algorithm 2.1 (to enumerate all
                            complete cycles)
    for( each cycle)
        examine cycle (to determine the
                            set of $\alpha$-cycles
                            and the set of $\beta$-
                            cycles) *
    for (each $\alpha$-cycle)
        examine the individual nodes that
                comprise the cycle and
                determine all $\Omega$ nodes. *
        rank the nodes in terms of $\Psi$
                (measure of what to
                 modify first)
        enterprise makes changes (if
        deemed necessary)*

    for (each $\beta$-cycle)
        examine the individual nodes that
                comprise the cycle and
                determine all $\Omega$ nodes. *
        rank the nodes in terms of $\Psi$
                (measure of what to
                 modify first)
        enterprise make changes (if deemed
        necessary) *
    a new data set is generated

    * indicates non-automated step
```

Of course, much of the modification(s) are dependent on the particular enterprise as well as other domain knowledge. We are currently investigating methods of incorporating this domain knowledge into the system itself.

Of even more impact, is the reinforcing of partial α-cycles and the diminishment of partial β-cycles. In order to accomplish this, we do the following:

```
For (each new data set)

    execute algorithm 3.1 (to enumerate all
                            near-, mid-, and
                            weak- cycles)

for( each ordered near-cycle)
    examine near-cycle (to determine if it
                            is a partial α-
                            cycle or a
                            partialβ-cycle)*
    Call Make_or_Break()  [below]
for( each ordered mid-cycle)
    examine mid-cycle (to determine if it
                            is a partial α-
                            cycle or a partial
                            β-cycle)*
    Call Make_or_Break()

for( each ordered weak-cycle)
    examine weak-cycle (to determine if it
                    is a partial α-cycle or a
                    partialβ-cycle)*
    Call Make_or_Break()
    a new data set is generated

    * indicates non-automated step

Procedure Make_or_Break()
    If (partial α-cycle)
        examine the individual rules that
                comprise the cycle and
                determine all those that
                fall below U
        For (each low rule)
                If any of the nodes
                associated with the body
                of the rule are Ω nodes,
                then:
                enterprise makes changes
                associated with node(s) to
                strengthen the association *
    Else
        For (each  rule in the
              partial cycle )
                If any of the nodes
                associated with the body
                of the rule are Ω nodes,
                then:
                enterprise makes changes
                associated with node(s) to
                weaken the association  *
End Procedure.
```

4. An Example

We have implemented cycle detection work in the machine learning area of inductive logic programming. Any inductive pursuit including data mining and knowledge discovery, however, can be used to acquire the constituent *if-then* patterns of cycles. We exemplify the formalisms presented in this paper here using classification rules that are mined. A *classification rule* is a rule of the form:

$$(X_1=b_1) \leftarrow (Y_1=a_1), \ldots, (Y_n = a_n)$$

The context of the example is a business setting where the overall enterprise goal is to foster customer purchases. Consider the data instance as shown in Table 1.

Table 1. Example data.

Cust ID	Made Purchase	Filled Out Card	Age > 35	Received Mailing
10180	Yes	Yes	No	yes
10181	Yes	Yes	Yes	yes
10182	No	No	No	no
10183	Yes	Yes	Yes	yes
10184	Yes	Yes	No	no
10185	Yes	Yes	Yes	yes
10186	No	No	Yes	no
10187	Yes	Yes	Yes	yes
10188	Yes	Yes	Yes	yes
10189	Yes	Yes	Yes	yes

Assume the following rules, as shown in Table 2, along with support, confidence, and τ-values were discovered from the above data.

There are two (fuzzy) cycles embedded in this data. Assuming the thresholds of 0.8, 0.78, and 0.72 for complete, near, and weak cycles respectively, we have exactly one weak cycle with its strength T value of 0.728 comprising the rules:

$$\begin{gathered} Made\_Purchase \rightarrow Filled\_out\_Card \quad [\tau = 0.9] \\ Filled\_out\_Card \rightarrow Age > 35 \quad [\tau = 0.729] \\ Age > 35 \rightarrow Made\_Purchase \quad [\tau = 0.728] \end{gathered}$$

Table 2. Example Data.

Rule	Support	Confidence	τ
Received_mailing → Made_Purchase	0.7	1.0	0.85
Made_Purchase → Filled_out_Card	0.8	1.0	0.9
Filled_out_Card → Received_mailing	0.7	0.875	0.788
Filled_out_Card → Age > 35	0.6	0.75	0.729
Age > 35 → Made_Purchase	0.6	0.857	0.728

We also have exactly one near cycle of strength value T = 0.788 with membership function of 0.8 (as defined in the subsection entitled *Fuzzy Mining of Cycles)* made up of constituent rules:

Received_mailing $\rightarrow$ *Made_Purchase* [τ =0.85]
Made_Purchase $\rightarrow$ *Filled_out_Card* [τ = 0.9]
Filled_out_Card $\rightarrow$ *Received_mailing* [τ =0.788]

Both of these are examples of partial α cycles. To illustrate how this type of information would be useful for a particular enterprise, consider the near-cycle above. It is observed that if a customer receives a mailing, then they are likely to make a purchase at the store. Nearly all customers fill out information cards upon making a purchase. However, it is not always the case that a customer receives a mailing (thus creating the cycle) if they fill out an information card. This tells the enterprise that they should examine their process of information card handling to isolate where the possible problem may be occurring. The near-cycle detection allowed the enterprise to discover a cycle that is in their best interest to strengthen, as well as a strategy for isolating the cause of the less than complete cycle.

5. Conclusion and Future Work

We have extended a cycle mining methodology to include the identification of stratifications of cycles via a sequence of user-specified thresholds. One specific stratification was offered housing fuzzy cycles which include complete, near, mid, and weak cycles. By detecting these patterns that are close to cycles, but not quite strong enough to be considered cycles, we enable the user to either form or prevent a continuous cycle. The mechanism of the Ω-node (an alterable node) has been presented where a user can form a desirable cycle, an α-cycle, or prevent an undesirable cycle, a β-cycle, from occurring by honing in on these changeable facts.

Along with extending our formalism to automatically detect α-cycles and β-cycles, we are currently endowing our implementation with the capability to detect fuzzy cycles. We are also examining methods to allow for automated techniques to reinforce and diminish α-cycles and β-cycles respectively. We anticipate heavy experimentation with the system on data relating to Lyme Disease Diagnosis. We are also examining the interconnection of fuzzy rules into cycles as well as considering overlapping cycles.

References

[1] R. Agrawal, T. Imielinski, and A. Swami, “Mining association rules between sets of items in large databases,” *ACM SIGMOD Bulletin*, May 1993, pp. 207-216.

[2] S. Brin, R. Motwani, J. Ullman, and S. Tsur, “Dynamic Itemset Counting and Implication Rules for Market Basket Data,” *ACM SIGMOD*, May 1997.

[3] S. Chaudhuri, and U. Dayal, “An overview of data warehousing and OLAP technology,” *SIGMOD Record*, Vol.26, Num. 1, 1997, pp. 65-74.

[4] Frawley and Piatetsky-Shapiro, editors, *Knowledge Discovery in Databases*, chapter Knowledge Discovery in Databases: An Overview, AAAI Press/The MIT Press. 1991.

[5] Giarratano, J. and Riley, G., *Expert Systems: Principles And Programming*, PWS-Kent Publishing, Boston, MA, 1989.

[6] Kimball, R., *The Data Warehouse Toolkit*, John Wiley and Sons, 1996.

[7] Ramakrishnan, R., *Database Management Systems*, McGraw-Hill, 1998.

[8] J. Seitzer, J. P. Buckley, and A. Monge, “Meta-Pattern Extraction: Mining Cycles”, *Proceedings of the Florida Artificial Intelligence Research Society International Conference (FLAIRS-99)*, Orlando, FL, pp. 466-470.

[9] I. A. Zadeh, "Fuzzy Sets," *Information And Control*, Vol 8, 1965, pp. 338-353.

[10] M. J. Zaki, S. Parthasarthy, M. Ogihara, and W. Li,., “New Algorithms for Fast Discovery of Association Rules”, *3rd International Conference on Knowledge Discovery and Data Mining*, August 1997.