Error Correction Using Baran: An Experimental Study

AMIRALI MADANI, Carleton University, Canada

Data cleaning is one of the most fundamental tasks for modern database systems. To resolve errors in database tuples, they must be first detected and then corrected. Proposed by Mahdavi and Abedjan, Baran is a state-of-the-art error correction system. When compared with other state-of-art error correction systems, Baran was shown to be highly effective in resolving erroneous values. However, we believe several important research questions were left unanswered in the original paper of Baran, and we investigate these questions in this paper. As an error correction system, Baran is reliant on sampling a limited number of dataset tuples to be corrected by an expert. As the main contribution of our project, we propose two alternative approaches for tuple sampling in Baran, one of which results in promising performance by producing better values for precision, recall, and F_1 for most datasets. Moreover, we propose a new model, namely the spelling-based model, and we integrate it into Baran. This new model also shows competitive performance when compared with the original implementation of Baran. However, our proposed techniques fail to statistically outperform Baran (as determined by the Mann-Whitney U test).

ACM Reference Format:
Amirali Madani. 2022. Error Correction Using Baran: An Experimental Study. *ACM Trans. Graph.* 37, 4, Article 111 (August 2022), 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

For modern databases, data cleaning is one of the most fundamental tasks [11]. In terms of errors in the tuples of a database, two main problems are encountered: error detection and error correction. Recently, Mahdavi and Abedjan investigated the problem of data cleaning by proposing a new error correction system named Baran [19]. Baran follows the "configuration-free paradigm" in which the user is only required to correct a limited number of tuples from the dataset. Mahdavi and Abedjan note that, unlike Baran, in the traditional data correction systems the users must have some knowledge of the dataset and the error correction system as they are required to predefine a set of rules for data cleaning.

Baran uses three error correctors, each of which is designed to address different types of errors. The value-based correction model is designed to address general errors such as common spelling and syntactic mistakes. Value-based errors occur regardless of the specific domain of the dataset, and Baran provides the option to pretrain the value-based corrector on the revision history of Wikipedia such that the performance of Baran in dealing with these general errors is further optimized. The vicinity-based correction model is designed to resolve errors in each tuple of a dataset tuple by considering the column-wise relationships inside that tuple. Finally, the domain-based correction model is used to fix errors in a data value of a dataset (with a specific row and attribute) using the clean values of other rows for the same attribute. Each of these error correctors produces many correction candidates for a given error, and Baran picks one final correction from this set of candidates using a binary classification approach. This aggregation of correction models is executed by training a binary classifier for each attribute (column) of the dataset.

When compared with other state-of-the-art approaches in [19], Baran was shown to be highly competitive with respect to effectiveness, efficiency, and human involvement. However, we believe several aspects of Baran are understudied and can be further explored and we try to cover these aspects as our course project.

2 ACHIEVEMENTS AND CONTRIBUTIONS

The list of our achievements and contributions in this project is as follows:

- As the first step of our project, we got Baran running and reproduced its reported results in [19]. The source code of Baran is publicly available [1].
- Our first contribution is a brief literature review of some of the past and recently proposed approaches for error detection and correction systems (as the related work).
- Our second contribution is proposing two new tuple sampling techniques for Baran. In the current implementation of Baran (and as discussed in [19]), tuples maximizing some criteria (such as the number of unresolved errors) are selected to be corrected by the user. Baran then uses these corrected tuples for learning purposes. However, Mahdavi and Abedjan only compared their proposed technique with random sampling. The tuple sampling formula of Baran favors errors that are both common and belong to the columns with higher numbers of unresolved errors. Therefore, we first study whether always choosing the tuple with the maximum score is the best approach. Furthermore, we model the problem of maximizing the number of unresolved errors and the number of frequent errors (with respect to the entire database) in each tuple as a bi-objective optimization problem such that tuples containing Pareto-optimal values for these numbers are sampled. Experimental results show that the second tuple sampling technique produces promising results.
- We explore some techniques to generalize the value-based model of Baran as it currently might lose performance under some circumstances. An example of this is provided in [19] where "Holland" is replaced by "Netherlands". However, this does not cover cases where the word "Holland" itself is misspelled (for example "Hlland"). The authors mention this issue in the same section by stating that their corrector needs to see the exact value "Holland" to be functional. Therefore,

Author's address: Amirali Madani, amiralimadani@cmail.carleton.ca, Carleton University , Ottawa, Ontario, Canada.

0730-0301/2022/8-ART111 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] https://github.com/BigDaMa/raha

the use of a value-based corrector might be impractical as it requires every possible error (every possible misspelled version of "Holland") to be explicitly defined. Therefore, we propose a new error correction system, namely the spelling-based model, and implement it in Baran. Experimental results show that the proposed approach achieves competitive performance.
- Throughout this project, we aim to solidify our empirical evidence using statistical significance tests. More specifically, to prove whether changing some components improves or worsens the performance of Baran, we use 10 independent runs and use the two-tailed Mann-Whitney U test [22] with a confidence level of 95% to check whether the difference between the two versions is *statistically significant*.

The remainder of this paper is organized as follows. Section 3 provides information regarding some state-of-the-art error correction techniques, some of which were compared with Baran in [19]. Section 4 provides some background information for understanding the context of this paper and our contributions. Section 5.1 provides information about our experimental setup, such as the used datasets, performance measures, and control parameters. Section 5.2.2, Section 5.2.4, and Section 5.3 discuss the first, the second, and the third experiments respectively. Finally, Section 6 provides the concluding remarks of this paper along with some potential avenues of future work.

3 RELATED WORK

Error correction systems can be classified into many categories. Some systems are based on integrity rules. For example, Chu *et al.* [6] proposed a novel framework (Holistic) for data cleaning. Holistic enables the users to list out a set of quality requirements using denial constraints. This specification of constraints is achieved using a language. This language is highly expressive; for example, it allows the users to set out rules involving numerical values. Moreover, Dallachiesa *et al.* [9] proposed NADEEF for error correction. NADEEF provides an interface in which users can specify both static and dynamic data quality rules.

However, not all data cleaning systems are based on pre-defined rules. Some data corrections revolve around statistical likelihoods; for example, Yakout *et al.* [33] proposed an error correction framework termed SCalable Automatic REpairing (SCARE). Using the statistical machine learning model, SCARE maximizes the likelihood of the provided dirty data given the underlying distribution. Wang *et al.* [32] proposed SampleClean in which the users are only required to clean a sample of data. SampleClean uses the cleaned samples (by the users) to correctly answer queries. Chu *et al.* [7] proposed KATARA, a data cleaning framework based on crowdsourcing and knowledge bases. Given a knowledge base and a crowd, KATARA is able to first identify the errors in a dirty database, then recommend possible fixes to each erroneous value.

In [1], the researchers investigated the problem of introducing errors into clean databases as a benchmark for data-cleaning algorithms. They intended to provide users with as much control over the error-generating process as possible, while at the same time developing solutions that scale to large databases. In the research paper, the authors showed that the problem of error generation is quite challenging, and it is in fact NP-complete. The authors developed a greedy algorithm that is correct and efficient while sacrificing completeness, but still proved to be successful under reasonable assumptions. Several optimizations are needed to scale the algorithm to millions of tuples, including a symmetry property of data quality constraints. The main technical contribution of the paper deals with the trade-off between control and scalability.

Geerts *et al.* [13] came up with a uniform framework known as LLUNATIC, a *chase*-based algorithm, which is the first system that runs over a DBMS for solving data-cleaning problems (or data-repairing problems) that require multiple types of constraints and different ways to select preferred values. In their approach, the violation is not *chased* at the tuple level, but at the class level, so they introduce a notion of a cost manager as a plug-in that selects which repairs can be kept versus those to be discarded. This paper provides a language for expressing constraints based on equality generating dependencies (EGDs) which generalizes many of the constraints found in the literature. As a result, they could express inter-table constraints and improve the expression of dependencies, leading to several benefits in terms of scalability. Furthermore, they provided experimental results confirming its high scalability compared with previous main memory implementations, as well as its superior quality in computing repairs.

Krishnan *et al.* [17] investigated the iterative cleaning process during statistical model training, a rapidly growing field of data analytics. The authors proposed ActiveClean, a method for cleaning statistical modeling problems using progressive and iterative approaches while preserving convergence guarantees. In ActiveClean, convex loss models are supported, which prioritizes cleaning the records that could affect the results of the model. In addition, they developed an approximation to estimate the theoretical optimum and developed a theoretically optimal sampling distribution. ActiveClean was evaluated on five real-world datasets including UCI Adult, UCI EEG, MNIST, IMDB, and Dollars For Docs, all of which had real and synthetic corruption. ActiveClean produced an 80% true-positive rate with less data compared with the alternatives in a fraud prediction example. In their experiments, they demonstrated that their models could be improved by 2.5x for the same amount of data cleaned using the optimizations they presented. In addition, ActiveClean produced more accurate models when a cleaning budget was fixed and when the datasets were contaminated in real life.

The authors in [26] developed a novel framework for combining the benefits of logical data cleaning (which takes into account both an error and a correct repair) with quantitative data cleaning (which employs the distributional properties of data to find the best repair). An initial rendition of the framework uses metric functional dependencies that generalize functional dependencies (FDs) to identify and measure statistical distortions between repair and desired distributions, based on Earth Mover's Distance. By utilizing statistical distortion during cleaning, they proposed a new strategy based on constraint-based cleaning that ensures the (minimal) repair is of high quality. According to their experimental evaluation, their techniques achieve a significantly lower level of statistical distortion than existing repair procedures, while maintaining similar levels of

efficiency. There is no doubt that their methodology can be applied to other metrics of statistical distortion and other constraints as well. To provide more robust cleaning solutions, both logical and statistical approaches can be combined - an important direction for the data cleansing industry.

Volkovs *et al.* [31] proposed an application-oriented continuous data cleaning framework to handle dynamic data and constraints. In this approach, both the data and its semantics can evolve, and a suggestion is made based on the evidence gathered so far. Additionally, the approach they developed considers not only the data and constraints as evidence but also the past repairs selected and applied by the user (user-repair preferences). In one of their studies, they presented a method for predicting the type of repair required such as data, FD, or a hybrid of the two to resolve an inconsistency, and were able to predict repair preferences based on repeated iterations, resulting in more relevant repairs. Data quality improvement can be achieved by experimenting with a large number of statistics and showing which of these statistics is predictive of finding the right class of repair to improve data quality. Empirically, they identified the statistics that are sensitive to classifying tasks and found that they can influence individual repair decisions through the data and constraint properties. The authors evaluated their techniques in terms of performance and quality, emphasizing adaptability of their framework, as well as the quality of repair and accuracy of classification.

Using two configuration-free systems, Raha and Baran, Mahdavi and Abedjan [20] developed an end-to-end pipeline for detecting and repairing data errors. Through effective feature representation, label propagation, and transfer learning methods, their framework demonstrated how user supervision could be reduced to a negligible amount of example corrections. They designed each cleaning step based on the same principle even though each detection and correction step faces different challenges. Each system utilizes a set of base detectors and correctors which are generated by an automatically generated set of algorithms and learned to combine using a few user labels By analyzing a small set of 20 user-annotated tuples, data quality issues can be effectively identified and corrected inside a dataset. Furthermore, both systems take advantage of the prior cleaning tasks a dataset has undergone. The two systems can use transfer learning to optimize the data cleaning task based on the error detection runtime and the effectiveness of the error correction.

Irrespective of the underlying mechanism, data cleaning is a fundamental task in data management and has been extensively studied in the literature. Therefore, any performance improvement over the existing state-of-the-art data cleaning frameworks would be greatly valuable to the data science community. With this in mind, our achievements and contributions in this project discussed in th next sections.

4 BACKGROUND

This section covers some background information.

4.1 Evolution and Tournament Selection

Used frequently in the fields of computer science, evolutionary computation, and operations research, a genetic algorithm (GA) [23] solves mathematical optimization problems (defined extensively in Section 4.2). GAs are based on natural selection, in the sense that they keep track of a *population* of *chromosomes* which represent abstract solutions to the underlying problem (generally an optimization problem).

As mentioned above, a GA keeps track of a population of chromosomes, and it evolves this population over different generations (iterations). At each generation, new offspring are created from the current population, which will form the population for the next generation. To create offspring, GAs use different operators such as mutation (for modifying a single existing solution and producing a new one) and crossover (for combining two existing solutions and producing a new one).

More importantly, the aforementioned offspring generation operators need solutions to modify or use for crossover. To select solutions from the current population, one might simply pick the best solutions (with respect to a *fitness* measure). However, one of the fundamental arguments of GAs is that such elitist approaches do not always work [23]. Since GAs are based on natural selection, the argument is that simply picking the best solutions will lead to *premature convergence*, or a *lack of diversity* in which all individuals (solutions) are similar. Therefore, GAs use many selection mechanisms that do not always pick the best possible solution but still favor solutions with better fitness values. Two of these approaches are defined below:

Definition 1. *Let $T = \{t_1, \ldots, t_n\}$ be a set of solutions, and let $f : T \mapsto \mathbb{R}$ denote a fitness function defined over these individuals. Given an integer $n_t > 0$ (the tournament size), a tournament selection method selects n_t solutions at random from T and returns the solution that has the best value with reference to f.*

In addition to tournament selection, the roulette wheel selection [23; 25] is also one of the commonly used selection operators of GAs:

Definition 2. *Let $T = \{t_1, \ldots, t_n\}$ be a set of solutions, and let $f : T \mapsto \mathbb{R}$ denote a fitness function defined over these individuals. A roulette wheel selection method selects one solution from T in a way such that solutions with better fitness values have higher chances of being selected.*

For example, if larger fitness values are preferred and we have two solutions t_1 and t_2 with respective probabilities 10 and 40, then t_1 and t_2 are selected with probabilities 0.2 and 0.8 respectively.

In Section 5.2.2, we study the problem of applying similar ideas to the tuple sampling component of Baran. More specifically, instead of picking the overall best tuple, one can give the less *elite* solutions more opportunities.

4.2 Pareto-Optimality and Non-Dominated Solutions

Optimization is the task of obtaining the best value according to a set of criteria and mathematical functions [2]. Many optimization problems consist of more than one objective [8][21] which are usually conflicting with each other. Also referred to as multi-objective optimization problems, such problems are concerned with obtaining a set of *optimal trade-offs* between two or three problems or

objectives [30]. Zitzler [34] defined the goals of a multi-objective optimization algorithm as follows:

(1) finding solutions which are close to the true solutions,
(2) finding solutions that are evenly spread out

Multi-objective optimization has many applications in real-life scenarios. For instance, some researchers [35] modeled the (conflicting) objectives of space mission cost and time as a multi-objective optimization problem. Multi-objective optimization has been used to maximize the correction and the duration of market predictions [15; 28; 30]. In the field of mechanics, multi-objective optimization has been used to minimize the conflicting objectives of equipment cost and energy consumption [10; 16; 29].
More formally, a multi-objective optimization problem is formulated as:

$$OPT \quad \mathbf{F} = (\mathbf{f}_1, \ldots, \mathbf{f}_m) \tag{1}$$

where m is the number of objectives. Note that OPT in the equation above can either be equal to MAX or MIN depending on the type of the objectives in a given problem. For example, if the goal is to minimize a *cost* function, then the objectives are to be minimized and smaller objectives values are preferred. However, if the goal is to maximize a *profit* function, then the objectives are to be maximize and larger objectives values are preferred. In our specific case (described in greater detail in Section 5.2.4), we model the tuple sampling procedure as a multi-objective maximization problem which aims to select a tuple from the dirty dataset that provides an optimal trade-off among two conflicting objectives.
The following definitions are fundamental in our second experiment [8]:

Definition 3. ***Pareto-dominance:*** *For a multi-objective* ***maximization*** *problem with $m > 1$ objectives, objective vector $\mathbf{F}_1$ is said to dominate (Pareto-dominate) decision vector $\mathbf{F}_2$ (denoted by $\mathbf{F}_2 \prec \mathbf{F}_1$) if and only if $\boldsymbol{f}_{1i} \geq \boldsymbol{f}_{2i}$ $\forall i \in [1, m]$ and $\exists i \in [1, m]$ such that $\boldsymbol{f}_{1i} > \boldsymbol{f}_{2i}$.*

In other words, a solution to the multi-objective optimization problem dominates another solution if all of its objective values are at least as good as the other solution, and there exists at least one objective for which it has a strictly better value. Furthermore, we have the following definition:

Definition 4. ***Pareto-optimal set (non-dominated set)*** *A set of objective vectors $\mathcal{F} = \{\boldsymbol{F}_1, \ldots, \boldsymbol{F}_{|\mathcal{F}|}\}$ is said to be a non-dominated set of solutions if and only if $\forall i, j \in \{1, \ldots, |\mathcal{F}|\} i \neq j : \quad \boldsymbol{F}_i \not\prec \boldsymbol{F}_j$.*

In other words, set $\mathcal{F}$ is a non-dominated (or Pareto-optimal) set of solutions if none of its solutions dominates another, or no solution is dominated by another solution from the set.

4.3 Edit Distance and BK-Trees

Edit distance is one of the significant problems in various fields of computer science, such as natural language processing, computational biology, and so on. Edit distance is concerned with finding string similarity [3]. More formally, the edit distance between two strings is the minimum number of edit operations, namely insertions, deletions, and substitutions for transforming one string into the other. Insertions is the act of placing a character at a specific position, deletion is done by taking out or removing a character, and finally substitution is a combination of insertion and deletion [14][3].
The BK-trees algorithm, named after their founders, refers to Burkhard-Keller trees [4]. By utilizing the triangular inequality and the edit Distance, the algorithm improves the time complexity of string matching operations. More specifically, given a query word and an existing dictionary of words, a BK-tree returns the list of all words of the dictionary that are within an edit distance of n_d from the query word. BK-trees are quite efficient in implementing the auto-complete feature in many software and applications [5]. For constructing a BK-tree, an arbitrary word is selected as the root node. Each word in the dictionary is stored as a node, which results in the total number of nodes being equal to the number of words being inserted into the dictionary. There are three criteria that govern the edit distance integer value of the edge that connects the nodes. The first criterion states that If the distance between X and Y is zero then X is equal to Y. The second criterion states that the distance from X to Y is equal to the distance between Y to X. Finally, the third criterion is based on the aforementioned concept of Triangle Inequality.

Algorithm 1 Tuple sampling using tournament selection (the first experiment)

1: **procedure** Tuple Sampling Using Tournament Selection
2: **Input and notation:** n_t (the tournament size), E_j (the set of all errors in column j), E'_j (the set of all unresolved errors in column j), count $\left(d[i, j] \mid E'_j\right)$ (the number of unfixed data errors in column j whose value is exactly $d[i, j]$), d the dataset
3: **Output:** A sampled tuple t^* to be corrected by the user.
4: **for** each tuple $t \in d$ **do**
5: Calculate and store the score of tuple t ($\mathbf{S}(t)$) using Eq. (2)
6: **end for**
7: *Candidates* ← Select n_t tuples from d uniformly at random
8: t^* ← the tuple from *Candidates* that maximizes the sampling score (as calculated in step 5).
9: **return** t^*
10: **end procedure**

5 EMPIRICAL PROCESS

This section extensively discusses the empirical process followed to assess the efficiency of our proposed modifications. The remainder of this section is organized as follows. Section 5.1 provides information about our experimental setup, such as the used datasets, performance measures, and control parameters. Section 5.2.2, Section 5.2.4, and Section 5.3 discuss the first, the second, and the third experiments respectively.

5.1 Experimental Setup

In this paper, we conducted three major sets of experiments, the details of which can be found in Section 5.2 to Section 5.3. However and for the sake of readability, this section provides information about our experimental setup.

Algorithm 2 Edit Distance Calculation

procedure LEVENSHTEIN DISTANCE
▷ This algorithm calculate the minimum number of modifications (insertions, deletions, and substitutions) needed to convert one string (s_1) to another (s_2
Input and notation: s_1 (the first string), s_2 (the second string)
Output: ED (the edit distance between s_1 and s_2
$n \leftarrow \text{Length}(s_1)$
$m \leftarrow \text{Length}(s_2)$
$A[0 \dots n][0 \dots m] \leftarrow 0$ ▷ This procedure calculates the edit distance between s_1 and s_2 in a dynamic programming-based and bottom-up fashion. A is an $(n+1) \times (m+1)$ matrix containing the partial solutions to the smaller sub-problems
for $i \in \{0, \dots, n\}$ **do**
for $j \in \{0, \dots, m\}$ **do**
if $i = 0$ **then**
$A[i][j] \leftarrow j$ ▷ if the length of the one of the strings is equal to zero, delete the entirety of the other string to convert them into equivalent strings. The edit distance in this case will be the size of the other string.
else if $j = 0$ **then**
$A[i][j] \leftarrow i$
else if $s_1[i-1] = s_2[j-1]$ **then**
$A[i][j] = A[i-1][j-1]$ ▷ If both strings have identical last characters no changes are needed.
else
$A[i][j] = 1 + \min\Big(A[i][j-1], A[i-1][j], A[i-1][j-1]\Big)$ ▷ If the strings do not have equal last characters, then one unit of *change cost* is incurred.
end if
end for
end for
return $A[n][m]$
end procedure

5.1.1 Performance Measures. We used the same performance measures as the original paper of Baran [19], namely precision, recall, and F_1. Precision is the fraction of correctly fixed errors to all fixed errors. Recall is the fraction of correctly fixed errors to all data errors. The F_1 score is the harmonic mean of precision and recall.

5.1.2 Parameters. For the sake of comparability with the original paper of Baran [19], the labeling budget of Baran was set to $\theta_{\text{Labels}} = 20$ (as used in [19]). In the first experiment, the tournament size was set to $n_t = 3$ as recommended in the relevant literature [23]. In the third experiment, the edit distance tolerance was set to $n_d = 3$. Similar to the original paper of Baran, we used AdaBoost [12] as the machine learning classifier in our experiments.
All algorithms (Baran itself and our experiments) were run 10 times on each dataset, the reported values (in Table 1 to Table 3) for

Algorithm 3 Tuple sampling using non-dominated solutions (the second experiment)

procedure TUPLE SAMPLING USING NON-DOMINATED SOLUTIONS
Input and notation: E_j (the set of all errors in column j), E'_j (the set of all unresolved errors in column j), count $\Big(d[i,j] \mid E'_j\Big)$ (the number of unfixed data errors in column j whose value is exactly $d[i,j]$), d the dataset
Output: A sampled tuple t^* to be corrected by the user.
for each tuple $t \in d$ **do**
Calculate and store the individual scores $\mathbf{f}_1$ and $\mathbf{f}_2$ using Eq. (5) and Eq. (5) respectively
Using the individual scores $\mathbf{f}_1$ and $\mathbf{f}_2$ calculated in the previous step, form a *score tuple* $\mathbf{F}_i$ ▷ Assume i denotes the index of tuple t in d
end for
Form a set of score tuples $\mathcal{F} = \{\mathbf{F}_1, \dots, \mathbf{F}_{|\mathcal{F}|}\}$
$Candidates \leftarrow$ Using Algorithm 4, retrieve the set of non-dominated score tuples
$t^* \leftarrow$ a randomly selected tuple from $Candidates$.
return t^*
end procedure

Algorithm 4 Selecting non-dominated solutions from a set of score tuples (the second experiment)

procedure SELECTING NON-DOMINATED SOLUTIONS FROM A SET OF SCORE TUPLES
Input and notation: A set of score tuples $\mathcal{F} = \{\mathbf{F}_1 \dots, \mathbf{F}_{|\mathcal{F}|}\}$ where each tuple $\mathbf{F}_j$ is of the form described in Eq. (4)
Output:A set of non-dominated score tuples $Candidates$
$Dominated[1 \dots |\mathcal{F}|] \leftarrow$ FALSE ▷ A list that keeps track of the dominated tuples (See Definition 3)
$Non\_Dominated \leftarrow \emptyset$
for $i \in \{1, \dots, |\mathcal{F}|\}$ **do**
for $j \in \{1, \dots, |\mathcal{F}|\}$ **do**
if $i \neq j$ and $Dominated[j]$ = FALSE **then**
if $\mathbf{F}_j \prec \mathbf{F}_i$ **then** ▷ See Definition 3
$Dominated[j] \leftarrow$ TRUE
end if
end if
end for
end for
for $i \in \{1, \dots, |\mathcal{F}|\}$ **do**
if $Dominated[i]$ = FALSE **then**
$Non\_Dominated \leftarrow Non\_Dominated \cup \{\mathbf{F}_i\}$
end if
end for
return $Non\_Dominated$
end procedure

Algorithm 5 The proposed spelling-based model (the third experiment)

1: **procedure** SPELLING-BASED MODEL
2: **Input and notation:** A BK-tree ($\mathcal{T}_c$), an erroneous string (s), n_d (the edit distance tolerance value)
3: **Output:** A set of correction candidates for s along with their probabilities *Candidates*
4: *Candidates* ← Using $\mathcal{T}_c$, n_d, and the procedure described in [4], obtain the list of all strings from $\mathcal{T}_c$ that have edit distance at most n_d from s.
5: **for** each string $s' \in$ *Candidates* **do**
6: Calculate the similarity of s' to s using Eq. (7) and update *Candidates* with its corresponding similarity (probability) value
7: **end for**
8: **return** *Candidates*
9: **end procedure**

precision, recall, and F_1 were averaged over these 10 runs to ensure fair comparisons.

5.1.3 Statistical Significance. All algorithms were run 10 times for each test instance. The algorithms were compared using the two-tailed Mann-Whitney U test [22] with a confidence level of 95%. More precisely, between the 10 values (corresponding to 10 independent runs of each algorithm) for precision, recall, and F_1 a two-tailed Mann-Whitney U test was conducted. If the resulting p-value of this test was less than 0.05, the difference between the two sets was deemed *statistically significant*. The results of these statistical tests are provided in the last columns of Table 1 to Table 3. If the difference between Baran and our experiments was statistically significant (p-value < 0.05), the value "Yes" was placed in the last columns of these tables. Otherwise, the value "No" was used to note that the difference between Baran and our experiments was not statistically significant.

5.1.4 Implementation Details. We used the source code of Baran for running our experiments. This source code is made publicly available by the authors [2] and it is implemented in Python. For our experiments, we identified the appropriate methods and applied our desired modifications.

5.1.5 Datasets. Due to the lack of time, we only used four datasets for evaluating different algorithms. More specifically, we used the following datasets:

- **Hospital** [27]
- **Flights** [18]
- **Beers** [19]
- **Rayyan** [24]

Hospital and Flights have a high amount of data redundancy, while the the opposite holds for the other two datasets (Beers and Rayyan) with low redundancy.

[2] https://github.com/BigDaMa/raha

5.2 Tuple Sampling Techniques

This section describes our first set of experiments. More specifically, we study the idea of giving the *less elite* tuples more opportunities to be selected.

5.2.1 The Problem. As briefly discussed in Section 1, Baran is an error correction system that uses external information from the user for cleaning data errors. More specifically, in each data cleaning *iteration*, Baran samples a tuple from the dirty dataset which it asks the external user to correct. Baran then uses this external information (corrected tuples) for updating its knowledge base for fixing other errors in the dirty dataset.
Using such an example-based error correction mechanism, the tuple sampling technique is fundamental to Baran. Note that Baran proposes to minimize the user intervention in the cleaning process, so the number of sampled tuples must be minimal [19]. Therefore, the sampled tuples must be chosen carefully so that a lot of information is extracted from a minimal number of tuples.
For sampling tuples, Baran first calculates the following for each tuple in the dirty dataset:

$$\mathbf{S}(t) = \prod_{d[i,j] \in t \cap E'_j} \exp\left(\frac{\left|E'_j\right|}{\left|E_j\right|}\right) \exp\left(\frac{\text{count}\left(d[i,j] \mid E'_j\right)}{\left|E'_j\right|}\right) \quad (2)$$

where E_j is the set of all errors in column j, E'_j is the set of all unresolved errors in column j, $\text{count}\left(d[i,j] \mid E'_j\right)$ is the number of unfixed data errors in column j whose value is exactly $d[i,j]$) and finally d is the dataset. After calculating the value in Eq. (2) for each tuple in d, Baran selects the tuple that has the maximum score with reference to Eq. (2). More formally, Baran uses the following equation to select the sampled tuple t^* at each iteration:

$$t^* = \underset{t \in d}{\text{argmax}} \prod_{d[i,j] \in t \cap E'_j} \exp\left(\frac{\left|E'_j\right|}{\left|E_j\right|}\right) \exp\left(\frac{\text{count}\left(d[i,j] \mid E'_j\right)}{\left|E'_j\right|}\right) \quad (3)$$

The idea proposed in Eq. (3) is indeed interesting. A tuple selected by such an approach will (at least hypothetically) prioritize tuples that have:

- many unresolved data errors
- data errors that exist in columns with relatively more data errors
- erroneous values that appear frequently in their respective columns of the dataset.

Although interesting, the proposed tuple sampling technique in [19] (and Eq. (3)) might not always select the best tuple. More specifically, in the remainder of this section we investigate the following research questions:

- $\mathcal{RQ}1$: Is always selecting the tuple with the highest score the best approach? Do all *useful* tuples always maximize Eq. (2)?
- $\mathcal{RQ}2$: What happens when a *useful* tuple does not maximize all of the desired criteria (such as the number of unresolved errors and the number of frequent unresolved errors)? Will it still be selected given the fact that Eq. (2) is a product (multiplication) of many criteria?

We believe that these questions are extremely important for achieving competitive data cleaning performance and should not have been left unanswered. In fact, the authors of Baran [19] only compared their sampling mechanism with random sampling which does not provide enough information as to whether using the proposed approach always produces the best results.

5.2.2 The First Experiment (Tournament Selection). This section describes our first set of experiments for investigating the tuple sampling component of Baran. More specifically, we aim to answer $\mathcal{RQ}1$ and study whether always selecting the tuple with the highest score is the best approach.

To investigate $\mathcal{RQ}1$, we use some ideas from genetic algorithms (GAs) [23]. As extensively discussed in Section 4.1, GAs select solutions (chromosomes) by favoring the ones with higher scores, but not necessarily ignoring the ones with lower scores. GAs use this approach to promote the diversity of solutions while still giving elite solutions more opportunities. More specifically, we adapt the tournament selection method (Definition 1) and implement it in Baran to check whether the performance is affected.

Algorithm 1 lists our proposed modifications for our first experiment. Note that we do not alter the original tuple scoring equation of Baran for the first experiment. Instead, we only change the method of selecting tuples according to their scores. As seen in Algorithm 1, we first calculate the score of each tuple $t \in d$ using Eq. (2) (lines 4 to 6). Then we choose n_t tuples uniformly at random from the dataset (line 7) and from these n_t tuples we choose the one with the highest score (line 8). Referred to as the tournament size, n_t is a strictly positive integer. In our experiments, we set the value of n_t to 3 as commonly used in relevant literature [23].

5.2.3 ***Experimental Results.*** To answer $\mathcal{RQ}1$, we compared the described tuple sampling procedure of Algorithm 1 with the original Baran implementation. We ran each algorithm on each dataset 10 times, and the averaged values for precision, recall, and f_1 are presented in Table 1.

As seen in Table 1, the first experiment resulted in worse error correction performance compared with that of Baran. In most cases, the original implementation of Baran significantly outperformed Baran with the tournament selection-based tuple sampling technique. For most test cases, Baran obtained better values for the performance measures; however, the two experiments produced identical numbers for the Beers dataset. Moreover, the experiments also produced statistically similar precision, recall, and F_1 scores as seen in the last column of Table 1.

For all datasets except Beers, the original Baran implementation produced better values for the performance measures and for some datasets (such as Flights) this difference in performance was always statistically significant. However, there were cases in which the original Baran implementation obtained better values for the performance measures, but failed to statistically outperform our proposed modifications to Baran for the first experiment. For example, Baran obtained a better averaged (over 10 runs) precision for the Hospital dataset (0.88 compared with 0.87). However, this difference between precision values was deemed statistically insignificant by the Mann-Whitney U test.

Note that sometimes statistical tests require more observations (runs per experiment) to confidently determine statistical significance. Therefore, by running each algorithm more than 10 times, we may observe that Baran statistically outperforms the modifications of the first experiments for all datasets. To summarize, we make the following important observation based on the values of Table 1:

Observation 1. ***To summarize the results of the first experiment,*** *the tournament selection-based tuple sampling technique failed to show competitive performance for almost all datasets (except Beers). Even with only 10 runs per dataset, it was statistically outperformed by Baran with reference to many test instances* [3]*. More experiments are needed to see whether using bigger tournaments ($n_t > 3$) can outperform the original implementation of Baran. More runs per experiment (more than 10) are also needed to see whether Baran always statistically outperforms the proposed techniques of the first experiment.*

5.2.4 The Second Experiment (Non-Dominated Solutions). This section describes our second set of experiments for investigating the tuple sampling component of Baran. More specifically, we aim to answer $\mathcal{RQ}2$ and study the idea of selecting tuples that maximize some of (not necessarily all of) the criteria of Eq. (2).

As Eq. (2) is a product of multiple criteria, the motivation behind our second experiment is to investigate creative ways of representing the tuple sampling criteria. For this purpose, we use the properties of multi-objective optimization. As extensively described in Section 4.2, multi-objective optimization problems are concerned with obtaining a set of *optimal trade-offs* between two or three problems or objectives.

More specifically, a tuple might have many "frequent errors" (with respect to the entire dataset) that are unfixed, but its total number of unfixed errors might be small. Due to the properties of Eq. (2), this tuple might not be selected, even though fixing it would benefit Baran by fixing frequent error values scattered over the dataset. To deal with such scenarios, we convert the score value of Eq. (2) into the following *score tuple* which is to be maximized:

$$\mathbf{F} = (\mathbf{f}_1, \mathbf{f}_2) \tag{4}$$

where

$$\mathbf{f}_1 = \prod_{d[i,j] \in t \cap E'_j} \exp\left(\frac{\left|E'_j\right|}{\left|E_j\right|}\right) \tag{5}$$

and

$$\mathbf{f}_2 = \prod_{d[i,j] \in t \cap E'_j} \exp\left(\frac{\text{count}\left(d[i,j] \mid E'_j\right)}{\left|E'_j\right|}\right) \tag{6}$$

The pseudo-code for our second experiment is provided in Algorithm 3. For this experiment, we simply replaced the tuple sampling component of Baran with the one described in Algorithm 3. As seen in lines 4 to 6 in Algorithm 3, for each tuple of the dataset first individual scores $\mathbf{f}_1$ and $\mathbf{f}_2$ are calculated using Eq. (5) and Eq. (6) respectively. Then for each tuple $t_i \in d$ individual scores $\mathbf{f}_1$ and $\mathbf{f}_2$ are combined to form a score tuple $\mathbf{F}_i$ (as in Eq. (4)). More specifically,

[3] Let a test instance be the value of a given performance measure (precision, recall, and F_1) for a given dataset.

Table 1. The results of experiment 1. The reported numbers are averaged over 10 independent runs, and the best-performing approach with respect to each metric is highlighted in grey background.

Dataset	Result	Baran	Baran (Experiment 1)	Statistical Significance?
Hospital	Precision	0.88	0.87	No
	Recall	0.86	0.63	Yes
	F1	0.87	0.73	Yes
Flights	Precision	1.00	0.99	Yes
	Recall	1.00	0.95	Yes
	F1	1.00	0.97	Yes
Beers	Precision	0.92	0.92	No
	Recall	0.89	0.89	No
	F1	0.90	0.90	No
Rayyan	Precision	0.72	0.65	No
	Recall	0.43	0.35	Yes
	F1	0.54	0.46	No

Table 2. The results of experiment 2. The reported numbers are averaged over 10 independent runs, and the best-performing approach with respect to each metric is highlighted in grey background.

Dataset	Result	Baran	Baran (Experiment 2)	Statistical Significance?
Hospital	Precision	0.88	0.89	No
	Recall	0.86	0.87	No
	F1	0.87	0.88	No
Flights	Precision	1.00	1.00	No
	Recall	1.00	1.00	No
	F1	1.00	1.00	No
Beers	Precision	0.92	0.95	No
	Recall	0.89	0.91	No
	F1	0.90	0.93	No
Rayyan	Precision	0.72	0.63	No
	Recall	0.43	0.31	Yes
	F1	0.54	0.41	Yes

Table 3. The results of experiment 3. The reported numbers are averaged over 10 independent runs, and the best-performing approach with respect to each metric is highlighted in grey background.

Dataset	Result	Baran	Baran (Experiment 3)	Statistical Significance?
Hospital	Precision	0.88	0.87	No
	Recall	0.86	0.86	No
	F1	0.87	0.86	No
Flights	Precision	1.00	1.00	No
	Recall	1.00	1.00	No
	F1	1.00	1.00	No
Beers	Precision	0.92	0.93	No
	Recall	0.89	0.90	No
	F1	0.90	0.91	No
Rayyan	Precision	0.72	0.65	No
	Recall	0.43	0.41	No
	F1	0.54	0.50	No

the set of all score tuples $\mathcal{F} = \{\mathbf{F}_1, \ldots, \mathbf{F}_{|\mathcal{F}|}\}$ is formed. From $\mathcal{F}$, the set of dominated solutions are discarded and the Pareto-optimal front (see definition 4) is stored in *Candidates* (line 9 in Algorithm 3).

For calculating the Pareto-optimal set, we use a simple $O(n^2)$-time algorithm. The pseudo-code for this algorithm is provided in Algorithm 4.

5.2.5 ***Experimental Results.*** To answer $\mathcal{RQ}2$, we compared the described tuple sampling procedure of Algorithm 3 with the original Baran implementation. We ran each algorithm on each dataset 10 times, and the averaged values for precision, recall, and f_1 are presented in Table 2.

As seen in Table 2, the proposed techniques of the second experiment produced promising values for different performance measures (precision, recall, and F_1). Our second tuple sampling method (Algorithm 3) produced better or equal numbers for all performance measures for all datasets except one (Rayyan). For some test instances, this difference was more noticeable. For example, Baran with our tuple sampling technique achieved 0.95 precision for the Beers dataset, improving upon Baran's number (0.92). For the Flights dataset, both versions of Baran achieved perfect scores for precision, recall, and F_1 for all 10 runs.

One interesting observation was that our tuple sampling technique produced better numbers for the Beers dataset. As mentioned previously in Section 5.1.5, the Beers dataset is challenging to clean due to having low redundancy. Because of this low amount of data redundancy, specific scenarios might occur which are handled by our sampling method. For example, a dataset with low redundancy might have tuples that have many errors but do not contain any frequent data values. The original tuple sampling technique of Baran would simply ignore these tuples; however, our sampling method would give such tuples more opportunities of being selected since it uses a multi-objective optimization-based approach.

Unlike Beers, our method failed to outperform Baran for the Rayyan dataset. Baran outperformed our proposed technique with reference to all performance measures for the Rayyan dataset. Moreover, this difference in performance was statistically significant for all performance measures except precision. As previously discussed in Section 5.1.5, Rayyan is a challenging dataset due to having low data redundancy. Unlike Beers (a dataset with the same property), our proposed technique failed to show competitive performance for Rayyan.

Despite producing competitive numbers for all datasets except Rayyan, our method failed to statistically outperform Baran for all datasets. In future work, one can perform more runs per dataset (more than the current number of 10) to see whether having more observations available will result in a significant statistical difference between our method and Baran itself. To summarize, we make the following important observation based on the values of Table 2:

Observation 2. ***To summarize the results of the second experiment,*** *the tuple sampling technique showed promising performance, obtaining better or equal numbers for all measures and all datasets except Rayyan. With only 10 runs per dataset, our proposed modification failed to statistically outperform Baran. More runs per experiment (more than 10) are needed to see whether the second tuple sampling method (Algorithm 3) can statistically outperform Baran with more observations available.*

5.3 The Third Experiment

In this section, we explore some techniques to generalize the value-based model of Baran as it currently might lose performance under some circumstances. An example of this is provided in [19] where "Holland" is replaced by "Netherlands". However, this does not cover cases where the word "Holland" itself is misspelled (for example "Hlland"). The authors mention this issue in the same section by stating that their corrector needs to see the exact value "Holland" to be functional. Therefore, the use of a value-based corrector might be impractical as it requires every possible error (every possible misspelled version of "Holland") to be explicitly defined. Therefore, we propose a new error correction system, namely the spelling-based model, and implement it in Baran.

For our third proposed method, we first follow these steps to build a *dictionary tree*:

(1) Before the start of the algorithm, the clean (error-free) values are tokenized and stored in a list L_c of strings. Note that by assumption, we have access to all errors before the start of the algorithm. In other words, in all experiments (on Baran itself and our modifications) we assume there exists a perfect error detector that can always correctly tell erroneous and non-erroneous values apart.
(2) We remove all numerical strings from L_c. The reasoning behind this is that the fundamentals of the edit distance usually do not apply to numerical strings. Precisely, numerical strings have numerical properties which the edit distance does not consider.
(3) We construct a BK-tree [4] based on the strings of L_c, we denote this tree as $\mathcal{T}_c$. As discussed in Section 4.3, BK-trees are efficient data structures for edit distance calculation. Given a string s and our BK-tree $\mathcal{T}_c$ (containing strings from L_c), we can obtain the list of all strings in L_c that have an edit distance of at most n_d from s. In our experiments, we used value $n_d = 3$.

After applying the above preprocessing steps, we integrate our new error correction model, the *spelling-based model*, into Baran. This model is designed to fix spelling mistakes, and is integrated alongside the original models of Baran, namely the value-based model, the vicinity-based model, and the domain-based model. Given an erroneous value in a dataset, the spelling-based model calculates the edit distance of this erroneous value to all correct values in the same dataset. Note that unlike the original models of Baran that are constantly updated during the error correction phase, the spelling-based model is *static* in the sense that it is based on the initial correct values of a dataset. More precisely, the spelling-based model is not updated as more clean values are produced by Baran. In future work, a *dynamic* spelling-based model can be investigated.

A new error correction model that is implemented into Baran must meet certain requirements. More specifically, given an erroneous value in the dataset, an error correction model in Baran must report a set of correction candidates paired up with probabilities (between 0 to 1). These probabilities signify the closeness of an erroneous value to each candidate, such that higher probability values represent more likely candidates. However, the spelling-based model is based on the edit distance which is not normalized and allocates smaller numbers to similar strings (unlike the probability values of Baran). Therefore, for each correction candidate, we convert the "edit distance" into a "normalized edit similarity". The edit similarity between two strings s_1 and s_2 is calculated using the following

equation:

$$\text{SIMILARITY}(s_1, s_2) = \begin{cases} 0, & \text{if length}(s_1) = 0 \\ & \text{or length}(s_1) = 0 \\ \frac{M - \text{DISTANCE}(s_1, s_2)}{M} & \text{Otherwise} \end{cases} \tag{7}$$

where $M = \max\{\text{length}(s_1), \text{length}(s_2)\}$, and $\text{DISTANCE}(s_1, s_2)$ is the edit distance between s_1 and s_2. As seen in Eq. (7), if one of the strings is an empty string (with length equal to zero) then their similarity score will be equal to zero. The reasoning for this is to avoid fixing missing-value errors by using irrelevant strings. We use the dynamic programming-based algorithm described in Algorithm 2 to calculate the edit distance.

The pseudo-code of our spelling-based model is provided in Algorithm 5. As seen in Algorithm 5, for an erroneous value s, first a list of candidates is extracted using our BK-tree $\mathcal{T}_c$ (line 4). These candidates are the error-free entries of the dataset that have an edit distance of at most n_d from s. Between s and each of its correction candidates, the edit similarity (of Eq. (7)) is calculated (lines 5 to 7 in Algorithm 5). Finally, the list of these correction candidates alongside their edit similarities is returned. This list is fed into Baran's original implementation in which each model returns a list of candidate-probability pairs for a given erroneous value.

5.3.1 ***Experimental Results.*** To assess the effectiveness of Baran in fixing spelling errors, we compared the described procedure of Algorithm 5 with the original Baran implementation. We ran each algorithm on each dataset 10 times, and the averaged values for precision, recall, and f_1 are presented in Table 3.

As seen in Table 3, the proposed techniques of the third experiment produced competitive values for different performance measures (precision, recall, and F_1). Our spelling-based model (Algorithm 5) produced better or equal numbers for all performance measures and all datasets except Hospital and Rayyan. Unlike the second experiment, the spelling-based model obtained inferior numbers for precision and F_1 for the Hospital dataset. However, this difference in performance was not statistically significant for any of the two measures. The spelling-based model obtaining poor numbers for the Hospital dataset is surprising because this dataset has a high level of data redundancy (as mentioned in Section 5.1.5). Intuitively, on such a dataset our spelling-based model should perform well as it maps incorrect errors of a dataset to its correct ones using the edit distance. However, in practice, the opposite was observed and this has to be studied in greater detail.

Similar to the second experiment, our method failed to outperform Baran for the Rayyan dataset. Baran outperformed our proposed technique with reference to all performance measures for the Rayyan dataset. However, unlike in the second experiment, this difference in performance was not statistically significant for any of the performance measures. As previously discussed in Section 5.1.5, Rayyan is a challenging dataset due to having low data redundancy. Our spelling-based model maps the erroneous values of a dataset to its correct ones, and thus is reliant on data redundancy.

Despite producing competitive numbers for all datasets except Rayyan and Hospital, our method failed to statistically outperform Baran for all datasets. As future work, one can perform more runs per dataset (more than the current number 10) to see whether having more observations available will result in significant statistical difference between our method and Baran itself. To summarize, we make the following important observation based on the values of Table 3:

Observation 3. ***To summarize the results of the third experiment,*** *the spelling-based model (Algorithm 5) showed promising performance, obtaining better or equal numbers for all measures and all datasets except Rayyan and Hospital. With only 10 runs per dataset, our proposed modification failed to statistically outperform Baran. More runs per experiment (more than 10) are also needed to see whether the spelling-based model (Algorithm 5) can statistically outperform Baran with more observations available.*

6 CONCLUSION AND FUTURE WORK

In this project, we studied the Baran error correction system and investigated some important research questions that were unanswered in the original paper of Baran. First, a literature review of some of the past and recently proposed approaches for error detection and correction systems was provided.

As an error correction system, Baran is reliant on sampling a limited number of dataset tuples to be corrected by an expert. As the second contribution of our project, we proposed two alternative approaches for tuple sampling in Baran, one of which resulted in promising results by producing better values for precision, recall, and F_1 for most datasets. Moreover, we proposed a new model, namely the spelling-based model, and we integrated it into Baran. This new model also showed competitive performance when compared with the original implementation of Baran. However, none of our proposed techniques could statistically outperform Baran (as determined by the Mann-Whitney U test).

Regarding future work, multiple avenues of future work come to mind. First, more datasets can be considered for comparison purposes. Due to the lack of time, we only included four datasets in our experiments, namely Hospital, Flights, Beers, and Rayyan. Furthermore, our first tuple sampling technique can be evaluated with more experiments. More specifically, one can study whether using bigger selection tournaments (we used $n_t = 3$ in our experiments) can improve the error correction performance.

Moreover, more experiments are needed to see whether our second tuple sampling technique and our spelling-based model can **significantly** improve the effectiveness of Baran. Note that the aforementioned approaches obtained better precision, recall, and F_1 values than Baran, but this improvement was deemed insignificant by the Mann-Whitney U test. However, we ran each algorithm on each dataset only 10 times due to the lack of time. In future work, one can increase the number of runs per dataset to see whether the difference between Baran and our proposed approaches will reach statistical significance.

REFERENCES

[1] Patricia C Arocena, Boris Glavic, Giansalvatore Mecca, Renée J Miller, Paolo Papotti, and Donatello Santoro. Messing up with bart: error generation for evaluating data-cleaning algorithms. *Proceedings of the VLDB Endowment*, 9(2):36–47, 2015.

[2] Richard Bellman, Roger Fletcher, Ronald A Howard, Fritz John, Narendra Karmarkar, William Karush, Leonid Khachiyan, Bernard Koopman, Harold Kuhn, László Lovász, et al. Mathematical optimization.

[3] Mahdi Boroujeni, Mohammad Ghodsi, and Saeed Seddighin. Improved mpc algorithms for edit distance and ulam distance. *IEEE Transactions on Parallel and Distributed Systems*, 32(11):2764–2776, 2021.

[4] Walter A. Burkhard and Robert M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, 1973.

[5] Youness Chaabi and Fadoua Ataa Allah. Amazigh spell checker using damerau-levenshtein algorithm and n-gram. *Journal of King Saud University-Computer and Information Sciences*, 2021.

[6] Xu Chu, Ihab F Ilyas, and Paolo Papotti. Holistic data cleaning: Putting violations into context. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 458–469. IEEE, 2013.

[7] Xu Chu, John Morcos, Ihab F Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1247–1261, 2015.

[8] Yann Collette and Patrick Siarry. *Multiobjective optimization: principles and case studies*. Springer Science & Business Media, 2004.

[9] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, and Nan Tang. Nadeef: a commodity data cleaning system. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 541–552, 2013.

[10] Kalyanmoy Deb and Rituparna Datta. Hybrid evolutionary multi-objective optimization and analysis of machining operations. *Engineering Optimization*, 44(6):685–706, 2012.

[11] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K Elmagarmid, Ihab F Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. The data civilizer system. In *Cidr*, 2017.

[12] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[13] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. The llunatic data-cleaning framework. *Proceedings of the VLDB Endowment*, 6(9):625–636, 2013.

[14] Gergo GOMBOS, János Márk SZALAI-GINDL, István DONKÓ, and Attila KISS. Towards on experimental comparison of the m-tree index structure with bk-tree and vp-tree. *Acta Electrotechnica et Informatica*, 20(2):19–26, 2020.

[15] Jeffrey Horn, Nicholas Nafpliotis, and David E Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the first IEEE conference on evolutionary computation. IEEE world congress on computational intelligence*, pages 82–87. Ieee, 1994.

[16] Sampreeti Jena. *Multi-Objective Optimization of the Design Parameters of a Shell and Tube Type Heat Exchanger Based on Economic and Size Consideration*. PhD thesis, 2013.

[17] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment*, 9(12):948–959, 2016.

[18] Xian Li, Xin Luna Dong, Kenneth Lyons, Weiyi Meng, and Divesh Srivastava. Truth finding on the deep web: Is the problem solved? *arXiv preprint arXiv:1503.00303*, 2015.

[19] Mohammad Mahdavi and Ziawasch Abedjan. Baran: Effective error correction via a unified context representation and transfer learning. *Proceedings of the VLDB Endowment*, 13(12):1948–1961, 2020.

[20] Mohammad Mahdavi and Ziawasch Abedjan. Semi-supervised data cleaning with raha and baran. In *CIDR*, 2021.

[21] Masuduzzaman and GP Rangaiah. Multi-objective optimization applications in chemical engineering. *Multi-Objective Optimization: Techniques and Applications in Chemical Engineering (With CD-ROM)*, pages 27–59, 2009.

[22] Patrick E McKnight and Julius Najab. Mann-whitney u test. *The Corsini Encyclopedia of Psychology*, pages 1–1, 2010.

[23] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[24] Mourad Ouzzani, Hossam Hammady, Zbys Fedorowicz, and Ahmed Elmagarmid. Rayyan—a web and mobile app for systematic reviews. *Systematic reviews*, 5(1):1–10, 2016.

[25] Tania Pencheva, K Atanassov, and A Shannon. Modelling of a roulette wheel selection operator in genetic algorithms using generalized nets. *International Journal Bioautomation*, 13(4):257, 2009.

[26] Nataliya Prokoshyna, Jaroslaw Szlichta, Fei Chiang, Renée J Miller, and Divesh Srivastava. Combining quantitative and logical data cleaning. *Proceedings of the VLDB Endowment*, 9(4):300–311, 2015.

[27] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. Holoclean: Holistic data repairs with probabilistic inference. *arXiv preprint arXiv:1702.00820*, 2017.

[28] Enrique H Ruspini and Igor S Zwir. Automated qualitative description of measurements. In *IMTC/99. Proceedings of the 16th IEEE Instrumentation and Measurement Technology Conference (Cat. No. 99CH36309)*, volume 2, pages 1086–1091. IEEE, 1999.

[29] Matias Sessarego, KR Dixon, DE Rival, and DH Wood. A hybrid multi-objective evolutionary algorithm for wind-turbine blade optimization. *Engineering Optimization*, 47(8):1043–1062, 2015.

[30] Ma Guadalupe Castillo Tapia and Carlos A Coello Coello. Applications of multi-objective evolutionary algorithms in economics and finance: A survey. In *2007 IEEE Congress on Evolutionary Computation*, pages 532–539. IEEE, 2007.

[31] Maksims Volkovs, Fei Chiang, Jaroslaw Szlichta, and Renée J Miller. Continuous data cleaning. In *2014 IEEE 30th international conference on data engineering*, pages 244–255. IEEE, 2014.

[32] Jiannan Wang, Sanjay Krishnan, Michael J Franklin, Ken Goldberg, Tim Kraska, and Tova Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 469–480, 2014.

[33] Mohamed Yakout, Laure Berti-Équille, and Ahmed K Elmagarmid. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 553–564, 2013.

[34] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

[35] Fernando Alonso Zotes and Matilde Santos Peñas. Particle swarm optimisation of interplanetary trajectories from earth to jupiter and saturn. *Engineering Applications of Artificial Intelligence*, 25(1):189–199, 2012.