In Search of Actionable Patterns of Lowest Cost - a Scalable Action Graph Method

Angelina A.Tzacheva,Arunkumar Bagavathi,Sharath C.B. Suryanarayanaprasad
Department of Computer Science
University of North Carolina at Charlotte
Charlotte, NC, 28223, USA
Email: aatzache@uncc.edu,abagavat@uncc.edu,sbagursu@uncc.edu

***Abstract*—Action Rules benefit its users to achieve their goals by extracting actionable information hidden in the large data. Undertaking such actionable recommendations incur some form of cost to users. The actionable recommendation system fails when the recommended actions are cost wise unendurable or non-profitable and uninteresting to the end user. Finding low cost actionable patterns in larger datasets is a time consuming and requires a scalable approach. In this work, we give a representation for Action Rules as graphs called Action Graphs, which uncovers undiscovered relationship between actionable patterns in the recommended action rules. Also, we define three popular graph algorithms: *Dijkstra's Shortest Path* algorithm, *Breadth First Search* algorithm and *Depth First Search* algorithm to search low cost Action Rules from Action Graphs in the distributed scenario using Spark framework. Upto our knowledge, our Depth First algorithm is the first work to be implemented using Spark framework. We apply the proposed algorithms to three datasets in transportation, medical, and business domains. Results show these domains can benefit from the discovered actionable recommendations of low cost, in time efficient way.**

***Index Terms*—Low Cost Action Rules, Action Graph, Graph Search, GraphX**

I. Introduction

Discovering surprising, unknown and useful knowledge from a massive data is the crucial task of data mining. Most of the data mining or machine learning algorithms manifest the relationship of data objects with other objects (Clustering) or classes (Classification). The Rule based learning tasks intend to circumscribe methods that identifies, learns or evolves 'rules' to store and manipulate knowledge. 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.

$$condition(s) \rightarrow result(s) \tag{1}$$

Action Rule is a rule based data mining technique that recommend possible transitions of data from one state to another, which the user can use to their advantage. For example, reducing hospital readmission in the medical domain [1]. 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 $(\theta \rightarrow \phi)$ represents desired decision action.

$$[(\Psi) \wedge (\alpha \rightarrow \beta)] \rightarrow (\theta \rightarrow \rho) \tag{2}$$

Actionable patterns from Action Rules are prone to incur certain form of cost to the user [2], [3]. Cost for actions in Action Rules include money, time, energy or human resources. Recommended actions can cause both positive(*benefits*) and negative($loses$) effects for users [4]. Thus, Action Rules recommendations system should incur low cost to the users to make them feasible actions. Existing approaches [5] [6] do not consider cost effectiveness for recommendations. In [2], the notion of cost of the Action Rules is introduced and refined. Searching for low cost Action Rules from a huge data can be very time consuming and requires a scalable and distributed approach for extracting them in a reasonable timeframe.

In this work, we construct a scalable graph, called (*Action Graph*) based on action terms of derived Action Rules [7] and their relations between other action terms. We use Spark GraphX [8] to build a scalable graph. We also introduce three most graph algorithms: *Dijkstra's Shortest Path* algorithm, *Breadth First Search* algorithm and *Depth First Search* algorithm to extract low cost Action Rules and compare their results. Although Dijkstra's Shortest path and Breadth First Search algorithms have made immense progress in parallel computing, Depth First Search is very complex to implement in parallel computing frameworks. In this work, for the first time we are defining Depth First Search for the constructed Action Graph and compare their results with other algorithms.

II. Related Works

There has been a recent trend to discover actionable petterns efficiently with distributed frameworks. For example, Tzacheva, et. al proposed MR-Random Forest algorithm [9] and Bagavathi, et. al proposed SARGS algorithm [7] 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 [2]. Recently Tzacheva, et.al [10] introduced the notion of Action Graph to examine connectivity patterns in Action Rules.

TABLE I
SAMPLE DECISION SYSTEM S

X	a	b	c	d
x_1	a_1	b_1	c_1	d_1
x_2	a_3	b_1	c_1	d_1
x_3	a_2	b_2	c_1	d_2
x_4	a_2	b_2	c_2	d_2
x_5	a_2	b_1	c_1	d_1
x_6	a_2	b_2	c_1	d_2
x_7	a_2	b_1	c_2	d_2
x_8	a_1	b_2	c_2	d_1

Ras and Tzacheva [11] introduced the notion of cost and feasibility of Action Rules as an interestingness measure. They proposed a graph based method for extracting feasible and low cost Action Rules. Ras and Tzacheva [2] 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.

Apart from Action Rules, some research has been done on extracting Actionable knowledge. For example, Karim and Rahman [12] proposed another method to extract cost effective actionable patterns for customer attribtion problem in post processing steps of Decision Tree and Naive Bayes classifiers. Cui, et.al [13] 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 [14] 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.

III. 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.

A. Decision System

Consider an information system given in Table I

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 want 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.

B. Action Rules

In this subsection, we give definitions of action terms, action rules and properties of action rules [16]

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 $(m, m_1 \to m_2)$, 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, $m_1 \neq m_2$ if $m \in A_{Fl}$. Example Action Rule for the Decision System in *Table 1* is given below: $(a, a_1 \to a_2).(b, b_1 \to b_2) \longrightarrow (d, d_1 \to d_2)$

$(a, a_1 \to a_2).(b, b_1 \to b_2) \longrightarrow (d, d_1 \to d_2)$

C. Cost of Action Rules

Typically, there is a cost associated with changing an attribute value from one class to another more desirable one. 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.

The definition of cost was introduced by Tzacheva and Ras [2] 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 $\wp_S(b, v_1 \to v_2)$ 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 $\wp_S(b, v_1 \to v_2) < \wp_S(b, v_3 \to v_4)$, 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:

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

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*.

D. Spark GraphX

Spark GraphX [8] 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(a data structure in Spark framework). 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. GraphX performs graph-specific operations as a series of distributed *map()*, *join()* and *reduce()* functions of RDDs.

IV. 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 [7]. 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.

A. Action Rules Extraction Using SARGS

The SARGS algorithm propsed in [7] uses LERS [19] and ARAS [5] methods for extracting Action Rules in a distributed fashion for larger datasets. SARGS algorithm consists of 3 modules namely: Data distribution, LERS and ARAS.

1) Data Distribution Module: The data distribution module is to evenly distribute the data based on the decision attribute. With data distribution module, SARGS gets final actionable knowledge from the distributed partitions are considered to be equal to that of the knowledge from the single data.

2) LERS module: SARGS follows distributed method of generating classification rules using LERS system. Using the information system S from Table I, LERS strategy can find all certain and possible rules describing decision attribute d in terms of attributes a, b, and c.

3) Modified ARAS module: SARGS uses all marked certain rules from the LERS module and derive 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 I are given below:

$ARs_1 : (d_1 \to d_2) = (a, \to a_2).(b, \to b_2) \longrightarrow (d, d_1 \to d_2)$

$ARs_2 : (d_1 \to d_2) = (a, \to a_2).(c, c_2) \longrightarrow (d, d_1 \to d_2)$

SARGS version of ARAS extracts all complete Action Rules. Following Action Rules are extracted from the decision system S given in Table I using SARGS method.

$AR_1(d_1 \to d_2) = (A, a_1 \to a_2).(B, \to b_2) \longrightarrow (D, d_1 \to d_2)$

$AR_2(d_1 \to d_2) = (A, a_3 \to a_2).(B, \to b_2) \longrightarrow (D, d_1 \to d_2)$

B. Action Graph

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 vetices 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.

C. Dijkstra's Shortest Path Algorithm

Algorithm 1 gives an overview on the Dijkstra's shortest path algorithm for Action Graphs for finding low cost Action Rules. In Spark GraphX, all nodes process thier properties in parallel. Ee consider following properties to each node in the graph to execute the algorithm.

- *vertexName, vertexCost*: corresponding vertex's name and cost respectively
- *d*: $(key, value)$ pair, where *key* represents the starting vertex(*s*) and *value* consists of a path followed from s to the current node and corresponding path's cost

In $SENDMSG$ function, we choose the sources that the destination is not having and send paths and costs of only sources that are not available in the destination. In $MERGEMSG$ function, for each source we select a path with minimum cost and in $RECEIVEMSG$, we receive all messages and add current node's cost and update graph properties. By following these functions, eventually paths and costs propagates to all nodes in the graph. By the end of $n/2$ iterations, all nodes would have least cost to reach from source to themselves.

Algorithm 1 Dijkstra's Shortest Path algorithm for Action Graphs

```
Require: $A_g = (A_v, A_e)$, a source vertex $u$ and cost threshold $\rho$
    $A'_g$ := $A_g$.mapVertices(v =¿ (v.vertexName,v.vertexCost,$d$)
2:  procedure SendMsg(id,srcVertex,dstVertex)
        sources := Collect sources from srcVertex.d that are not available in dstVertex.d
4:      return a dictionary with sources to dstVertex
    procedure MergeMsg(m1,m2)
6:      mergedMessage := $\emptyset$
        for $source \in m1.sources$ do
8:          if m1.source.cost < m2.source.cost then
                mergedMessage.source := m1.source
10:         else
                mergedMessage.source := m2.source
12:     return mergedMessage
    procedure ReceiveMsg(id,oldProp,newProp)
14:     for $source \in newProp$ do
            newCost := Add this.name,this.cost to newProp.source
16:         oldProp.source = newCost
        return oldProp
18: $A_g^{final}$ := $A'_g$.aggregateMessages(SendMsg, MergeMsg, ReceiveMsg)
    return all paths and costs from all vertexes
```

D. Breadth First Search Algorithm for Action Graph

Since maintaining a queue to track the traversal is complex in parallel computing engines like Spark GraphX, we follow modified strategies for *Breadth-First* and *Depth-First* searches in Action Graphs. BFS works alike Algorithm 1 with one exception. Instead of choosing a path with minimum cost in the $MERGEMSG$ function, for each source vertex we choose a path from the latest source. For example, if vertexes 1 and 2 are sending path and cost for the source node 3 to vertex 4, the $MERGEMSG$ function of vertex 4 chooses path and cost of source node 3 from vertex 2. Once this entry is updated, it cannot be altered in future but it can propagate to update its neighbor entries.

E. Depth First Search algorithm for Action Graph

For long time in the literature, parallelizing Depth First Search is one of the main concerns and several variations of parallelized DFS has been proposed [20] [21]. In this paper, we propose DFS for Action Graphs to extract low cost Action Rules as given in Algorithm 2. For the sake of Spark GraphX framework, we attach following parameters to each node for the algorithm:

- *vertexName, vertexCost*: corresponding vertex's name and cost respectively
- *neighborPath*: path followed by a node to traverse among immediate neighbors
- *l*: Similar to dictionary *d* in Algorithm 1. We also attach which node to visit next along with path and cost

Thus each vertex share their *neighborPath* in the first iteration with their neighbors(but only to specified neighbor vertex get the content). In the remaining iterations $SENDMSG$ function sends the dictionary *l*. Unlike *Dijkstra's* and *BFS*, we are not gathering paths for all possible source and destination pairs. Instead, we are setting each vertex as a source vertex and collecting a path using DFS traversal to reach all other vertexes. Thus in the $MERGEMSG$ function, we are getting updated pat and cost from neighbors. In $RECEIVEMSG$ function, we simply find nodes, from *neighborPath*, that are not visited and attach them to the path in same sequence and update the cost and next node to visit parameters.

V. Experiment and Results

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

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 [22]. The Car Evaluation Data consists of records describing a car's goodness and acceptability. The Mammographic Mass data contains records that measure severity of human cancer.

The city of Charlotte North Carolina BusinesWise data , wich 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 several features like City, StartYear, Sector, SiteType, Employees count at the site, 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 II which we use to test our algorithm.

With our datasets, we run the SARGS algorithm on each data. We collect Action Rules which meet the minimum support(s), and minimum confidence(c), threshold. If we have *n* action terms in an Action Rule, we record the cost(ρ) for each *n* Action Terms. We calculate Total Cost of each Action Rule by adding the cost of all Action Terms in the rule. The cost of each action term is provided by a domain expert who has enough knowledge about the data. For example, for the Mammography dataset, a medical doctor specifies the cost for the suggested actions. However, for our experiment purpose, we assign a random cost number to each ActionTerm. We assign the cost of *0* for stable attribute Action Terms

Algorithm 2 Depth First Search algorithm for Action Graphs

```
Require: $A_g = (A_v, A_e)$, a source vertex $u$ and cost threshold $\rho$
    $A'_g$ := $A_g$.mapVertices(v =¿ (v.vertexName, v.vertexCost, neighborPath, $l$)
 2: procedure SendMsg(id,srcVertex,dstVertex)
        return dstVertex.l
 4: procedure MergeMsg(m1,m2)
        mergedMessage := $\emptyset$
 6:     for $source \in m1.sources$ do
            if —m1.source— $>$ —m2.source— then
 8:             mergedMessage.source := m1.source
            else
10:             mergedMessage.source := m2.source
        return mergedMessage
12: procedure ReceiveMsg(id,oldProp,newProp)
        for $source \in newProp.msg$ do
14:         if newProp.source.target = id then
                newPath = newProp.source.path
16:             for $node \in oldProp.neighborPath$ do
                    if $node \notin newProp.source.path$ then
18:                     newPath := Add (node.name,node.cost)
                if $allVertices \in newPath.names$ then
20:                 target := $\emptyset$
                else
22:                 target := newPath.last.name
                oldProp.source := (newPath,target)
24:     return oldProp
    $A_g^{final}$ := $A'_g$.aggregateMessages(SendMsg, MergeMsg, ReceiveMsg)
26: return all paths and costs from all vertexes
```

TABLE II
Properties of the datasets

Property	**Car Evaluation Data**	**Mamm. Mass Data**	**Business Data**
# of instances	1728	961	22441
Attributes	7 attributes	6 attributes	17 attributes including
Decision attribute values	Class (unacc, acc, good, vgood)	Severity (0 - benign, 1-malignant)	Estimated Sales (<$2M,2-10M,10-25M,25-50M,50-100M,100-500M,>500M)
Data size	52 KB	16 KB	5.5 MB

TABLE III
EXAMPLE ACTION RULES OF LOWEST COST, MEDIUM COST AND HIGH COST FOR CAR EVALUATION DATASET

High Cost Action Rules
1) $AR_{C1} : (doors = 2) \wedge (lugBoot, big \rightarrow small) \wedge (maint, low \rightarrow vhigh) \wedge (persons, 2 \rightarrow more) \Longrightarrow (class, unacc \rightarrow acc)[Support : 4, OldConfidence : 25\%, NewConfidence : 100\%, Utility : 25\% COST : 3.523]$
Low Cost Action Rules
1) $AR_{C7} : (buying, high \rightarrow vhigh) \wedge (doors = 4) \wedge (lugBoot, big \rightarrow med) \wedge (maint, vhigh \rightarrow low) \wedge (persons = more) \wedge (safety, high \rightarrow med) \Longrightarrow (class, unacc \rightarrow acc)[Support : 1, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 1.289]$

TABLE IV
EXAMPLE ACTION RULES OF LOWEST COST, MEDIUM COST AND HIGH COST FOR MAMMOGRAPHIC MASS DATA

High Cost Action Rules
1) $AR_{M1} : (BI - RADS, 6 \rightarrow 4) \wedge (Margin, 5 \rightarrow 3) \wedge (Shape, 4 \rightarrow 2) \Longrightarrow (Severity, 1 \rightarrow 0)[Support : 13, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, ; COST : 2.688]$
Low Cost Action Rules
1) $AR_{M7} : (BI - RADS, 5 \rightarrow 2) \wedge (Density, 1 \rightarrow 3) \Longrightarrow (Severity, 1 \rightarrow 0)[Support : 6, OldConfidence : 80\%, NewConfidence : 100\%, Utility : 80\%, COST : 1.286$

TABLE V
EXAMPLE ACTION RULES OF HIGHEST COST, MEDIUM COST AND LOWEST COST FOR THE BUSINESS DATA

High Cost Action Rules
1) $AR_{B1} : (BLDGTYPE, Miscellaneous \rightarrow Office) \wedge (EMPALLSITE, 1 - 3Employees \rightarrow 10 - 24Employees) \wedge (EMPSITE, 1 - 3Employees \rightarrow 4 - 9Employees) \wedge (OWNBLDG, Y \rightarrow N) \wedge (SECTOR, Services \rightarrow WholesaleTrade) \wedge (SITETYPE, SingleSite \rightarrow Headquarters) \Longrightarrow (ESTSALES, \$2Mto\$10M \rightarrow \$10Mto\$25M)[Support : 2, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 4.069$
Low Cost Action Rules
1) $AR_{B7} : (CITY, Matthews \rightarrow Cornelius) \wedge (EMPALLSITE, 50 - 99Employees \rightarrow 100 - 249Employees) \wedge (EMPSITE, 1 - 3Employees \rightarrow 50 - 99Employees) \Longrightarrow (ESTSALES, \$2Mto\$10M \rightarrow \$10Mto\$25M)[Support : 2, OldConfidence : 100\%, NewConfidence : 100\%, Utility : 100\%, COST : 1.071$

and between 0 and 1 for flexible attribute Action Terms. In Table III, Table IV and Table V, we show samples of high cost Action Rules, and Low Cost Action Rules, for the *Car Evaluation Data*, *Mammographic Mass Data* and *Business Data* respectively. The high cost Action Rules are actions that the user would probably ignore since they are very high compared to the given cost threshold ρ. The low cost Action Rules are more suitable for users since all low cost Action Rules are below the given cost threshold ρ. In Table III, Table IV and Table V, low cost Action Rules are ones which has cost less than ρ.

These Action Rules define what actions do a company/user should employ to achieve their desired goal. For example, the rule AR_{C1} recommends that if a car company decreases the *Buying Cost* from *very high* to *low* and increases *luggage boot size* from *medium* to *big* and increases the *Maintenance Cost* from *low* to *very high* and increases *Safety Measures* from *low* to *medium* and if the *Seating Capacity* is *more than 4*, then the Car Condition may change from *Unacceptable* to *Acceptable* with the cost of 3.53. 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 comparision. We use algorithms 2 and 1 to return all low cost Action Rules ($cost < \rho$).

Table VI gives details about the number of Action Rules and basic properties of these graphs such as number of nodes, edges and number of connected component in Action Graphs and parameters set for algorithms to extract rules. It is notable that the Action Graph for the Business data is disconnected and contains 3 component in the graph. For experiment purpose, we set the minimum cost threshold ρ as 1.3 for all datasets.

In Tables VII, VIII and IX, we give our system's runtime performance comparing with non distributed and distributed versions of the Dijkstra's shortest path, Breadth First Search and Depth First Search algorithms respectively for finding low cost Action Rules. The distributed version of the Action Graph, which we implement in Apache Spark [17] using the GraphX [8] library shows faster processing times for large datasets compared to single machine implementation in Java.

VI. CONCLUSION

We studied 3 methods for searching the Action Graph for Rules of Lowest Cost. We find that BFS and DFS perform better in terms of processing time, for large datasets when incorporated into parallel frameworks like Spark GraphX. For smaller datasets, all parallel algorithms perform almost similar to serialized versions of algorithms. However, the Dijkstra's algorithm discovers higher number of Low Cost Action Rules. We implemented the DFS method in Spark GraphX, which has not been implemented before. For Action Graphs, we modified DFS algorithm to work similar to neighborhood aggregation.

TABLE VI
ANALYSIS ON ACTION GRAPHS

Property	Car Evaluation Data	Mamm. Mass Data	Business Data
Stable attributes	Doors, Buying	Age	Start Year
Required decision action	(Class) $unacc \rightarrow acc$	(Severity) $1 \rightarrow 0$	Estimated Sales \$2M – \$10M $\rightarrow$ \$10M – \$24M
No.of Action Rules	82,503	552	191,934
No.of Action Terms / Nodes	45	112	188
Minimum Support s and Confidence c	1, 10%	2, 70%	2, 60%

TABLE VII
RUNTIMES OF DIJKSTRA'S SHORTEST PATH ALGORITHM ON DIFFERENT DATASETS IN SECONDS

Dataset	Java Dijkstra's	Spark Dijkstra's
Car Evaluation data	5s	2s
Mammographic Mass data	4s	4s
Business data	16s	10s

This is great advantage over previous implementations, as it allows for DFS search in very large graphs.

TABLE VIII
RUNTIMES OF BREADTH FIRST SEARCH ALGORITHM ON DIFFERENT DATASETS IN SECONDS

Dataset	Java BFS	Spark BFS
Car Evaluation data	3s	2s
Mammographic Mass data	4s	4s
Business data	11s	7s

TABLE IX
RUNTIMES OF DEPTH FIRST SEARCH ALGORITHM ON DIFFERENT DATASETS IN SECONDS

Dataset	Java DFS	Spark DFS
Car Evaluation data	4s	5s
Mammographic Mass data	5s	7s
Business data	18s	9s

REFERENCES

[1] 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.

[2] Z. W. Raś 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.

[3] 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

[4] 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*.

[5] Z. W. Raś, 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.

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

[7] 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.

[8] 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.

[9] A. A. Tzacheva, A. Bagavathi, and P. D. Ganesan, "Mr-random forest algorithm for distributed action rules discovery," *International Journal of Data Mining & Knowledge Management Process (IJDKP)*, vol. 6, no. 5, pp. 15–30, 2016.

[10] B. A. D. A. K. Tzacheva, Angelina A., "In search of actionable patterns of lowest cost - a scalable graph method," *International Journal of Database Management Systems*.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] Z. W. Raś and A. Dardzińska, "From data to classification rules and actions," *International Journal of Intelligent Systems*, vol. 26, no. 6, pp. 572–590, 2011.

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

[19] 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.

[20] G. Chu, C. Schulte, and P. J. Stuckey, "Confidence-based work stealing in parallel constraint programming," in *International conference on principles and practice of constraint programming*. Springer, 2009, pp. 226–241.

[21] M. Naumov, A. Vrielink, and M. Garland, "Parallel depth-first search for directed acyclic graphs," in *Proceedings of the Seventh Workshop on Irregular Applications: Architectures and Algorithms*. ACM, 2017, p. 4.

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