A Generic and Computationally Efficient Automated Innovization Method for Power-Law Design Rules

COIN Report Number 2020006

Kanish Garg  
Dept. of Mechanical & Industrial Engg.  
Indian Institute of Technology Roorkee  
Roorkee, India  
kanish@me.iitr.ac.in

Anish Mukherjee  
Dept. of Mechanical & Industrial Engg.  
Indian Institute of Technology Roorkee  
Roorkee, India  
amukherjee@me.iitr.ac.in

Sukrit Mittal  
Dept. of Mechanical & Industrial Engg.  
Indian Institute of Technology Roorkee  
Roorkee, India  
smitall1@me.iitr.ac.in

Dhish Kumar Saxena  
Dept. of Mechanical & Industrial Engg.  
Indian Institute of Technology Roorkee  
Roorkee, India  
dhish.saxena@me.iitr.ac.in

Kalyanmoy Deb  
Dept. of Electrical & Computer Engg.  
Michigan State University  
East Lansing, MI, USA  
kdeb@egr.msu.edu

Abstract—Automated Innovization (AI) originally aimed to extract power-law based rules from a design optimization task without any human intervention. Existing AI methods have employed Evolutionary Algorithms (EAs) twice: first to search for Pareto-optimal (PO) solutions, and second to extract rules hidden in them. Furthermore, these methods are limited in scope, in that, they are not capable of tackling both continuous and discrete variables for either single-cluster or multi-cluster rules. In a unique departure from the state-of-the-art, this paper presents a computationally efficient, single EA-based, AI method capable of simultaneously extracting single-cluster or multi-cluster rules for both continuous and discrete variable spaces. The robustness of the proposed method is evident from its successful rule extraction tasks even with small-size datasets, where existing AI methods fail to apply. The generic scope, computational efficiency, and robustness of the proposed method are demonstrated through a number of benchmark design problems.

Index Terms—Automated Innovization, Knowledge Mining, Design Principles, Continuous Space, Discrete Space, Multi-Objective Optimization

I. INTRODUCTION

Automated Innovization (AI) refers to extraction of hidden power-law based rules from a multi-objective design optimization problem without any human intervention [2]. These rules are similar to empirically known relationships, hence, are easy to verify in real-world problems. Existing AI methods start by obtaining the PO solutions of the original design problem using an Evolutionary Multi-objective Optimization (EMO) algorithm, and subject these PO solutions to another Genetic Algorithm (GA) based procedure to extract the relationships. The first AI framework was presented in [2], paving way for further improvements in [3], [4].

The main goal of automating the power-law extraction, which was first proposed manually in [1], was to eliminate any kind of knowledge input from the designer. Some related studies have been proposed as post-optimality analysis to achieve a similar goal, however they still needed some human interaction to complete the knowledge extraction process. Examples of such techniques are $k$-means clustering for data-visualization [10], observing structure of decision variables using Kriging and self-organizing maps [9], etc. but they still leave the knowledge-extraction task to the designer.

The first AI method was developed to deal with design tasks having only continuous variables. This method was later extended to discrete search space problems, but proposed as a completely different method [4]. Despite the fact that these were the first attempts in expanding the applicability of innovization task, some challenges were inherent, including:

- The methods for dealing with continuous and discrete variables could not be integrated.
- As stated earlier, the existing AI techniques use a GA-based method, i.e., the computational expense is similar to solving the original optimization problem.
- These methods can identify rules with significance more than 70% or 80%, i.e., more than 70% or 80% solutions from the PO dataset supported the obtained innovized rule.
- The authors’ initial experimentation revealed that the existing techniques fail to reveal the relevant relationships on smaller datasets.

In an effort to overcome these challenges, here, authors propose a generic, efficient and more robust automated innovization method, that is not just capable of dealing with continuous and/or discrete variables, but is also applicable...
on smaller datasets. This paper proposes a novel automated innovization method, which in turn is capable of producing the innovized power-law based rules at a lesser computational expense.

In the remainder of this paper, Section II discusses existing innovization and knowledge-extraction techniques, and Section III explains the proposed innovization method for power-law extraction. The authors’ claims and propositions have been demonstrated on five benchmark design-optimization problems in Section IV. Section V concludes the paper while proposing some future extensions of this method.

II. RELATED STUDIES

As discussed in the earlier section, there has been some studies reported under the umbrella of automated innovization methods, while some other data-visualization techniques with a similar goal of knowledge-extraction from PO solutions (post-optimality analysis) have been reported in literature. The main difference lies in the fact if the technique requires human intervention at all in process of revealing the true relationships. In [7], the relationships between decision variables in the Pareto-optimal front is depicted using dendrograms. Pareto shells were proposed in [8] for visually analyzing the Pareto-optimal dataset in a multi-objective optimization problem. A combination of self-organizing maps and kriging was used in [9] for visualizing the structure of decision variables using non-dominated solutions. k-means clustering was used on the Pareto-optimal solutions in [10] to simplify the analysis.

However, there are some data-modelling techniques in the context of post-optimality analysis like Multivariate Adaptive Regression Splines (MARS) [5] and Kriging [6]. The structure of the hidden relationships extracted from this class of methods is generally complex, i.e., it becomes another task for the designer to decipher the physical significance of the extracted relationship. As an example, a sample rule extracted using MARS approach (from [2]) is given below:

$$\log S = 3.94 \max(0, \log V - 1.34) + 0.99 \max(0, -1.34 - \log V)$$

which fits the PO dataset properly in context of $S$ and $V$, but the physical significance of this rule is not intuitive to the designer. These relations are complex and hence, not of much practical use.

The first Automated Innovization method [2] was developed only for continuous variable spaces and single rule discovery, which was later extended to multiple rule discovery and discrete variable spaces in [3] and [4] respectively. These methods were developed to extract non-intuitive rules from the PO solutions of the design problem, however, the biggest challenge was to make them understandable by the designer. Hence, researchers came with power-law form of rules, which were equivalent to the empirical relationships that could have been obtained, as shown below,

$$\psi(\phi(x)) \equiv \prod_{j=1}^{N} \phi_j(x)^{b_j} = c,$$  \hspace{1cm} (2)

where $N$ is the number of components in the product-rule and each $\phi_j(x)$ is called a basis function, which can be either a constraint, variable, objective, or any other function of interest which can be calculated using the design variables of the optimization problem. This formulate allows the automated method to identify a relationship between any subset of the $N$ basis functions.

Apart from the applicability, there is another aspect attached to these automated methods which is the time complexity of the applied algorithm, which is $O(M^2)$ for these methods ($M$ being the size of the PO dataset). Since, these are GA-based algorithms, the total run-time of these methods when applied depends on the GA population-size ($\text{popsize}$) and the number of generations ($\text{ngen}$), where the employed clustering algorithm with complexity $O(M^2)$ is being repeated $\text{popsize} \times \text{ngen}$ times. This repetitive run-time is similar to what a GA-based Evolutionary Multi-objective Optimization Algorithm (EMOA) uses while solving some optimization problem.

In addition, this complexity of the existing AI techniques limits their application for online innovization, i.e., learning explicit relationships during the optimization run itself and using them the repair the design variables aiming towards faster convergence. [16] proposed a method to reduce the time complexity of power-law based rule extraction using log-linear modelling, followed by multi-variable linear regression. However, there were certain inherent challenges in the proposed method like it couldn’t identify any break-points in the input dataset. This was based on the assumption that the entire population follows the same rule, which resulted in poor accuracy of the rule learnt (in case the break-point existed).

III. METHODOLOGY

Observing the simplicity of the log-linear modelling discussed in Section II, authors were motivated to develop the multivariate linear-regression technique further to evaluate parameters like break-points in the dataset, significance of the relationship, etc. In log-space, equation 2 becomes:

$$\log(\psi(\phi(x))) = \sum_{j=1}^{N} b_j \log(\phi_j(x)) = \log(c).$$  \hspace{1cm} (3)

This section proposes a novel direct-clustering approach which targets the PO solutions directly and clusters them by identifying the right power-law based relationships and marking the points which follow that relationship. From equation 3, it can be observed that the task is to identify, (i) all $b_j(s)$, and (ii) the corresponding $c$-value for every cluster/rule and it’s associated significance. The proposed method is divided into two parts based on the tasks identified above, which are explained further in sections III-A and III-B respectively. The method description is given using the PO dataset from the TRUSS problem [11] as an example.

A. Finding optimal values for powers $b_j$

All existing AI techniques propose some kind of clustering procedure employed in a GA-based method, which eventually
yields the right values of \( b_j(s) \). Since the clustering procedure is repeated for every candidate solution at every generation in GA, the overall run-time and complexity of the overall innovization method is very high. The proposed method is an attempt to cluster the PO solutions directly, i.e., without involvement of GA specifically to design rule-discovery. This section will further explain the method to extract all existing combinations of \( b_j(s) \) from the PO solutions and their corresponding subset that follows the rule.

From the TRUSS problem, the obtained optimal solutions (using NSGA-II) are shown in figure 1, where figure 1(a) shows the PO front in original dimensions (\( V \) v/s \( S \)) while 1(b) represents the PO front in log-scale, i.e., \( \log V \) v/s \( \log S \). Theoretically, the power-law design rule that exists between \( S \) and \( V \) is \( S \times V = 400 \). For this \( V\)-\( S \) case, the number of basis functions \( N = 2 \), \( b_1 = b_2 = 1 \) and \( c = 400 \).

![Fig. 1: Example Plot between two basis functions.](image)

It is clearly difficult to identify any kind of irregularity in the original \( V\)-\( S \) plot, but this change of trend is clearly visible in the logarithmic space, i.e., the slope of the line changes near the bottom right corner in figure 1(b). With \( N = 2 \), the hyperplane in log-linear space is the straight line, i.e., the points lying in that straight line follow the same relationship which clearly has the maximum significance. Identifying the gradient followed by maximum number of points can lead us to the set of points lying in that cluster. Once the points are clustered, the task to extract \( b_j(s) \) is equivalent to any linear-regression task. Hence, the first step is to transform the data-points into logarithmic scale, followed by gradient evaluation which is explained below.

In order to identify the points which follow the most dominating gradient value, we evaluate the gradient (slope) of all possible combinations of data-points. For \( N \) basis functions and dataset size \( m \) (PO solutions), the total gradient values become \( \binom{m}{N} \). Let us call this set of gradient values as \( \mathcal{G} \) which is the superset of actual \( b_j \) values. The values in \( \mathcal{G} \), i.e., \( n = \lceil \sqrt{m} \rceil \) or 499,500 values, for the \( V\)-\( S \) example are shown in figure 2.

![Fig. 2: Gradient points (\( \mathcal{G} \)) versus Gradient value (in degrees).](image)

It is clearly visible in figure 2 that majority of the gradient values lie around the value of \(-45^\circ\), and at the same time, the equivalent dispersion of gradient values on both sides confirm the presence of a fuzzy dataset. Further, to identify the right value of gradient from this fuzzy set points, the concept of \textit{sqrt decomposition} is used, which divides the \( n \) points into \( \lceil \sqrt{n} \rceil \) buckets followed by calculating the number of data-points lying in each bucket. These counts are stored in another data-array, say \( \mathcal{C} \). The count in each bucket (\( \lceil \sqrt{n} \rceil = 706 \)) is plotted w.r.t. the corresponding bucket number in figure 3. It was observed in initial experimentation that using variable number of buckets yielded better results than defining a fixed number.

![Fig. 3: Plot for Number of Gradient Points per Bucket versus Bucket Number.](image)

It is evident from figure 3 that there is distinctive presence of a spike in bucket-counts, which refers to the data-points cluster with maximum significance. But, upon increasing the scale of this graph, it can be noticed that there exist some peaks as well which are highlighted in figure 4. Hence, we put two checks to take a buckets into consideration as a cluster, namely, (i) the bucket count is a local maxima, and (ii) this count is greater than \( \lceil \sqrt{m} \rceil \). This reveals all clusters that exist in the PO solutions, with even small rule significance like 5% or 10%. This is additional knowledge to the designer which is out-of-scope for the existing AI techniques.

It is interesting to note that these spikes represent the presence of relatively larger number of points lying in a particular bucket, i.e., having similar gradient values. The global maxima denotes the cluster corresponding to the rule with maximum significance, though the smaller spikes effectively represent the rules that lie outside the largest cluster data-points. Alternatively, these smaller cluster rules can be obtained in an iterative manner. The expands to the fact that if the data-points belonging to the cluster (with global maxima)
are removed, the second highest spike will become the most dominant rule in the remaining dataset which can be extracted by running the proposed method again.

The region of the smaller spikes in Figure 4 relates two different rules followed by the points at the bottom-right end of the curve in Figure 1(b). These rules, combined, are followed by approximately 6.4% of data-points from the truss PO dataset [11]. This is the additional knowledge that our method is capable to reveal.

Moving ahead, once these buckets are identified, they can be divided into sub-buckets with the same sqrt decomposition as used and subsequent peaks can be identified in those sub-buckets. This process can be repeated till a singleton set of gradient is obtained, which corresponds to the $b_j(s)$ of our desired power-law(s).

The existing AI method [2] and other earlier proposed extensions to AI are not able to identify the additional rules from the PO solutions. These are rather just left as unclustered points, which are now evaluated as an integral part of our proposed method. These rules, denoted by the region of the smaller spikes in Figure 3, form the additional knowledge content for the designer. These small clusters can ideally be further analyzed to know if any constraints are active in those regions, or if the optimizer converged properly.

**B. Finding optimal values for constant c**

Once the right set(s) of coefficients $b_j(s)$ are acquired, the task of identifying their corresponding $c$-values is relatively simpler. This part of the proposed method starts by evaluating the $c$-values for one particular set of $b_j(s)$ from the $m$ PO solutions using equation 2. For our example of $V$-$S$ rule, the obtained $c$-values from the most dominant rule are shown in figure 5.

After sorting these calculated $c$-values in ascending order, their point-wise gradients (in sequence for adjacent elements) can be evaluated. Let us store these values in a new data-array, say $G_c$. This set of gradients is then subjected to a uniform statistical tolerance of $\mu \pm \sqrt{3}\sigma$ (from [15]). The data-array $G_c$ is then partitioned at points which do not lie within the statistical tolerance limits of $\mu \pm \sqrt{3}\sigma$ resulting in the formation of multiple sub-arrays. The average of all individual sub-arrays whose size is greater than 2 are calculated and stored in another array $C_{\text{clusters}}$ which holds the clustered $c$-values as illustrated in algorithm 2. The sub-arrays containing 1 or 2 elements are not considered for calculating clustered $c$-values because we consider a cluster to be a set comprising of at least 3 elements. An illustration for partitioning of $c$-values for the formation of clusters in the truss dataset is shown in Figure 6 in which the region of the largest cluster is highlighted. If more than one set of $b_j(s)$ are identified, this process needs to be repeated for every set.

**Algorithm 1 Finding the powers $b_j$ of basis functions $\phi_j$.**

**Input:** Pareto-dataset $D$ of size $m$, basis functions $\Phi$, number of basis functions $N$

**Output:** Sets of powers $b_j(s)$ corresponding to their respective basis functions.

1: $D \leftarrow \log(D)$
2: $G_s \leftarrow \emptyset$ % $G_s$ is a 2D array
3: for all combinations of pairs $(\phi_i, \phi_j) \forall 1 \leq i,j \leq N$ do
4: $X \leftarrow \text{Select all points from } D \text{ corresponding to } \phi_i$
5: $Y \leftarrow \text{Select all points from } D \text{ corresponding to } \phi_j$
6: $G \leftarrow \emptyset$
7: for $u = 0$ to $m$ do
8: for $v = u + 1$ to $m$ do
9: $G \leftarrow G \cup \left\{ \arctan \frac{Y_v - Y_u}{X_v - X_u} \right\}$
10: end for
11: end for
12: $G_s \leftarrow G_s \cup \{G\}$
13: end for
14: $P_s \leftarrow \emptyset$
15: for each $G$ in $G_s$ do
16: while $G$ has more than 1 unique point do
17: $n \leftarrow \lfloor \sqrt{|G|} \rfloor$ % $|G|$ = count of elements in $G$
18: $\text{Maximas} \leftarrow \emptyset$
19: $\text{Counts} \leftarrow \emptyset$
20: Divide $G$ into $n$ buckets as per individual values
21: $\text{Counts} \leftarrow \text{number of points in each bucket of } G$
22: for each $C$ in $\text{Counts}$ do
23: if $C$ is local maxima and $C \geq n$ then
24: $\text{Maximas} \leftarrow \text{Maximas} \cup \{C\}$
25: end if
26: end for
27: $G \leftarrow \{G^i : \text{Counts}^i \in \text{Maximas}\}$
28: end while
29: $G_{\text{select}} \leftarrow \text{Select the single unique point in } G$
30: $P_s \leftarrow P_s \cup \left\{ \frac{1}{\tan G_{\text{select}}} \right\}$
31: end for
32: return $P_s$
Gradient of Constants

IV. RESULTS

In order to demonstrate the working of our proposed automated innovization method we have chosen five design problems used by Deb and Srinivasan in [11] to illustrate the potential of innovization. We attempt to reproduce the design principles discovered in [11] using the proposed automated innovization method and visualize them via cluster plots of ordered values of \(c\). Further, we compare our method with the existing GA-based automated innovization method described in [2], in terms of accuracy of the result obtained and run-time of both the implemented methods. The parameters used for benchmarking the existing automated innovization method used in are same as used in [2] to avoid any discrepancies in the results and run-time. The efficacy of our method is demonstrated in two parts, i.e., (i) comparison with existing approach based on accuracy of rules and run-time as shown in section IV-A, and (ii) extracting power-law relationships from smaller datasets and comparison with existing AI method as shown in section IV-B.

A. Comparison with existing approach

First we perform benchmarking for continuous variable design problems TRUSS and WELD taken from [11].

The variables tackled in the TRUSS design problem relate to total volume of the truss structure \(V\), the maximum stress in the structure \(S\), and the cross-sectional areas of the individual bars in the structure \([x_1]\) and \([x_2]\). For the this problem, our proposed method delivered more accurate results as compared to the existing AI method as shown in Table I, given that the theoretically derived rules are \(S \propto V = 400\) and \(x_1 \propto V^{-1} = 0.1118\). Although, the accuracy depreciated a little in case of \(x_2 \propto V^{-1} = 0.2236\) and \(x_2 \propto x_1^{-1} = 2\), there is significant improvement in run-time performance for all four cases.

For the WELD design problem, the variables tackled here are the thickness of the beam \(b\), the deflection \(D\) and the cost function \(C\). As established in Table II our method

Algorithm 2 Finding all constants \(c\) of given power rule.

\begin{verbatim}
Input: Pareto-dataset \(D\) of size \(m\) and set of powers \(P\nOutput: List of constants \(c\) and their respective significance.
1: \(D \leftarrow \log\(D\)
2: \(C \leftarrow \emptyset
3: \text{for } i = 0 \text{ to } m \text{ do}
4: \(c \leftarrow D_i \cdot P\)
5: \(C \leftarrow C \cup \{c\}
6: \text{end for}
7: \text{sort } C \text{ in ascending order}
8: \(G_c \leftarrow \text{Gradient of each element in } C \text{ w.r.t. its adjacent one}
9: \(\mu \leftarrow \text{Mean of set } G_c\)
10: \(\sigma \leftarrow \text{Standard Deviation of set } G_c\)
11: \(LL \leftarrow \mu - \sqrt{3}\sigma\)
12: \(UL \leftarrow \mu + \sqrt{3}\sigma\)
13: \(T \leftarrow \emptyset\)
14: \(C_{clusters} \leftarrow \emptyset\)
15: \(\text{Significance} \leftarrow \emptyset\)
16: \text{for } i = 0 \text{ to } m \text{ do}
17: \(\text{if } LL \leq G_c^{(i)} \leq UL \text{ then}
18: \(T \leftarrow T \cup \{C^{(i)}\}\)
19: \text{else}
20: \(\text{if } |T| \geq 3 \text{ then}
21: \(C_{clusters} \leftarrow C_{clusters} \cup \{c_{average}(T)\}\)
22: \(\text{Significance} \leftarrow \text{Significance} \cup \{|T|/m\}\)
23: \text{end if}
24: \(T \leftarrow \emptyset\)
25: \text{end if}
26: \text{end for}
27: \text{return } C_{clusters} \text{ and Significance}
\end{verbatim}

TABLE I: Result comparison between existing (row-1) and proposed (row-2) AI method for rules obtained from TRUSS problem. Better results are marked in bold.

<table>
<thead>
<tr>
<th>Rules</th>
<th>Variables (b_1)</th>
<th>Variables (b_2)</th>
<th>Total Significance</th>
<th>Time (in sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S^{b_1}) vs. (V^{b_2})</td>
<td>0.99988</td>
<td>400.77</td>
<td>91.8%</td>
<td>623.4</td>
</tr>
<tr>
<td>(x_1^{b_1}) vs. (V^{b_2})</td>
<td>-0.99738</td>
<td>0.1105</td>
<td>87.0%</td>
<td>629.1</td>
</tr>
<tr>
<td>(x_2^{b_1}) vs. (V^{b_2})</td>
<td>-0.99905</td>
<td>0.2236</td>
<td>88%</td>
<td>639.2</td>
</tr>
<tr>
<td>(x_2^{b_1}) vs. (x_1^{b_2})</td>
<td>-1.00010</td>
<td>0.2236</td>
<td>86.6%</td>
<td>619.3</td>
</tr>
</tbody>
</table>
delivered more accurate results for the theoretically derived rule $b + D^{-1} = 0.002195$. There is no theoretical derivation for the rule $C v/s D$, hence the accuracy comparison isn’t ideal but our claimed result has better significance than the benchmark method. Again, our run-time performance is maintained in all cases.

TABLE II: Result comparison between existing (row-1) and proposed (row-2) AI method for rules obtained from WELD problem. Better results are marked in bold.

<table>
<thead>
<tr>
<th>Rules</th>
<th>Variables</th>
<th>b₁</th>
<th>b₂</th>
<th>c</th>
<th>Total Significance</th>
<th>Time (in sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b^2$ vs. $D^2$</td>
<td>1</td>
<td>-0.99987</td>
<td>0.002197</td>
<td>94.6%</td>
<td>669.3</td>
<td></td>
</tr>
<tr>
<td>$c^2$ vs. $D^2$</td>
<td>1</td>
<td>0.90924</td>
<td>-</td>
<td>78.6%</td>
<td>673.2</td>
<td></td>
</tr>
</tbody>
</table>

Now, we target some mixed variable design problems having both continuous and discrete variables. We have chosen CLUTCH and SPRING problems from [11] for this purpose. In the CLUTCH problem, both the rules $r_a$ (outer radius of disk) versus $r_i$ (inner radius of disk) and $T$ (stopping time) versus $S$ (braking area) have no theoretical derivation. But as evident from Table III our algorithm produced results similar to the existing AI method, but in a much lesser run-time.

TABLE III: Result comparison between existing (row-1) and proposed (row-2) AI method for rules obtained from CLUTCH problem. Better results are marked in bold.

<table>
<thead>
<tr>
<th>Rules</th>
<th>Variables</th>
<th>b₁</th>
<th>b₂</th>
<th>c</th>
<th>Total Significance</th>
<th>Time (in sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t^b_1$ vs. $r^b_2$</td>
<td>1</td>
<td>-0.78885</td>
<td>-</td>
<td>100%</td>
<td>413.7</td>
<td></td>
</tr>
<tr>
<td>$t^b_1$ vs. $s^b_2$</td>
<td>1</td>
<td>0.98683</td>
<td>-</td>
<td>100%</td>
<td>0.037</td>
<td></td>
</tr>
</tbody>
</table>

TABLE IV: Result comparison between existing (row-1) and proposed (row-2) AI method for rules obtained from SPRING problem. Better results are marked in bold.

<table>
<thead>
<tr>
<th>Rules</th>
<th>Variables</th>
<th>b₁</th>
<th>b₂</th>
<th>b₃</th>
<th>Total Significance</th>
<th>Time (in sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d^b_1$ vs. $n^b_2$</td>
<td>1</td>
<td>0.33333</td>
<td>-</td>
<td>-</td>
<td>100%</td>
<td>593.4</td>
</tr>
<tr>
<td>$d^b_1$ vs. $d^b_2$</td>
<td>1</td>
<td>-0.74999</td>
<td>-</td>
<td>-</td>
<td>100%</td>
<td>0.02</td>
</tr>
<tr>
<td>$d^b_1$ vs. $d^b_2$ vs. $n^b_3$</td>
<td>1</td>
<td>-0.74999</td>
<td>-0.25000</td>
<td>-</td>
<td>100%</td>
<td>0.055</td>
</tr>
</tbody>
</table>

The SPRING design problem comprises of three variables namely the wire diameter $[d]$, mean coil diameter $[D]$ and the number of turns $[N]$. Our proposed algorithm produced better results for two rules $d * D^{-0.75} = c$ and $d * D^{-0.75} * N^{-0.25} = c$ and equivalent result for the rule $D * N^{\frac{1}{3}} = c$ as shown in Table IV in a lesser computational time.

When we ran our algorithm on the TRUSS dataset it discovered the rule $S * V^{1.00004} = c$ with 93.6% significance. The remaining 6.4% (64 points) could’ve followed an entirely different rule which hasn’t been explored in existing techniques yet. When we extracted knowledge from those points, we discovered the rules $S * V^{0.33905} = 3358.31962$ with significance 89.06% (57 points out of 64) and $S * V^{0.54529} = 1635.43580$ with significance 10.94% (7 points out of 64). As stated in Section III, this is additional knowledge revealed by the proposed AI method. Similarly, for the WELD dataset and basis functions $C$ and $D$ our proposed method had discovered the rule $C * D^{0.96795} = c$ with 79.8% significance and when we tried to extract knowledge from the remaining 20.2% (202) points our proposed AI method discovered the rules $C * D^{0.13214} = 1.08933$ with significance 56.44% (114 points out of 202) and $C * D^{0.83467} = 0.04035$ with significance 43.07% (87 points out of 202). These rules were left as unclustered points by the existing AI method [2] with no knowledge extraction.

B. Rules discovered from smaller datasets

In the presence of discrete variables (or mixed variables), PO data-points tend to have duplicates, which if removed, significantly reduce the size of the dataset. This is known to pose challenges for existing automated innovization algorithms. For instance, the algorithm in [2] fails to produce meaningful results on the GEAR, CLUTCH and SPRING problems [11] owing to insufficient size of the data set. In this section, we highlight the robustness of our algorithm on smaller datasets (without duplicates) for the above problems, and benchmark it against the existing algorithm [2].

1) GEAR problem: This gear train design problem comprises of the following variables, namely, the number of teeth on the driving gear $[T_a]$, the number of teeth on the driven gear $[T_f]$, the number of teeth on the gear attached to the driven gear $[T_b]$ and the number of teeth on the following gear $[T_f]$. The key highlights of the results are as follows.

a) $T_a$ versus $T_f$: Out of the 1000 PO solutions, only 22 are distinct for the basis functions $T_a$ and $T_f$. Our proposed method is able to discover the rule $T_a * T_f^{-0.97101} = c$ even with this reduced dataset with 3 clusters for $c$ values. The rule discovered by our algorithm is much closer to the theoretically derived rule $T_a * T_f^{-1} = c$ than the existing algorithm, as is evident in Table V.

b) $T_d$ versus $T_f$: For the basis functions $T_d$ and $T_b$, there are only 11 distinct PO solutions. Yet, our algorithm is able to discover the rule $T_d * T_b^{0.99970} = c$ which is close to the theoretical result $T_d * T_b = c$. In this case, the existing algorithm produces more accurate results.

c) $T_f$ versus $T_b$: For these basis functions there are only 27 distinct PO solutions, and the rule discovered by our algorithm matches exactly with that of the theoretical rule $T_f * T_b = c$, as shown in Table V.
2) **CLUTCH problem**: This multiple-disk Clutch brake design problem involves four variables, as defined in Section IV-A. The benchmarks between our and the existing algorithm are shown in Table VI, and the key highlights are as follows.

a) \( r_o \) versus \( r_i \): For the basis functions \( r_o \) and \( r_i \) there are only 21 distinct PO solutions. In that, our algorithm outperforms the existing algorithm, both in accuracy (refer Table III) and speed.

b) \( T \) versus \( S \): For these basis functions there are only 86 distinct PO solutions. Again, our algorithm outperforms the existing algorithm, both in accuracy (refer Table III) and speed.

3) **SPRING problem**: This spring design problem involves three variables, as defined in Section IV-A. There are only 67 distinct PO solutions, for each choice of basis functions, summarized in Table VII and discussed below.

a) \( D \) versus \( N \): here, much like the existing algorithm, our algorithm could extract the rule which exactly conforms with the theoretical result of \( D \star N^{0.33333} = c \).

b) \( d \) versus \( D \): here, our algorithm outperforms the existing algorithm, both in accuracy (theoretical result being \( d \star D^{-0.75} = c \)) and speed.

c) \( d \) versus \( D \) versus \( N \): in this case of three basis functions, unlike the existing algorithm, our algorithm extracts a rule that almost conforms with the theoretical result of \( d \star D^{-0.75} \star N^{-0.25} = c \).

V. CONCLUSIONS

In what marks a significant departure from the state-of-the-art, this paper proposes a single-EA based automated innovization method, capable of tackling problems with both continuous and discrete variables. The proposed method is computationally far more efficient than the existing methods, as demonstrated through various benchmark design problems. Hence, this method is well poised for online innovization in future, where the rules learnt from the intermediate EA generations could be utilized to guide the subsequent search. Besides the novelty, generic scope, and computational efficiency, another stand-out feature of the proposed method links to its robustness - the ability to successfully tackle small-size datasets where existing AI methods fail to apply. Another offshoot of the proposed method with potentially useful bearings is the additional knowledge about the solutions which defy the dominant rule, and conventionally discarded as unclustered points in the existing methods. In future, our endeavor would be to extend the proposed AI method for online innovization and simultaneous discovery of rules.

ACKNOWLEDGEMENT

Authors would like to acknowledge the support provided by the Government of India under SPARC project No. P-66.

REFERENCES

**Table VI: Result comparison between existing (row-1) and proposed (row-2) AI method for rules obtained from CLUTCH problem. Better results are marked in bold.**

<table>
<thead>
<tr>
<th>Rules</th>
<th>Variables</th>
<th>Total ( b_1 )</th>
<th>Significance (in sec)</th>
<th>Time (in sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_0 ) vs. ( r_i )</td>
<td>( b_1 )</td>
<td>( -0.66406 )</td>
<td>0.0%</td>
<td>4.17</td>
</tr>
<tr>
<td></td>
<td>( b_2 )</td>
<td>( -0.78986 )</td>
<td>66.67%</td>
<td>0.003</td>
</tr>
<tr>
<td>( T ) vs. ( S )</td>
<td>( b_1 )</td>
<td>( 0.66745 )</td>
<td>0.0%</td>
<td>12.51</td>
</tr>
<tr>
<td></td>
<td>( b_2 )</td>
<td>( 0.98869 )</td>
<td>89.53%</td>
<td>0.024</td>
</tr>
</tbody>
</table>


Fig. 13: Cluster Plot for $d * D^{-0.74999} = c$.

Fig. 14: Cluster Plot for $d * D^{-0.74999} * N^{-0.25000} = c$.

TABLE VII: Result comparison between existing (row-1) and proposed (row-2) AI method for rules obtained from SPRING problem. Better results are marked in bold.

<table>
<thead>
<tr>
<th>Rules</th>
<th>Variables</th>
<th>$b_1$</th>
<th>$b_2$</th>
<th>$b_3$</th>
<th>Total Significance</th>
<th>Time (in sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D^b_1$ vs. $N^b_2$</td>
<td>1</td>
<td>0.33333</td>
<td>-</td>
<td>-</td>
<td>94.03%</td>
<td>12.132</td>
</tr>
<tr>
<td>$D^b_1$ vs. $D^b_2$</td>
<td>1</td>
<td>0.33333</td>
<td>-</td>
<td>-</td>
<td>100%</td>
<td>0.018</td>
</tr>
<tr>
<td>$d^b_0$ vs. $D^b_2$</td>
<td>1</td>
<td>0.09267</td>
<td>-</td>
<td>-</td>
<td>62.69%</td>
<td>12.01</td>
</tr>
<tr>
<td>$d^b_1$ vs. $D^b_2$ vs. $N^b_3$</td>
<td>1</td>
<td>-0.74999</td>
<td>-</td>
<td>-</td>
<td>74.63%</td>
<td>0.013</td>
</tr>
<tr>
<td>$d^b_1$ vs. $D^b_2$ vs. $N^b_3$</td>
<td>1</td>
<td>0.82658</td>
<td>-0.33333</td>
<td>-</td>
<td>94.03%</td>
<td>13.35</td>
</tr>
<tr>
<td>$d^b_1$ vs. $D^b_2$ vs. $N^b_3$</td>
<td>1</td>
<td>-0.74999</td>
<td>-0.25000</td>
<td>-</td>
<td>95.52%</td>
<td>0.024</td>
</tr>
</tbody>
</table>