IN SEARCH OF ACTIONABLE PATTERNS OF LOWEST COST - A SCALABLE GRAPH METHOD

Angelina A. Tzacheva, Arunkumar Bagavathi and Aabir K. Datta

Department of Computer Science, University of North Carolina at Charlotte
North Carolina, USA-28223

ABSTRACT

Action Rules are rule based systems for discovering actionable patterns which are hidden in a large dataset. All recommended patterns from Action Rules incur some form of cost to the users. It is obvious that recommendations are interesting to the users only if the cost that the user pays for the recommended actions is low. In other words, the recommendations should be profitable or valuable to the user when they perform a chain of actions, at the lowest possible cost. In the modern era of big data, organizations are collecting massive amounts of data, growing constantly. Finding low cost actionable patterns for such large data in these domains, is time consuming and requires a scalable approach. In this work, we introduce the notion of Action Graph and propose an algorithm to search the Action Graph for actionable patterns of lowest cost. We apply the proposed algorithm to three datasets in transportation, medical, and business domains. Results show how these domains can benefit from the discovered actionable recommendations of low cost.

KEYWORDS

Low Cost Action Rules, Action Graph, Graph Search, GraphX, Pregel

1. INTRODUCTION

Data Mining is a stage of Knowledge Discovery in Databases, which identifies previously unidentified, interesting and useful patterns and trends from a large quantity data. Rule based knowledge discovery tasks intend to circumscribe methods that identify, learn or evolve 'rules' to store and manipulate knowledge. In the field of data mining, many algorithms are available to generate rules which are used for association - to find frequently associated patterns in the data and classification - to classify patterns to one or more classes. Rules takes the format as given in equation (1), where the *antecedent* (left side of the rule) is a conjunction of conditions and the *consequent* (right side of the rule) is a resulting pattern for the conditions in antecedent.

$$condition(s) \rightarrow result(s) \qquad (1)$$

The primary obstacle for such data mining and machine learning algorithms is *the lack of actionability* [1]. For example, a credit card company can assign credit scores to its customers based on their underlying classification model. For their low credit score customers, they may want to assign a person to give personal suggestions to the customer's improve credit score. Action Rule is a rule based knowledge discovery technique that recommend actionable patterns or possible transitions from one choice to another, which the user can use to their advantage. In other words, Action Rules helps to reclassify the data from one category to another, recommending patterns to improve performance of an object or establishing better work to the user. For example, one would want to find actionable patterns in the data to improve his/her salary. Some of the applications for Action Rules are: improving customer satisfaction in

business - suggesting how to improve the customer status from detractor to promoter, using online product surveys [2]. In medical domain: reducing hospital readmission in a state by giving actionable recommendations to doctors on certain procedures they can follow [3], and suggesting how to re-classify a breast cancer tumour from malignant to benign [4]. In transportation domain, suggesting how to re-classify a car condition from unacceptable to acceptable [5]. Action Rules are extracted from Decision table [6], which is more similar to the relational databases. A database becomes a decision table or decision system, when the attribute space of the data can be split into *Stable Attributes, Flexible Attributes* and a *Decision attribute*. Stable attributes in any Action Rule *AR* remain constant or cannot form action in *AR*. While flexible attributes can change their value for example attribute a change from a_i to a_j. Decision attribute is also a flexible attribute, but it is the attribute that the user has chosen to get the final decision that the user need to achieve. Action Rules can take the representation as given in equation (2), where Ψ represents a conjunction of stable features, $(\alpha \rightarrow \beta)$ represents a conjunction of changes in values of flexible features and $(\theta \rightarrow \varphi)$ represents desired decision action. Action Rules are validated using Support, Confidence, Utility and Coverage measures.

$$[(\Psi) \wedge (\alpha \rightarrow \beta)] \rightarrow (\theta \rightarrow \varphi) \qquad (2)$$

All actionable patterns given by an Action Rule subject to certain form of cost to the user [7], [8]. The extracted Action Rules are more interesting to users if the system recommends more diverse Action Rules and if Action Rules incur less cost to the users. Cost for actions in Action Rules can take a form of money, time, energy, human resources, etc. [9] Recommended actions can cause both positive and negative impact for users. Positivity in the rules is given by the measure of what amount of benefit the users can obtain from the recommendations. However, a recommendation can create negative impact if the user cannot accommodate such actions due to the cost for undertaking such actions is very high and it is not feasible for them. Thus, the actionable recommendations from a system should cause low cost to the users to make them feasible. However, most of the Action Rule discovery systems [10] [11] [12] [13] do not consider cost effectiveness for recommendations. In [7] [14], the notion of cost of the Action Rules is introduced and refined. Action Rules extraction algorithms produces very large number of Action Rules for big datasets. Searching for low cost Action Rules from such a huge volume of Action Rules can be very time consuming and requires a scalable and distributed approach for extracting them in a reasonable timeframe.

Distributed Processing frameworks like Hadoop [15] and Spark [16] have been introduced to make big data processing and data mining faster and easier. These frameworks distribute the data among nodes in a cluster of computers. Usually, these clusters are configured nodes of high computational and storage power (RAM and CPU). Thus, when the data processing work is split among those multiple high processing nodes, each of which performs computations on their part of the data, a big chunk of work gets complete quickly. Finally, when all nodes finish executing their tasks, the results are merged to present the final result. Apache provides innumerable frameworks like Hadoop [15], Spark [16], Hive, Pig to handle all such big data and distributed processing for multiple purposes. In this work, we use Apache Spark [16] framework for implementing a scalable solution to our proposed method and make it suitable for big data. Spark provides APIs such as MLlib [17] for Machine Learning tasks in a distributed setup, GraphX [18] for an efficient parallel processing in large graphs.

In this work, we utilize Action Rules produced using distributed Action Rules extraction algorithm: MR-Random Forest [19] and SARGS [20]. We introduce a graph representation for Action Rules that we extracted called *Action Graph.* We construct distributed graphs based on action terms of derived from Action Rules and their correlations. We use Spark GraphX [18] to build Action Graphs and perform implement search algorithms to discover low cost Action Rules

from the graph. We propose a distributed and a revised version of the Dijksra's shortest path [21] algorithm to search the Action Graph and discover low cost Action Rules using Pregel API [22] provided by Apache Spark. We evaluate our method with non-distributed version of the Dijsktra's algorithm and compare the times it takes to extract low cost Action Rules.

2. RELATED WORKS

More than a decade, researchers have been conducting studies on Action Rules mining to discovery actionable pattern from datasets. Some Action Rule discovery algorithms include: DEAR [10], ARAS [11] and Association Action Rules [12] in a single machine. However, with the advent of big data and constantly growing databases, the original Action Rules mining algorithms no longer can perform the mining at reasonable time. For that reason, recently, Tzacheva, et. al proposed MR-Random Forest algorithm [19] and Bagavathi, et. al proposed SARGS algorithm [20] for scalable Action Rules extraction in a distributed environment such as Hadoop MapReduce and Apache Spark to handle Big Data. However, these algorithms do not consider the Cost of the discovered Action Rules. All actionable patterns involve some form of Cost such as money, time, power and other resources to achieve the desired results [7].

Ras and Tzacheva [23] introduced the notion of cost and feasibility of Action Rules as an interestingness measure. They proposed a graph method for extracting feasible and low cost Action Rules. Ras and Tzacheva [7] proposed a heuristic search of new low cost Action Rules, where objects supporting new set of rules also supports existing rule set but the cost of reclassifying them is much lower for new rules. Later, Tzacheva and Tsay [14] proposed tree based method for extracting low cost Action Rules.

Apart from Action Rules, some research has been done on extracting Actionable knowledge. For example, Yang, et.al [24] considered *Customer Attrition* in Customer Relationship Management (CRM) in telecommunications industry and the cost complexities involved in gaining profit to all customers. They proposed a method to extract low cost Actionable patterns for converting undesired customers to loyal ones while improve the net profit of all customers. Karim and Rahman [25] proposed another method to extract cost effective actionable patterns for customer attrition problem in post processing steps of Decision Tree and Naive Bayes classifiers. Su, et.al [8] proposed a method to consider positive benefits that occurs by following an Action Rule apart from all costs that incur from the same rule. Cui, et.al [1] proposed to extract optimal actionable plans during post processes of Additive Tree Model (ATM) classifier. These actionable patterns can change the given input to a desired one with a minimum cost. Hu, et.al [26] proposed an integrated framework to gather cost minimal actions sets to provide support for social projects stakeholders to control risks involved in risk analysis and project planning phases. Lately, Hu, et.al [27] developed a cost sensitive and ensemble framework to predict software project risk predictions and conducted large scale analysis over 60 models 327 real world project samples.

In this work, we propose a graph based model to extract low cost Action Rules. We use Spark based Action Rules extraction algorithm: SARGS [20] to obtain Action Rules. We build Action Graph, based on the extracted Action Rules using Spark GraphX [18]. We propose a distributed version of the Dijkstra shortest path [21] algorithm, and implement it via Pregel API [22] to extract Action Rules of lowest cost

2. BACKGROUND – ACTION RULES, COST OF ACTION RULES AND SPARK

In this section, we give basic knowledge about Decision system, Action Rules, Spark and GraphX frameworks to understand out methodology.

2.1. Decision System

Consider an information system given in *Table 1*. Information System can be represented as $S = (X,A,V)$ where, X is a nonempty, finite set of objects: $X = \{x_1,x_2,x_3,x_4,x_5,x_6,x_7,x_8\}$, A is a nonempty, finite set of attributes: $A = a,b,c,d$ and V_i is the *domain* of attribute a which represents a set of values for attribute $i : i \in A$. For example, $V_b = b_0,b_2$.

An information system becomes a Decision system, if $A = \{A_{St} \cup A_{Fl} \cup d\}$, where d is a *decision attribute*. The user chooses the attribute d if they wants to extract desired action from $d_i : i \in V_d$. A_{St} is a set of *Stable Attributes* and A_{Fl} is a set of *Flexible Attributes*. For example, *ZIPCODE* is a Stable Attribute and *User Ratings* can be a Flexible Attribute. Let's assume from *Table1* that $c \in A_{St}$. a, b $\in A_{Fl}$ and $d \in d$. and the decision maker desires Action Rules that triggers the decision attribute change from d_1 to d_2 throughout this paper for examples.

Table 1: SAMPLE DECISION SYSTEM *S*

X	a	b	c	d
$x1$	$a1$	$b1$	$c1$	$d1$
$x2$	$a3$	$b1$	$c1$	$d1$
$x3$	$a2$	$b2$	$c1$	$d2$
$x4$	$a2$	$b2$	$c2$	$d2$
$x5$	$a2$	$b1$	$c1$	$d1$
$x6$	$a2$	$b2$	$c1$	$d2$
$x7$	$a2$	$b1$	$c2$	$d2$
$x8$	$a1$	$b2$	$c2$	$d1$

2.2. Action Rules

In this subsection, we give definitions of action terms, action rules and properties of action rules [28]. Let $S = (X, \{A \cup d\}, V)$ be a decision system, where d is a decision attribute and $V = \cup V_i : i \in A$. Action terms can be given by the expression of $(m,m_1 \rightarrow m_2)$, where $m \in A$ and $m_1,m_2 \in V_m$. $m_1 = m_2$ if $m \in A_{St}$. In that case, we can simplify the expression as (m,m_1) or $(m = m_1)$. Whereas, m_1 6= m_2 if $m \in A_{Fl}$. Action Rules can take a form of $t_1 \cap t_2 \cap .... \cap t_n$, where t_i is an atomic action or action term and the Action Rule is a conjunction of action terms to achieve the desired action based on attribute d. Example Action Rule for the Decision System in *Table 1* is given below: $(a,a_1 \rightarrow a_2).(b,b_1 \rightarrow b_2) \rightarrow (d,d_1 \rightarrow d_2)$.

2.2.1. Properties of Action Rules

Action Rules are considered interesting based on the metrics such as Support, Confidence, Utility and Coverage. Higher these values, more interesting they are to the end user. Consider an action rule R of form:

$$(Y_1 \rightarrow Y_2) \rightarrow (Z_1 \rightarrow Z_2)$$

where, Y is the condition part of R and Z is the decision part of R
Y_1 is a set of all left side action terms in the condition part of R
Y_2 is a set of all right side action terms in the condition part of R
Z_1 is the decision attribute value on left side
Z_2 is the decision attribute value on right side

In [6], the support and confidence of an action rule R is given as

$$\text{Support}(R) = \min\{\text{card}(Y_1 \cap Z_1), \text{card}(Y_2 \cap Z_2)\}$$

$$\text{Confidence}(R) = [\frac{card(Y_1 \cap Z_1)}{card(Y_1)} * \frac{card(Y_2 \cap Z_2)}{card(Y_2)}]$$

Later, Tzacheva et.al [29] proposed a new set of formula for calculating Support and Confidence of Action Rules. Their idea is to reduce complexities in searching the data several times for Support and Confidence of an Action Rule. The new formula are given below.

$$\text{Support}(R) = \text{card}(Y_2 \cap Z_2)$$

$$\text{Confidence}(R) = \frac{card(Y_2 \cap Z_2)}{card(Y_2)}$$

Tzacheva et. al [29] also introduced a notion of utility for Action Rules. Utility of Action Rules takes a following form. For most of cases Utility of Action Rules equals the Old Confidence of the same Action Rule.

$$\text{Utility}(R) = \frac{card(Y_1 \cap Z_1)}{card(Y_1)}$$

Coverage of an Action Rule means that how many decision from values, from the entire decision system S, are being covered by all extracted Action Rules. In other words, using the extracted Action Rules, *Coverage* defines how many data records in the decision system can successfully transfers from Z_1 to Z_2.

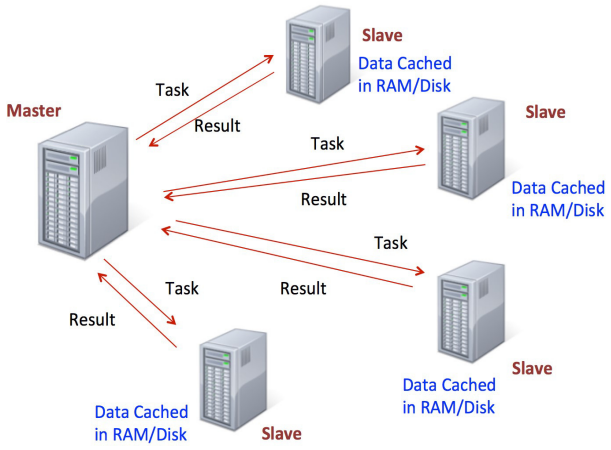

Figure 1: Overview of Spark execution using Resilient Distributed Datasets(RDD). Tasks such as transformations are given to the slave nodes. Slaves after performing the tasks, cache the result in RAM. Results can be given back to the Driver node or can be used for another transformation operation

2.3. Cost of Action Rules

Typically, there is a cost associated with changing an attribute value from one class to another - more desirable one. The cost is a subjective measure, in a sense that domain knowledge from the user or experts in the field is necessary in order to determine the costs associated with taking the actions. Costs could be monetary, moral, or a combination of the two. For example, lowering the interest percent rate for a customer is a monetary cost for the bank; while, changing the marital

status from 'married' to 'divorced' has a moral cost, in addition to any monetary costs which may be incurred in the process. Feasibility is an objective measure, i.e. domain independent.
According to the cost of actions associated with the classification part of action rules, a business user may be unable or unwilling to proceed with them. The definition of cost was introduced by Tzacheva and Ras [7] as follows:

Assume that $S = (X,A,V)$ is an information system. Let $Y \subseteq X$, $b \in A$ is a *flexible* attribute in S and $v_1, v_2 \in V_b$ are its two values. By $\wp_S(b, v_1 \rightarrow v_2)$ we mean a number from $(0,\omega]$ which describes the average cost of changing the attribute value v_1 to v_2 for any of the qualifying objects in Y. These numbers are provided by experts. Object $x \in Y$ qualifies for the change from v_1 to v_2, if $b(x) = v_1$. If the above change is not feasible, then we write $\wp_S(b, v_1 \rightarrow v_2) = \omega$. Also, if $\wp_S(b, v_1 \rightarrow v_2) < \wp_S(b, v_3 \rightarrow v_4)$, then we say that the change of values from v_1 to v_2 is more feasible than the change from v_3 to v_4. Assume an action rule r of the form:

$$(b1, v_1 \rightarrow w_1) \wedge (b2, v_2 \rightarrow w_2) \wedge \ldots \wedge (bp, v_p \rightarrow w_p) \rightarrow (d, k_1 \rightarrow k_2)$$

If the sum of the costs of the terms on the left hand side of the action rule is smaller than the cost on the right hand side, then we say that the rule r is *feasible*.

2.4. Spark

Spark [16] is a framework that is similar to MapReduce [15] to process large quantity of data efficiently in a parallel fashion and in a short span of time. The disadvantage of MapReduce framework is frequent system's disk access for writing and reading the data between Map and Reduce phases. However, Spark introduces a distributed memory abstraction strategy named Resilient Distributed Datasets(RDD). The RDDs works by splitting the data into multiple nodes, do in-memory computations on whose nodes and store the results in memory itself if there are any available space in RAM. These results can be accessed for future processes and analyses, which in-turn create another RDD. Once the RAM goes out of memory, Spark uses some strategies to push the results that are unused for a long time to the disk. Thus, Spark cuts off large number of disk accesses for storing intermediate outputs like in Hadoop MapReduce. Spark works in a *Master-Slave* approach. The Driver node(*Master*) allocate tasks to the Worker nodes(*Slaves*). Spark preserves data-locality (i.e) locating worker nodes nearer to the current node which contains a part of the data. A task that the worker perform can be either a *Transformation* or an *Action*. During *Transformation* stage, computations are made on the data split and results are stored in-memory of the worker node. Results of all worker nodes together form another RDD. While the *Action* stage on an RDD collect results from all workers and send it to the driver node or save the results to a storage system. *Figure 1*. shows an overview of the execution of Spark.

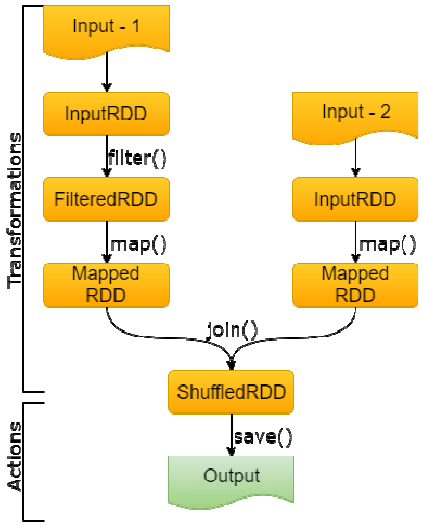

Figure 2: Spark Lineage Graph Example

Spark helps machine learning algorithms which relies on multiple iterations on the given data with the help of RDD's in memory computation. Spark handles node failures by having a lineage graph of RDDs. The lineage graph is a Directed Acyclic Graph (DAG) where each node represents a transformation stage. *Figure 2* shows a sample lineage graph of combining RDDs from two inputs. When a failure occurs at a certain stage, Spark uses the last available working point (RDD) from the lineage graph and restart all computations from that working point rather than repeating the entire process from the beginning or saving the intermediate results and replicating them across multiple nodes. This strategy of data management, fault tolerance and in-memory processing make Spark to do computations faster than MapReduce

2.5. Spark GraphX

Spark, with its efficiency in Resilient Distributed Datasets

(RDDs) help wide variety of applications such as Machine Learning with MLlib library [17], Graph Analysis with GraphX library [18]. GraphX is an embedded graph processing framework built on top of Apache Spark. In general, graphs can be represented as $G=(V,E)$, where V is the set of vertices in G and E, which takes the general representation as $e_{ij} = Edge(i,j)$, is the set of edges connecting 2 vertices (i,j) in G. GraphX treats the complete graphs as an RDD. It maintains the graph RDD in the type of [VD, ED], where *VD* and *ED* are other RDDs representing vertex properties and edge properties respectively. *Figure 5* provides the simple GraphX framework and functions it provide to support various graph operations. GraphX performs graph-specific operations as a series of distributed *map()*, *join()* and *reduce()* functions of RDDs. Besides these functions, GraphX comprise of Google's *Pregel API* [22]. GraphX uses Pregel API to perform iterative tasks like PageRank, Graph search algorithms like Depth First Search (DFS) and Breadth First Search (BFS) and finding shortest routes in graphs like Dijkstra's algorithm. In iterative graph algorithms, vertices of the graph have to pass some messages to their neighbours. Since the graph is maintained as a single RDD in GraphX, the message passing is complicated compared to other graph libraries. The Pregel API automates this message sending and receiving module and provides a functionality to do these jobs efficiently to suit the Spark environment. GraphX also shows great speedups for iterative graph algorithms such as PageRank compared to other graph libraries such as GraphLab [30] and Giraph [31]. For iterative graph processing, GraphX provides *Pregel API* [22]. Pregel works in a message passing

fashion between the graph vertices. In GraphX, Pregel has three functions: *sendMsg()* - to process and send a message to a vertex's immediate neighbours, *mergeMsg()* - to merge all messages from a vertex's immediate neighbors and *receiveMsg()* - to receive and process the merged message. Following these steps, each vertex can share and collect information with their neighbours. With this method, the information can flow from one end of the graph to another gradually. For iterative procedure, Pregel iterations are named as *super steps*. In each super step, each vertex executes all three above mentioned functions.

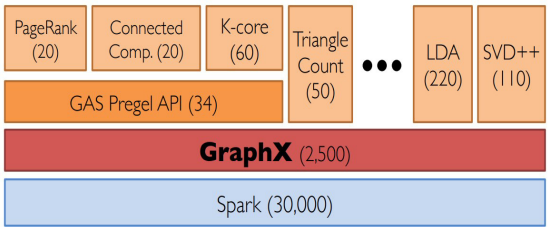

Figure 3: GraphX Framework with basic graph algorithms

4. METHODOLOGY

In this work, we propose a graph-based method to search for optimal low cost Action Rules. To extract low cost Action Rules, first we extract Action Rules with a distributed mechanism: SARGS [20]. From the extracted Action Rules, we build an Action Graph. We then propose a method based on Dijkstra's algorithm to search the Action Graph for low cost Action Rules. In this section, we give the SARGS algorithm, Action Graphs and our search algorithm to extract low cost Action Rules

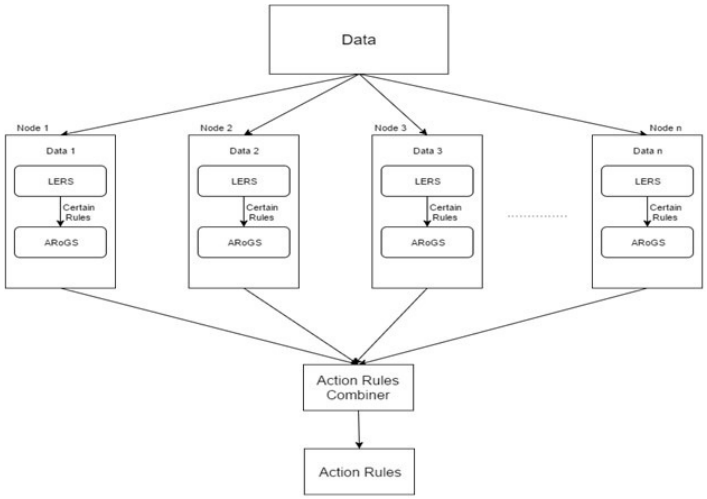

Figure 4: Distributed Actionable Pattern Mining using SARGS algorithm overview

4.1. Action Rules extraction using SARGS

The SARGS algorithm propsed in [20] uses LERS [32] and ARAS [11] methods for extracting Action Rules in a distributed fashion for larger datasets. *Figure 4* gives an overview of the SARGS algorithm. SARGS algorithm consists of 3 modules namely: Data distribution, LERS and ARAS.

4.1.1. Data distribution Module

The data distribution module is to evenly distribute the data based on the decision attribute. The main objective of the data distribution module is to overcome the obstacle of inaccurate knowledge discovery while extracted in a distributed setup. The given input data is split into n groups, where n=*no. of decision attribute vales* and each group consists of records from the information system matching the corresponding decision value. Also, the proportion constraint $P_g \approx P_S$ is maintained, where P_g is the proportion of records in a partition g with decision attribute value d_i and P_S is the proportion of records in the given information system S with decision attribute value d_i. By this way, each partition contains same proportion of data which is equal to the original dataset. The final actionable knowledge from these partitions are considered to be equal to that of the knowledge from the single data. *Figure 5* shows an example data partition for the information system S shown in *Table 1*.

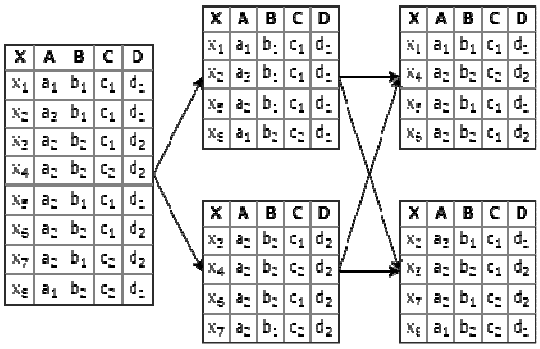

Figure 5: Example Data Distribution in SARGS for the Decision System given in Table 1

4.1.2. Data distribution Module

The second module in the SARGS algorithm is the LERS [32]. LERS is a Learning from Examples based on Rough Sets which extracts classification rules from the information system. SARGS follows distributed method of generating classification rules using LERS system. Using the information system S from *Table 1*, LERS strategy can find all certain and possible rules describing decision attribute d in terms of attributes $a,b,$ and c. Since LERS follows bottomup strategy, it constructs classification rules with conditional part covering x attributes, then it continues to construct rules with conditional part of $x + 1$ attributes during the following iterations. Only marked rules from the LERS module are considered for the ARAS module. A classification rule c_i if and only if $S_{ci} \subseteq S_{d*}$, where S_{ci} is the set of rows in S that support the classification rule c_i and S_{d*} is the set of rows in S that support the decision attribute value d_*.

4.1.3. Modified LERS Module

The third module in the SARGS method is the modified version of ARAS [11] and it uses all marked classification rules from the second (LERS) module and derives Action Rules. ARAS method, which extracts incomplete Action Rules, may not be useful when the user requires valid recommendations. Sample Action Rules from the system ARAS for the Decision System S given in *Table 1* are given below:

$$ARs_1 : (d_1 \rightarrow d_2) = (a, \rightarrow a_2).(b, \rightarrow b_2) \Rightarrow (d, d_1 \rightarrow d_2)$$
$$ARs_2 : (d_1 \rightarrow d_2) = (a, \rightarrow a_2).(c, c_2) \Rightarrow (d, d_1 \rightarrow d_2)$$
$$ARs_3 : (d_1 \rightarrow d_2) = (b, \rightarrow b_1).(c, c_2) \Rightarrow (d, d_1 \rightarrow d_2)$$

$$ARs_4 : (d_1 \rightarrow d_2) = (b, \rightarrow b_2).(c, c_1) \rightarrow (d, d_1 \rightarrow d_2)$$

This method gives the modified version of ARAS module that the SARGS algorithm uses to extract all complete Action Rules. This algorithm extracts all missing values from the conditional (left) part of the given Action Rule. The algorithm then get cartesian product of all missing values (except the values of same attribute) and fills in the action rule. Following Action Rules are extracted from the decision system S given in *Table 1* using SARGS method.

$$AR_1(d_1 \rightarrow d_2) = (A, a_1 \rightarrow a_2).(B, \rightarrow b_2) \rightarrow (D, d_1 \rightarrow d_2)$$
$$AR_2(d_1 \rightarrow d_2) = (A, a_3 \rightarrow a_2).(B, \rightarrow b_2) \rightarrow (D, d_1 \rightarrow d_2)$$

4.2. Action Graphs

We build a graph called *Action Graph* from the Action Rules extracted using the SARGS algorithm. We build Action Graph by using *action terms* in Action Rules and their relation with other action terms. In general, graphs take the representation of $G = (V,E)$, where V is a set of vertices and E is a set of edges connecting vertex pairs in V. All vertices and edges can contain properties that combined together uniquely represent vertices and edges respectively. We represent our Action Graph as an undirected graph $A_g = (A_v, A_e)$. In Action Graph, we treat action terms that we get from Action Rules as a set of vertices (A_v) and we create edge between a vertex pair $(a_m, a_n | a_m, a_n \in r_i)$, where r_i is an Action Rule. We set basic properties of an action term such as *Vertex Id*, *Name*, *Cost*, *Support*, *Neighbour Ids* and *Action Rules* of low cost based on the vertex as vertex properties of the Action Graph and *Cooccurrence Frequency* of a vertex pair as an edge property. For example, red node means highest frequency, yellow node means medium frequency, and blue node means low frequency. *Figure 6* gives a sample Action Graph for Action Rules extracted from *Table 1* using the SARGS algorithm.

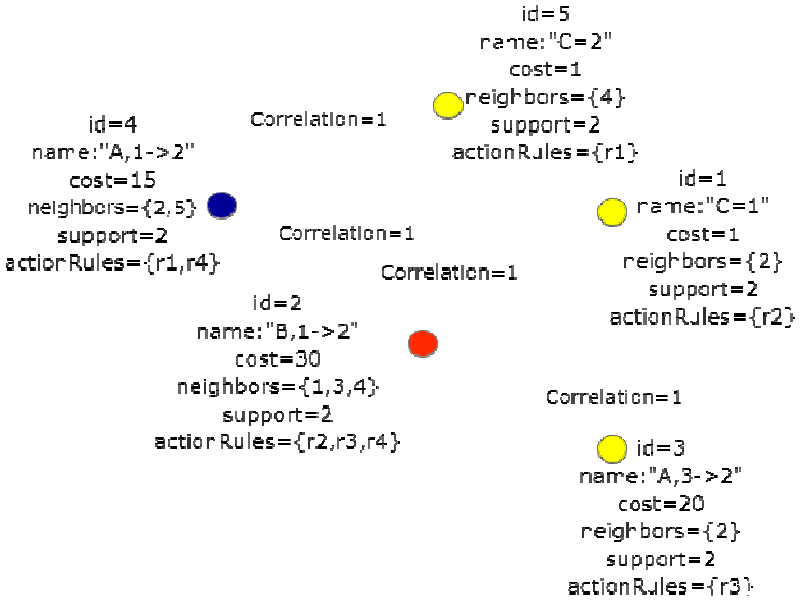

Figure 6: Sample Action Graph with Vertex Properties and Edge weights; Vertex color represents how frequently the action term occurs, with Red being the most frequent, and Yellow the least frequent.

4.3. Action Graph search algorithm for extracting Action Rules of lowest cost

Algorithm 1 Action Graph Search Algorithm for Action Rules of Lowest Cost

Require: $A_g = (A_v, A_e)$ and a cost threshold ρ value

```
    procedure RECEIVEMSG(message)
 2:     for m ∈ message do
            a_n := m.getActionRule().union(MyName)
 4:         if a_n ∉ MyActionRules and r_ρ < ρ then
                Add (r, r_ρ) to MyActionRules
 6: procedure SENDMSG(edges)
                                ▷ ∀e, e ∈ A_e|e ∈ edges, A_e ∈ A_g
 8:     for e ∈ edges do
            send r to e.dstn
10:             ▷ r ⊆ MyActionRules|r ⊆ e.dstn.neighbors
    procedure MERGEMSG(m1,m2)
12:                             ▷ m1,m2 are of type (r, r_ρ)
        return m1+m2
14:
    A'_g ← (ReceiveMsg, SendMsg, MergeMsg)
```

Algorithm 1: Action Graph Search Algorithm for Action Rules of Lowest Cost

Algorithm 1 gives an overview of our search algorithm with functions to send, receive and merge messages. The basic idea behind our search algorithm is very similar to *Dijkstra's shortest path algorithm* [21] adapted to distributed environment on cloud. In each iteration: all vertices share their action term with its cost with their neighbours; all vertices add action terms arriving from neighbours to their dictionary; all vertices combine the valid low cost action terms with the ones already in their dictionary; the resulting action rules are sorted by cost in descending order; finally, all vertices share the set of low-cost action rules with their neighbours; algorithm runs for *n – iterations*, where *n* is the number of action terms in the longest action rule, from the input list of action rules. The search algorithm takes the Action Graph $A_g = (A_v, A_e)$, where A_v is a set of vertices or action terms and A_e is a set of edges connecting vertex pairs in A_v, and minimum cost threshold ρ. We send an initial empty message to start the functions. The first function to execute is the *ReceiveMsg()*. For better readability we explain in the order of *SendMsg()*, *MergeMsg* and *ReceiveMsg()*. *Steps 6-10* gives procedure to do for all vertices when they need to send a message to their immediate neighbours. Each vertex process each edge originating from them. For each available low cost Action Rule *r*, it checks if $r \subseteq$ *dstn.neighbors* in Step *9*. This step filters the dictionary in each vertex remove action terms that are irrelevant to the destination vertex. To avoid duplicate rules from multiple vertices, we send only the combination of action terms that are new to the destination vertex. In *Steps 11-13* we give a procedure for each vertex to combine messages from multiple vertices. This function simply combines all messages (dictionaries of action terms with their Costs) and into a single Dictionary. This single Dictionary is processed via the *ReceiveMsg()* function for processing. In *Steps 1-5* we show the processing the *ReceiveMsg()* performs - for all vertices when they receive a message. When a vertex receives a set of action term combinations and their corresponding costs, it adds its own cost to produce a Low Cost Action Rule. If the total cost is less than or equal to the given cost threshold ρ, the vertex adds the Action Rule to its list of Low Cost Action Rules. The main function is described in *Step 16*, where we initiate the first *messageSend()* operation to $v \in A_v$. First, we populate *Action Rules* property of each vertex to the combination of current vertex and its

immediate neighbour and respective cost. Next, all vertices send an empty message to all their immediate neighbours. This continues for n iterations as mentioned above. Once all iterations are over, we obtain an Action Graph A^0_g containing Action Rules along with their cost for each vertex. We then sort the rules by cost in Descending Order and suggest to the user the top 5 lowest cost rules for each vertex. The top 5 lowest cost Action Rules from all vertices form the set of the discovered Action Rules of Lowest Cost.

4.4. Post-processing: Action set correlations of low cost Action Rules

By following the *Algorithm*1, we obtain all low cost Action Rules. Some Action terms in Action Rules may have high correlations. We propose a method to reduce further the cost of the obtained rules by considering edge weights in our *Action Graph*. We assign edge weights between two vertices or action terms based on their frequencies of co-occurring together in Action Rules. We define a correlation threshold η to check if two action terms in an Action Rule is highly correlated. We assume that two action terms $a_{r1},b_{r1}|(a_{r1},b_{r1}) \in r1$, where $r1$ is an Action Rule, to be highly correlated if their co-occurring frequency w is greater than or equal to η. We propose that when two action terms satisfy the $w \geq \eta$, then the action suggested by the first term is expected to trigger the action suggested by the second one. Therefore, the lowest cost action can be dropped from the total cost. For each vertex, we define a correlation matrix, which gives correlation frequency between the current vertex or action term and its neighbour. *Figure 7* gives a sample correlation matrix for the action term vertex $(b,1 \rightarrow 2)$. With this correlation matrix, we can identify which 2 terms are highly correlated. Then we process each Action Rule from the dictionary of low cost Action Rules of the current vertex. When a highly correlated pair occurs in the Action Rule, we drop the cost of lowest cost action term. For example, cost of the Action Rule $(b,1 \rightarrow 2) \cap (c = 1)$ can be reduced from *31* to *30*, if the correlation threshold η is set to *1*.

	(a,3->2)	(b,1->2)	(c=1)
(a,3->2)	0		
(b,1->2)	1	0	
(c=1)	0	1	0

Figure 7: Example Correlation Matrix of the action term $(b,1 \rightarrow 2)$

5. EXPERIMENTS AND RESULTS

To test our methods, we use three datasets: *Car Evaluation* data, *Mammographic Mass* data, and the city of *Charlotte North Carolina BusinessWise* data.

Table 2: Dataset properties

Property	Car Evaluation Data	Mamm. Mass Data	Business Data
# of instances	1728	961	22441
Attributes	7 attributes -Buying -Maintenance -Doors -Persons -Luggage Boot -Safety -Class	6 attributes -BI-RADS -Patient's age -Shape -Margin -Density -Severity	17 attributes including -City -Sector -Site Type -Building Type -Estimated Sales -Total Employees Count
Decision attribute values	Class (unacc, acc, good, vgood)	Severity (0 - benign, 1malignant)	Estimated Sales (<$2M,2-
# of instances / decision value	unacc - 1210 acc - 384 good - 69 vgood - 65	0 – 516 1 - 445	<$2M – 12503 $2-$10M – 1927 $10-$25M – 393 $25-$50M – 130 $50-$100M – 69 $100M-$500M – 57 >$500M – 50
Data size	52 KB	16 KB	5.5 MB

Table 3: Parameters used for Action Rules discovery using SARGS algorithm

Property	Car Evaluation Data	Mamm. Mass Data	Business Data
Stable attributes	Doors, Persons	Age	Start Year
Required decision action	(Class) *unacc* → *acc*	(Severity) 1 → 0	Estimated Sales $2*M* – $10*M* → $10*M* – $24*M*
Minimum Support α and Confidence β	2, 70%	2, 70%	100, 70%
Cost Threshold φ	1500	2000	3000

The Car Evaluation and Mammography are obtained from the Machine Learning repository of the Department of Information and Computer Science of the University of California, Irvine [33]. The Car Evaluation Data consists of records describing a car's goodness and acceptability based on features such as buying frequency, maintenance cost, safety measure, seating capacity and luggage boot size. Mammographic is the most effective method for screening breast cancer. The Mammographic Mass data contains records that measure severity of the cancer based on patient's age, cancer shape, cancer density and BI-RADS(a test score to denote how severe the cancer is).

Table 4: Example Action Rules of lowest cost for Car Evaluation Dataset

Low Cost Action Rules
1. AR_{C4} : (*buying, high → med*) ∧ (*lugBoot, med → small*) ∧ (*maint, low → med*) ∧ (*persons* = 4) ∧ (*safety, low → high*) ➔ (*class, unacc → acc*)[*Support* : 4,*OldConfidence*: 100%, *New Confidence* : 100%, *Utility* : 100%] *COST* : 1300.0 2. AR_{C5} : (*buying, low → med*) ∧ (*lugBoot, med → big*) ∧ (*maint, low → med*) ∧ (*persons* = *more*) ∧ (*safety, low → med*) ➔ (*class, unacc → acc*)[*Support* : 4,*OldConfidence*: 100%, *New Confidence*: 100%, *Utility*: 100%] *COST*: 1400.0
Low Cost Action Rules after Correlation
1. AR_{C4} : (*buying, high → med*) ∧ (*lugBoot, med → small*) ∧ (*maint, low → med*) ∧ (*persons* = 4) ∧ (*safety, low → high*) ➔ (*class, unacc → acc*) [*Support*: 4, *Old Confidence*: 100%, *New Confidence*: 100%, *Utility*: 100%] *COST*: 1000.0 2. AR_{C5} : (*buying, low → med*) ∧ (*lugBoot, med → big*) ∧ (*maint, low → med*) ∧ (*persons* = *more*) ∧ (*safety, low → med*) ➔ (*class, unacc → acc*) [*Support* : 4,*Old Confidence*: 100%, *New Confidence*: 100%, *Utility* : 100%] *COST*: 1100.0

The city of Charlotte North Carolina BusinessWise data, which was donated by the Charlotte Chamber of Commerce. This data collects details of over 20,000 business companies in Mecklenburg county, North Carolina. The data includes their City, Start Year, Sector, Specialization of the company in a selected sector, Site Type, Employees count at the site, Total employees in the company including all branches, Site building type, Total sites and Estimated Sales. From this data, our focus is how to increase the Estimated Sales amounts in USD. We show detailed description of each dataset properties in *Table 2* which we use to test our algorithm.

Table 3 give parameters that we set for each dataset to collect Action Rules. For the *Car Evaluation* data, we choose *Class* attribute as a decision attribute and we collect Action Rules to help the car company to change the car from *Unacceptable* state to *Acceptable* state. For the *Mammographic Mass* data, we choose *Cancer Severity* as a decision attribute and we collect Action Rules to suggest Actions to doctors on how to reduce the tumour severity from *Malignant* to *Benign*. For *Business Data* we choose class attribute as *Estimated Sales*, and we collect Action Rules to suggest Actions to business on how to increase their Estimated Sales in USD from the range *2million-10million* USD to *10millon-24million* USD.

With our datasets and using parameters that we set in *Table 3*, we run the SARGS algorithm on each data. We collect Action Rules which meet the minimum support(α), and minimum confidence(β), threshold. We record the cost(φ) for each action term. We calculate Total Cost of each Action Rule by adding the cost of all action terms in the rule. Usually, the cost of each action term is specified by an expert in the domain. For example, for the Mammography dataset, a medical doctor specifies the cost for the suggested actions. For Car Evaluation data, the car manufacturer specifies the cost for the suggested actions. However, for our experiment purpose, we assign a random cost number to each action term. We assign the cost of *0* for action terms which have stable attributes, because the stable attributes cannot be changed. For the Flexible Attributes, we set the cost values between 0 and 1000. In *Table 4*, *Table 5* and *Table 6*, we show samples of Low Cost Action Rules, and Low Cost. Action rules - after post-processing steps or Correlation over the low cost Action Rules, for the *Car Evaluation Data*, *Mammographic Mass Data* and *Business Data* respectively. These rules support the parameters which we set in *Table 3*. In *Table 4*, *Table 5* and *Table 6*, low cost Action Rules are ones which has cost less than φ.

Table 5: Example Action Rules of lowest cost for Mammographic Dataset

Low Cost Action Rules
1. AR_{M4}: (*BI−RADS,* 6 → 4) ∧ (*Density,* 3 → 2) ∧ (*Margin,* 5 → 1) ➔ (*Severity,* 1 → 0) [*Support*: 25*, Old Confidence*: 100%*, New Confidence*: 100%*, Utility*: 100%] *COST*: 550.0 2. AR_{M5}: (*Age* = 60) ∧ (*BI − RADS,*6 → 4) ➔ (*Severity,* 1 → 0) [*Support*: 11*, Old Confidence*: 100%*, New Confidence*: 100%*, Utility*: 100%] *COST*: 450.0 3. AR_{M6}: (*BI−RADS,* 6 → 4) ∧ (*Margin,* 5 → 1) ∧ (Shape*, 3* → 2) ➔ (*Severity,* 1 → 0) [*Support*: 13*, Old Confidence*: 100%*, New Confidence*: 100%*, Utility*: 100%] *COST*: 800.0
Low Cost Action Rules after Correlation
1. AR_{M4}: (*BI−RADS,* 6 → 4) ∧ (*Density,* 3 → 2) ∧ (*Margin,* 5 → 1) ➔ (*Severity,* 1 → 0) [*Support*: 25*, Old Confidence*: 100%*, New Confidence*: 100%*, Utility*: 100%] *COST*: 450.0 2. AR_{M5}: (*Age* = 60) ∧ (*BI − RADS,*6 → 4) ➔ (*Severity,* 1 → 0) [*Support*: 11*, Old Confidence*: 100%*, New Confidence*: 100%*, Utility*: 100%] *COST*: 400.0 3. AR_{M6}: (*BI−RADS,* 6 → 4) ∧ (*Margin,* 5 → 1) ∧ (Shape*, 3* → 2) ➔ (*Severity,* 1 → 0) [*Support*: 13*, Old Confidence*: 100%*, New Confidence*: 100%*, Utility*: 100%] *COST*: 600.0

These Action Rules define what actions do a company/user should employ to achieve their desired goal. For example, the rule AR_{C4} recommends that if a car company decreases the *Buying Cost* from *high* to *medium* and increases the *Maintenance Cost* from *low* to *medium* and increases *Safety Measures* from *low* to *high* and if the *Seating Capacity* is *4*, then the Car Condition may change from *Unacceptable* to *Acceptable* with the cost of 1300.0. For all datasets, we consider cost just as a measure of an Action Rule since the actual costs are assigned by experts.

Next, we build an Action Graph using the list of extracted Action Rules as an input. We implement the Action Graph in both non-parallel environment, and in a clustered environment for performance and scalability comparison. The Non-parallel version is implemented in Java. The Apache Spark [16] using the Spark GraphX library. We use Scala programming language. We test the system on a Spark cluster running over Hadoop YARN. The cluster has 6 nodes connected via 10 GigaBytes per second Ethernet network. We use Pregel API [22] in Spark GraphX [18] framework to search the Action Graph in an iterative procedure by using the Algorithm described in figure Algorithm 1. This algorithm returns all low cost Action Rules ($cost < \varphi$). From these Action Rules, we do post processing step and highly correlating action terms pair ($correlation\ frequency \geq \eta$). If there is any correlation pair in the Action Rule, we drop the lowest cost in that pair. For example, consider the low cost Action Rule AR_{B5} from Table6. Cost of this Action Rule is 1394.0. In the post-processing step, we find that the Action Term (*EMPSITE,*4−9*Employees* → 10 − 24*Employees*) of cost 504.0 co-occurs frequently with the action term (*EMPALLSITE,*25 − 49*Employees* → 500 − 999*Employees*) of cost 63.0. So, we consider that one of these action terms trigger the other action to happen eventually. Thus, we drop the cost of (*EMPALLSITE,*25−49*Employees* → 500−999*Employees*) and reduce the cost of the Action Rule to 1213.0.

Table 6: Example Action Rules of lowest cost for Charlotte Businesswise Dataset

Low Cost Action Rules
1. AR_{B1}: (*EMPALLSITE,* 50 – 99 *Employees* → 250 – 499 *Employees*) ∧ (*EMPSITE,* 50-99 *Employees* → 4–9 *Employees*) ∧ (*SECTOR, Services* → *Retail Trade*) ∧ (*STARTYR* = 2006 – 2010) ➔ (*ESTSALES,* $2*M* – $10*M* → $10*M* – $24*M*) [*Support*: 2, *Old Confidence*: 60%, *New Confidence*: 66%, *Utility*: 90%] *COST*:1615.0 2. AR_{B2}: (CITY, Matthews → Charlotte) ∧ (*EMPALLSITE, 25*–49 *Employees* → 500–999 *Employees*) ∧ (*EMPSITE, 4*-9 *Employees* → 10–24 *Employees*) ∧ (OWNBLDG, Y → N) ➔ (*ESTSALES,* $2*M* – $10*M* → $10*M* – $24*M*) [*Support*: 2, *Old Confidence*: 60%, *New Confidence*: 66%, *Utility*: 90%] *COST*:1276.0
Low Cost Action Rules after Correlation
1. AR_{B1}: (*EMPALLSITE,* 50 – 99 *Employees* → 250 – 499 *Employees*) ∧ (*EMPSITE,* 50-99 *Employees* → 4–9 *Employees*) ∧ (*SECTOR, Services* → *Retail Trade*) ∧ (*STARTYR* = 2006 – 2010) ➔ (*ESTSALES,* $2*M* – $10*M* → $10*M* – $24*M*) [*Support*: 2, *Old Confidence*: 60%, *New Confidence*: 66%, *Utility*: 90%] *COST*:1330.0 2. AR_{B2}: (CITY, Matthews → Charlotte) ∧ (*EMPALLSITE, 25*–49 *Employees* → 500–999 *Employees*) ∧ (*EMPSITE, 4*-9 *Employees* → 10–24 *Employees*) ∧ (OWNBLDG, Y → N) ➔ (*ESTSALES,* $2*M* – $10*M* → $10*M* – $24*M*) [*Support*: 2, *Old Confidence*: 60%, *New Confidence*: 66%, *Utility*: 90%] *COST*:1213.0

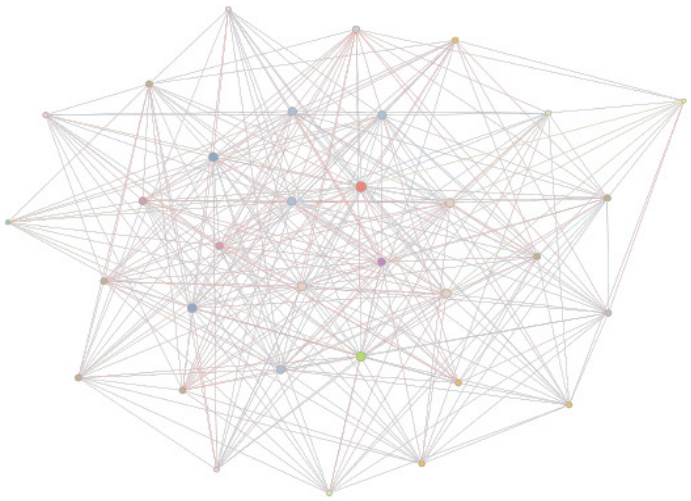

Figure 8: Action Graph for Action Rules from Car Evaluation Dataset

A visualization of the Action Graph for the *Car Evaluation* data is shown in in *Figure 6*. The colour and size of the vertex *v* in the Action Graph represent how frequently a specific action term occurs in our Action Rules set. The more frequent action terms are shown in larger size nodes, and the less frequent in smaller size nodes. Also, the darker colours signify the most frequently occurring action terms, and the lighter colours less frequent action terms. For the Car Evaluation data, the action term *(persons=more)* occurs most frequently, is shown in the red colour *red node* in *Figure 6*, and the action term *(safety, low → high)* is the second most frequent term in our Action Rules set, which is shown in green colour *node* in Figure 6. Table 7 gives details about the number of Action Rules, and the processing time in seconds for the proposed algorithm to build Action Graphs (one for each dataset) and basic properties of these graphs such as number of nodes and edges.

Table 7: Action Graph Properties for different datasets

Property	Car Evaluation Data	Mamm. Mass Data	Business Data
No.ofAction Rules	415	290	2043
No.ofAction Terms / Nodes	33	98	224
Minimum Support α and Confidence β	2, 70%	2, 70%	2, 60%
Cost Threshold φ	1500	900	3000

In *Table 8*, we give our system's runtime performance comparing with non distributed version of the same algorithm. The distributed version of the Action Graph, which we implement in Apache Spark [16] using the GraphX [18] library, and the Pregel API [22], shows faster processing times for large datasets compared to single machine implementation in Java.

Table 8: Analysis on Action Graphs for Low Cost Action Rules

Dataset	Non-distributed Algorithm	Distributed Algorithm
Car Evaluation Data	1.2 mins	7.1 secs
Mamm. Mass Data	17 secs	5.8 secs
Business Data	> 10 mins	3.1 mins

6. CONCLUSION

The distributed version of the Action Graph Search for Lowest Cost Action Rules, which we implement in Apache Spark [16] using the GraphX [18] library, and the Pregel API [22], shows faster processing times for large datasets compared to single machine implementation in Java. Our proposed method presents an improvement over the Search for Action Rules of Lowest Costs in [7], as we use a distributed version for Graph Search, which is suitable to scale well for big datasets. In addition, it addresses a significant drawback of the previous method, which is using a heuristic search, and hence sometimes it is unable to reach the goal, and discovery any rules. The new proposed method always reaches the goal and discovers the rules of lowest cost. In the future, we plan to build a Decision Tree like structure, which can be searched, and shows the Lowest Cost Action Rules at the leaves of the tree. We plan to improve the support and confidence of the discovered Low Cost Action Rules by incorporating these parameters into the search procedure. We also plan to use the proposed Graph structure in order to design a distributed version of Association Action Rules extraction algorithm.

REFERENCES

[1] Z. Cui, W. Chen, Y. He, and Y. Chen, "Optimal action extraction for random forests and boosted trees," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 179–188.

[2] J. Kuang, A. Daniel, J. Johnston, and Z. W. Ras, "Hierarchically´ structured recommender system for improving nps of a company," in *International Conference on Rough Sets and Current Trends in Computing*. Springer, 2014, pp. 347–357.

[3] M. Al-Mardini, A. Hajja, L. Clover, D. Olaleye, Y. Park, J. Paulson, and Y. Xiao, "Reduction of hospital readmissions through clustering based actionable knowledge mining," in *Web Intelligence (WI), 2016 IEEE/WIC/ACM International Conference on*. IEEE, 2016, pp. 444– 448.

[4] A. A. Tzacheva, E. A. Koenig, and J. R. Pardue, "Actions ontology system for action rules discovery in mammographic mass data," in *Proceedings of the International Conference on Data Mining (DMIN)*.

[5] The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013, p. 1.

[6] A. A. Tzacheva, C. C. Sankar, S. Ramachandran, and R. A. Shankar, "Support confidence and utility of action rules triggered by metaactions," in *Knowledge Engineering and Applications (ICKEA), IEEE International Conference on*. IEEE, 2016, pp. 113–120.

[7] Z. W. Ras and A. Wieczorkowska, "Action-rules: How to increase profit of a company," in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2000, pp. 587–592.

[8] Z. W. Ras and A. A. Tzacheva, "In search for action rules of the lowest´ cost," in *Monitoring, Security, and Rescue Techniques in Multiagent Systems*. Springer, 2005, pp. 261–272.

[9] P. Su, D. Li, and K. Su, "An expected utility-based approach for mining action rules," in *Proceedings of the ACM SIGKDD Workshop on Intelligence and Security Informatics*, ser. ISI-KDD '12. New York, NY, USA: ACM, 2012, pp. 9:1–9:4. [Online]. Available:http://doi.acm.org/10.1145/2331791.2331800

[10] A. A. Tzacheva, R. S. Shankar, R.A, and A. Bagavathi, "Action rules of lowest cost and action set correlations," in *Fundamenta Informaticae Journal, European Association for Theoretical Computer Science (EATCS), IOS Press*.

[11] L.-S. Tsay* and Z. W. Ras, "Action rules discovery: system dear2,´ method and experiments," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 17, no. 1-2, pp. 119–128, 2005.

[12] Z. W. Ras, E. Wyrzykowska, and H. Wasyluk, "Aras: Action rules´ discovery based on agglomerative strategy," in *International Workshop on Mining Complex Data*. Springer, 2007, pp. 196–208.

[13] Z. W. Ras, A. Dardzinska, L.-S. Tsay, and H. Wasyluk, "Association action rules," in *Data Mining Workshops, 2008. ICDMW'08. IEEE International Conference on*. IEEE, 2008, pp. 283–290.

[14] S. Im and Z. W. Ras, "Action rule extraction from a decision table:´ Ared," in *International Symposium on Methodologies for Intelligent Systems*. Springer, 2008, pp. 160–168.

[15] A. A. Tzacheva and L.-S. Tsay, "Tree-based construction of low-cost action rules," *Fundamenta Informaticae*, vol. 86, no. 1, 2, pp. 213–225, 2008.

[16] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[17] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228301

[18] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen *et al.*, "Mllib: Machine learning in apache spark," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1235–1241, 2016.

[19] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph processing in a distributed dataflow framework." in *OSDI*, vol. 14, 2014, pp. 599–613.

[20] A. A. Tzacheva and Z. W. Ras, "Association action rules and action paths triggered by meta-actions," in *Granular Computing (GrC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 772–776.

[21] A. Bagavathi, P. Mummoju, K. Tarnowska, A. A. Tzacheva, and Z. W. Ras, "Sargs method for distributed actionable pattern mining using spark," in *2017 IEEE International Conference on Big Data (Big Data)*, Dec 2017, pp. 4272–4281.

[22] M. M. Rathore, A. Ahmad, A. Paul, and G. Jeon, "Efficient graphoriented smart transportation using internet of things generated big data," in *Signal-Image Technology & Internet-Based Systems (SITIS), 2015 11th International Conference on*. IEEE, 2015, pp. 512–519.

[23] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[24] Z. W. Ras, A. A. Tzacheva, L.-S. Tsay, and O. Giirdal, "Mining for interesting action rules," in *Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*. IEEE, 2005, pp. 187–193.

[25] Q. Yang, J. Yin, C. Ling, and R. Pan, "Extracting actionable knowledge from decision trees," *IEEE Transactions on Knowledge and data Engineering*, vol. 19, no. 1, pp. 43–56, 2007.

[26] M. Karim and R. M. Rahman, "Decision tree and naive bayes algorithm for classification and generation of actionable knowledge for direct marketing," *Journal of Software Engineering and Applications*, vol. 6, no. 04, p. 196, 2013.

[27] Y. Hu, J. Du, X. Zhang, X. Hao, E. Ngai, M. Fan, and M. Liu, "An integrative framework for intelligent software project risk planning," *Decision Support Systems*, vol. 55, no. 4, pp. 927–937, 2013.

[28] Y. Hu, B. Feng, X. Mo, X. Zhang, E. Ngai, M. Fan, and M. Liu, "Costsensitive and ensemble-based prediction model for outsourced software project risk prediction," *Decision Support Systems*, vol. 72, pp. 11–23, 2015.

[29] Z. W. Ras and A. Dardzi´ nska, "From data to classification rules and´ actions," *International Journal of Intelligent Systems*, vol. 26, no. 6, pp. 572–590, 2011.

[30] S. R. A.A. Tzacheva, C.C. Sankar and R. Shankar, "Support confidence and utility of action rules triggered by meta-actions," in *proceedings of 2016 IEEE International Conference on Knowledge Engineering and Applications*, ser. ICKEA 2016. IEEE Computer Society, 2016.

[31] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: a framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.

[32] C. Avery, "Giraph: Large-scale graph processing infrastructure on hadoop," *Proceedings of the Hadoop Summit. Santa Clara*, vol. 11, no. 3, pp. 5–9, 2011.

[33] S. M. J.W. Grzymala-Busse and Y. Yao, "An empirical comparison of rule sets induced by lers and probabilistic rough classification," in *Rough Sets and Intelligent Systems*. Springer, 2013, vol. 1, pp. 261–276.

[34] M. Lichman, "UCI machine learning repository," Irvine, CA, USA, Tech. Rep., 2013.

Authors

Angelina A. Tzacheva is a Teaching Associate Professor at the Department of Computer Science at the University of North Carolina at Charlotte. Her research interests include: data mining and knowledge discovery in databases, multimedia and distributed databases, and big data analytics.

Arunkumar Bagavathi is a Ph.D candidate of Computer Science department at the University of North Carolina at Charlotte. He received his Master's degree specialized in data mining from the University of North Carolina at Charlotte in 2016. His research interests include data mining and knowledge discovery, big data analytics, cloud computing and network analysis and representation.

Aabir Kumar Dutta is a Master's student in the department of Computer Science at the University of North Carolina at Charlotte. His research interests include Data Mining, Knowledge Discovery and Machine Learning.