CSP-DC: Data Cleaning via Constraint Satisfaction Problem Solving

Nibel Nadjeh, Sabrina Abdellaoui and Fahima Nader
Laboratoire des Méthodes de Conception de Systèmes (LMCS),
Ecole Nationale Supérieure d'Informatique (ESI), BP, 68M Oued-Smar, 16270 Alger, Algeria

Keywords: Data Quality, Data Consistency, Data Cleaning, Constraint Satisfaction Problems.

Abstract: In this paper, we present CSP-DC, a data cleaning system that integrates a new intelligent solution into the cleaning process to improve data accuracy, consistency, and minimize user involvement. We address three main challenges: (1) Consistency: Most repairing algorithms introduce new violations when repairing data, especially when constraints have overlapping attributes. (2) Automaticity: User intervention is time-consuming, we seek to minimize their efforts. (3) Accuracy: Most automatic approaches compute minimal repairs and apply unverified modifications to repair ambiguous cases, which may introduce more noise. To address these challenges, we propose to formulate this problem as a constraint satisfaction problem (CSP) allowing updates that always maintain data consistency. To achieve high performances, we perform a first cleaning phase to automatically repair violations that are easily handled by existing repair algorithms. We handle violations with multiple possible repairs with a CSP solving algorithm, which selects from possible fixes, values that respect all constraints. To reduce the problem's complexity, we propose a new variables ordering technique and pruning strategies, allowing to optimize the repair search and find a solution quickly. Our experiments show that CSP-DC provides consistent and accurate repairs in a linear time, while also minimizing user intervention.

1 INTRODUCTION

Lately, data has been widely used for prediction, analysis, and decision making, therefore it has become the most valuable resource in the world. However, real data is usually of poor quality, according to a Kaggle survey[1], dirty data is considered as the biggest barrier in data science (Berti-Equille, 2019). Moreover, the use of this poor-quality data can lead to bad decision-making which can cause time and money losses, as well as serious repercussions in some critical application areas, such as healthcare and automotive (Yakout et al., 2011). Consequently, data quality management has become crucial for both data scientists and decision makers.

To improve the quality of the data, quality rules (QRs) have been used to identify errors and inconsistencies. Data is then modified to respect the set of QRs expressed as integrity constraints (ICs) such as: Functional Dependencies (FD) (Bohannon et al., 2005), Conditional Functional Dependencies (CFD) (Cong et al., 2007), Denial Constraints (DC) (Xu Chu et al., 2013), etc. (Ilyas and Chu, 2015)

Over the years, several data cleaning approaches have been proposed (Raman and Hellerstein, 2001), (Yakout et al., 2011), (Fan et al., 2012), (Volkovs et al., 2014). Most of them have opted for repairs that minimize a cost function defined as the distance between original and modified data (Dallachiesa et al., 2013), (Bohannon et al., 2005), (Chiang and Miller, 2011), (Wang and Tang, 2017). However, choosing the minimal repair may add more noise to the data by modifying the wrong cells or selecting incorrect repair values (Rezig et al., 2021), (Yakout et al., 2011). Furthermore, repairing cells that participate in several ICs is challenging, especially when dealing with errors from the active domain.

To improve the quality of repairs, external data like knowledge bases (KBs) have been used to extract relevant data repairs (Chu et al., 2015), (Abdellaoui et al., 2017). However, human intervention is usually needed especially when multiple possible repairs are proposed (Yakout et al., 2011), (Chu et al., 2015), (Abdellaoui et al., 2017). Human intervention is a tedious and time-consuming process, especially with the huge amount of data to deal with and the high number of possible repairs usually generated by repair algorithms.

[1]https://www.kaggle.com/code/amberthomas/kaggle-2017-survey-results/report

Nadjeh, N., Abdellaoui, S. and Nader, F.
CSP-DC: Data Cleaning via Constraint Satisfaction Problem Solving.
DOI: 10.5220/0011897000003393
In *Proceedings of the 15th International Conference on Agents and Artificial Intelligence (ICAART 2023) - Volume 2*, pages 478-488
ISBN: 978-989-758-623-1; ISSN: 2184-433X

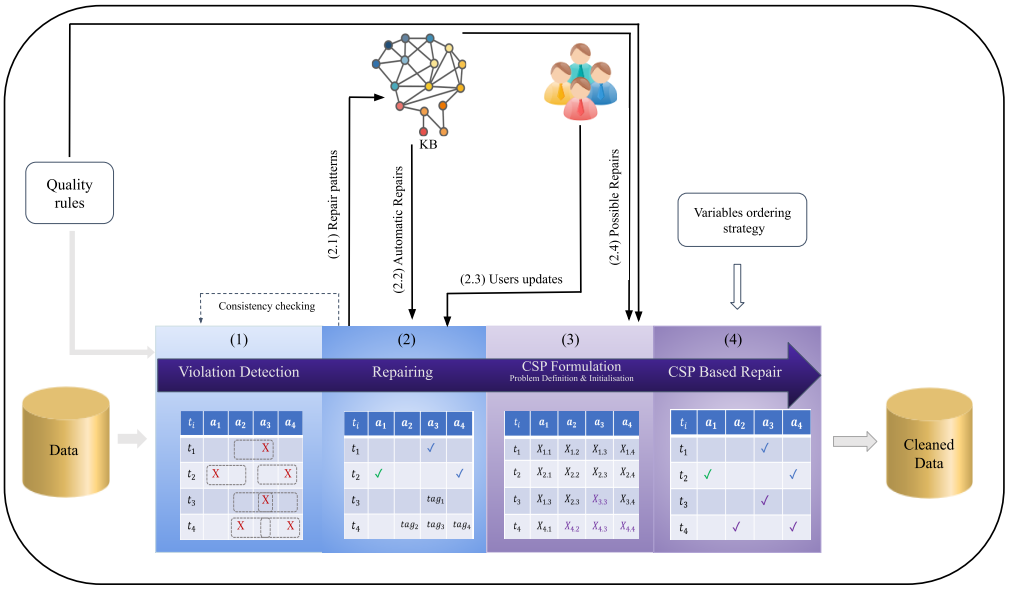


Figure 1: The workflow of CSP-DC.

With the aim of automating the repair process and minimizing human effort without trading off the consistency nor the accuracy of the repairs, we propose CSP-DC, a new data cleaning system. CSP-DC uses one of the state of the art data cleaning algorithms to automatically repair data when possible or generate possible repairs otherwise. For this step, we use QDflows (Abdellaoui et al., 2017) but our solution supports any other repair algorithm as long as it returns possible fixes that could be exploited by the CSP-based solution.

Instead of involving the user to choose the appropriate repair, we propose a new solution to automatically repair ambiguous repair cases. Our solution is inspired from constraint satisfaction problems resolution (CSP) (Poole and Mackworth, 2010). A CSP consists of variables X, domains D which represent variables' possible values, and a set of constraints C. They are designed to solve problems under constraints by assigning to each variable a value that satisfies all constraints. In our case, it consists of finding repairs to dirty cells, which respect the entire set of integrity constraints simultaneously. The choice of translating the consistency problem to a CSP was based on the fact that the two problems are very similar. Indeed, variables correspond to cells, domains refer to the possible repairs of cells, and constraints represent the set of ICs. Furthermore, when solving a CSP, the purpose is to find assignments to variables that must respect all constraints. Data cleaning on the other hand aims to find repairs to dirty data in order to respect the ICs.

As far as we know, no existing approach has formulated the data cleaning problem as a CSP. We believe that this representation and the use of a CSP solving algorithm allow us to improve data quality and guarantees the consistency of repairs. Since all constraints are verified when instantiating variables, a holistic choice is made to repair each erroneous cell which ensures high accuracy. Like many data cleaning solutions (Mahdavi and Abedjan, 2020), (Geerts et al., 2013),(Xu Chu et al., 2013), (Yakout et al., 2011), we enable users' intervention when no possible repair is returned by QDflows. This allows us to repair violations when no evidence is available in data or when the repair value must be out of the active domain.

To achieve our goal, we face several challenges including:

- The data cleaning problem has never been formalized as a constraint satisfaction problem. Therefore, there is no definition or representation of the problem. The challenge is: (1) Identifying the variables of the problem and how to relate them to the data without losing the context of the cells. (2) In a CSP, a constraint is defined on variables but in our case, ICs are defined on attributes. Therefore, it is difficult to translate an IC into a constraint on variables without losing generality (i.e., exhaustively defining a constraint for each subset of the problem's variables). (3) Since a part of the data is automatically repaired, the CSP solving algorithm must take into account their values without modifying them.
- Solving a CSP consists of finding a repair that re-

spects all constraints. If the initialized variables do not respect the constraints, the CSP may become unsolvable. As a part of the data is repaired automatically and used as an initialization of the CSP, their repair must be done with great care in order to avoid generating inconsistencies, because the final result strongly depends on this step.

- Solving a CSP is NP-Complete (Pang and Goodwin, 2003). Moreover, the time complexity of the most known CSP solving algorithms is exponential (Razgon, 2006). For example, the worst case time complexity of the backtracking algorithm is of the order $O(e * d^n)$ (with n the number of variables, e the number of constraints, and d the size of the largest domain) (Mouelhi et al., 2013). As our problem has a large number of variables and values, it becomes challenging to solve it efficiently.

Contribution:

1. We propose a representation of the data cleaning problem as a CSP (Section 3.1) and solve it using one of the dedicated solving algorithms (Section 3.2). This allows to:
   - Find repairs to the ambiguous repair cases in an effective and consistent way.
   - Repair cells participating in several ICs safely. As changing their values is difficult and can trigger new violations, these cells are critical to repair. By formulating our problem as a CSP, the solving algorithm takes into account all constraints while repairing.
   - Replace the user intervention by automatically selecting fixes to errors not easily handled by classical algorithms.
2. We propose a pruning strategy to improve the efficiency of our solution (Section 4.1)
3. We propose a new variable ordering technique adapted to our data cleaning problem which allows us to solve it efficiently (Section 4.2).
4. We conduct experiments to assess our solution's effectiveness and efficiency. We compare its performance to those of some existing algorithms and study how our optimizations impact the complexity of the problem (Section 5).

2 MOTIVATION & SOLUTION OVERVIEW

In this section, we present some motivating examples as well as the architecture overview of CSP-DC.

2.1 Motivation

Let the relation in Figure 2 and the following set of integrity constraints:
$FD1 : ZIPCode \rightarrow State$
$FD2 : PhoneNumber \rightarrow ZIPCode$
$FD3 : ProviderNumber \rightarrow City, PhoneNumber$
$CFD : (ZIPCode \rightarrow State, 35233 \| AL)$.
A typical cleaning scenario is to detect violating tuples, for instance, tuples t_4, t_6, and t_7 violate the FD1 and $t_1 t_2$, and t_5 violate the FD3, then use a repair algorithm to find updates that satisfy the defined ICs.
In the following, we define multiple scenarios where it would be useful to use our proposed solution.

1. When multiple possible repairs validated by external data are returned, but still can't decide about cells to modify and values to select: In this case, users are involved to repair dirty data.

   Example 1. Consider that we use a repair algorithm that returns multiple possible repairs extracted from data or KBs. For the first violation, ('35233', '35235') are returned as possible fixes for $t_4[ZIPCode]$ and 'SC' for $t_4[State]$. For the second one, '10011' and '2053258100' are returned as possible repairs for $t_5[ProviderNumber]$ and $t_5[PhoneNumber]$ respectively. Instead of involving the user to ensure data consistency, we mark each cell participating in these violations, extract the possible repair values and select via a CSP solving algorithm the combination that guarantees the data consistency.

2. When dealing with ICs with overlapping attributes: Repairing such a case is difficult because modifying a cell value to resolve a violation may trigger a new violation of another IC. Most existing repair algorithms do not handle these types of ambiguous repair cases very well. To resolve violations mentioned in Example 1, they would apply a random repair or modify the right-hand-side to resolve the violation. This may introduce more noise in data by modifying $t_4[State]$ or generate a new violation of FD2 by changing $t_4[ZIPCode]$ value to '35233' and/or $t_5[PhoneNumber]$ to '2053258100'.
   By formulating our problem as a CSP, dealing with overlapping attributes becomes safer as all constraints are verified before updating data.
3. When the data do not contain enough evidence to resolve the conflict safely: A possible way to handle this case is to use lluns (Geerts et al., 2013) or fresh values (Xu Chu et al., 2013) for example to mark the concerned cells. This allows to return

ti	ProviderNumber	City	State	ZIPCode	PhoneNumber
t1	10018	BIRMINGHAM	AL	35233	2053258100
t2	10018	BIRMINGHAM	AR	35233	2053258100
t3	10011	BIRMINGHAM	AL	35235	2058383122
t4	10011	BIRMINGHAM	AL	29730	2058383122
t5	10018	BIRMINGHAM	AL	35235	2058383122
t6	420002	ROCK HILL	SC	29730	8033291234
t7	420002	PLAINVIEW	SC	29730	8033291234
t8	50022	RIVERSIDE	CA	35233	9517883000

Figure 2: Example of tuples from a dirty Dataset.

a consistent result then, ask for the user intervention to repair marked cells. CSP-DC on the other hand, handles this case via the CSP based solution as long as possible fixes to the concerned cells are available.

Example 2. A possible repair is to update t_8[State] to 'AL' or to change t_8[ZIPCode]. However, no possible value from the active domain satisfies the ICs therefore, the user intervention is required to guarantee the accuracy.

2.2 Solution Overview

Figure 1 illustrates the workflow of CSP-DC. It is composed of two cleaning phases:

1. **Violation Detection and Updates Discovery:** In this phase, violations are identified using the specified QRs expressed as ICs. Then, QDflows constructs for each violation, a repair pattern based on properties involved in the concerned QRs. Next, horizontal and vertical matches of the repair patterns are searched in the KB. Outputs of QDflows are divided into three groups:
   (a) *Violations with one possible repair:* this means that the proposed update is likely to be the right one, data is thus automatically repaired.
   (b) *Violations with no possible repairs:* here, the users are invited to repair the involved cells.
   (c) *Violations with multiple possible repairs:* in this case, no decision is made at this stage. The concerned cells are annotated using "tags" and their possible repair values are collected.

   At the end of this phase, the consistency of data is verified. If new violations were triggered when repairing the first violations, another cleaning iteration is done.
2. **CSP Based Repair:** In this phase, updated data, QRs, and possible repairs are transformed to define the problem as a CSP and initialize its elements. Variables referring to clean data are initialized, domains of annotated cells are constructed using the possible repairs, and the QRs are translated to constraints on variables. Finally, the CSP is solved by finding repair to annotated data using a backtracking search algorithm which returns the repaired data.

3 CSP BASED SOLUTION

In this section, we present our solution based on constraint satisfaction problems by explaining how our problem is translated into a CSP, how its elements are initialized, and how the problem is solved.

3.1 Problem Definition and Initialization

Our data cleaning problem is composed of two main elements: (1) Data that represents a list of tuples, each one contains multiple attribute values. (2) Integrity constraints that formulate a set of quality rules in order to ensure data consistency.
The representation of our problem as a CSP was inspired from AIMA's [2]formulation of the SUDOKU problem to a CSP. We define our problem as follows:

- *Variables* ($\mathcal{X}$): Each cell is considered as a variable $X_{i \cdot j}$ of the problem where "i" indicates the identifier of the tuple to which the variable belongs and "j" the attribute it refers to.
- *Domains* ($\mathcal{D}$): Represent the different possible values that can be assigned to variables. Each variable has its own domain but it's possible to have multiple variables sharing the same domain.

  Example 3. $D(X_{4.3})$: ["AL", "SC"],
  $D(X_{4.4})$: ["29730", "35233", "35235"].

- *Constraints*($\mathcal{C}$): are a translation of ICs into conditions on variables. A variable may be under multiple constraints.

[2]https://github.com/aimacode/aima-python

After formalizing the problem, we initialize its elements using the CSPBasedRepair algorithm, given in Algorithm 1 (line 1). Variables related to the clean cells are initialized by assigning the value corresponding to the tuple "i" and the attribute "j" to the variable $X_{i.j}$ the rest of variables (related to annotated cells) are set to 'tags'. After that, the possible repairs returned for the annotated cells, are used in addition to their original values to construct their domains.

3.2 Problem Resolution

Our constraint satisfaction problem's resolution is done using a backtracking search algorithm which consists in assigning to each variable a value and recursively checking for each assignment, if a solution to the problem can be returned from the current set of assignments. If no solution is possible (i.e., the assignment does not respect one or multiple constraints), a backtrack is necessary. Algorithm 2, referred to as BacktrackingSearch, describes the different steps. It takes as input variables related to each cell, their domains, their neighbors (i.e., other variables to be compared with in order to ensure the consistency), the defined constraints, and optionally, heuristics for variables and values ordering and/or a filtering strategy (inference). In turn, from the set of unassigned variables, a variable is chosen and a value from its domain is selected. Variables and values could be picked randomly but it is possible to use existing variable selection heuristics and value ordering heuristics to optimize the search. If an assignment respects all constraints (i.e., No triggered violation) then, this value is assigned to the current variable. In the same way, assignments to other variables are found. If a variable has all its domain values violating one or multiple constraints, a backtrack is triggered. The purpose is to change the value of a previously instantiated variable that may cause this inconsistency, which offers more options for the next steps of the search. The solution is returned when all variables are instantiated which means that consistent data is returned.

4 OPTIMIZATIONS

As solving a CSP is NP-complete, we propose multiple optimizations to avoid the exponential time complexity of the backtracking search.

Algorithm 1: CSPBasedRepair.

Input: An annotated dataset D, Constraints C, Variables V, VariableSelectionHeuristic VSH, ValueOrderingHeuristic VOH, inference inf
Output: Repaired dataset
1: $Partial\_assignment \leftarrow$ initPartial_assignment(D,V)
2: **return** BacktrackingSearch ($Partial\_assignment$, V, C, Dom, N, VSH, VOH, inf)

Algorithm 2: BacktrackingSearch.

Input: $Partial\_assignment$, V, C, Dom, N, VSH, VOH, inf
Output: Solution to the CSP
1: **if** $|Partial\_assignment| = |V|$ **then**
2: **return** $Partial\_assignment$
3: **end if**
4: **for** var in $VSH(V)$ **do**
5: **for** val in $VOH(Dom[var])$ **do**
6: **for** $var2$ in $N[var]$ **do**
7: $val2 \leftarrow Partial\_assignment[var2]$
8: **if** $constraints(var, val, var2, val2)$ **then**
9: $Partial\_assignment$[var] $\leftarrow val$
10: **if** $inf(var, val, Partial\_assignment)$ **then**
11: $result \leftarrow$BacktrackingSearch ($Partial\_assignment$, V, C, Dom, N, VSH, VOH, inf)
12: **end if**
13: **if** $result$ is not $None$ **then**
14: **return** $result$
15: **end if**
16: **end if**
17: **end for**
18: **return** $None$
19: **end for**
20: **end for**

4.1 Neighbors and Domains Pruning

Neighbors Pruning: The complexity of the backtracking search depends on the number of variables, constraints, and values of the largest domain. However, as explained before, instantiating a variable implies a consistency verification with respect to all constraints. In practice, this comes to a pairwise comparison between each non-instantiated variable and all the other variables concerned by the same constraint (neighbors).

To overcome the pairwise comparison, we propose a pruned list of neighbors. To do this, for each IC, we extract a non-redundant list of "n"-uplets (For instance (35233, AL)) with "n" the number of attributes of the IC. This means that the comparison will be done only with one occurrence of the "n"-uplets instead of all of them, which avoids the comparison with the same data combination multiple times. The list of neighbors is constructed then, by selecting variables referring to cells of the first appearance of the "n"-uplet in data. If a variable corresponds to an at-

tribute participating in multiple constraints, neighbors are selected considering all these constraints.
Domains Pruning: A possible way to initialize variables' domains is to select all values of the attribute's domain that the concerned variable refers to. To avoid the exponential time complexity, we use the list of possible repairs. This allows us to narrow down the number of values to test before finding the right repair and consequently, reducing the repair time.

4.2 Variables Ordering Technique to Improve the Repair Efficiency

As explained above, when the backtracking search algorithm is used to solve a CSP, variable selection heuristics and value ordering heuristics are usually used. However, the use of the existing heuristics is sometimes time consuming. Since our problem has a high number of variables, we propose another way to order them. The idea behind is to use conflict score (cf_F) used for FD ordering (Chiang and Miller, 2011). The conflict score cf_F reflects the conflicts that a constraint F has with other constraints based on the number of common attributes between them. Instead of calculating the cf_F score for constraints ordering, we propose to use it also for attribute ordering. Indeed, variables can be grouped using the attributes that they refer to, so instead of having $n*m$ variables to order, we just define an order on the "m" groups, each one having "n" variables. The selection of variables inside groups can be done randomly, since they share the same constraints, the accuracy of repairs won't be affected. The selection is done as follows:

1. The constraint with the highest cf_F is chosen first.
2. The most overlapping attributes of the chosen IC are selected in descending order.
3. Two attributes having the same overlap level can be selected randomly if they belong to different ICs; otherwise, the LHS attribute is selected first.
4. The process is repeated for all other ICs by choosing only attributes that were not selected yet.
5. Variables belonging to the same attribute are repaired in a random order or in a specific order using a dedicated heuristic.

Example 4. Consider the previously used FDs and the annotated tuple t_5:(tag_i, BIRMINGHAM, AL, 35235, tag_j). Repairing ProviderNumber first, can be difficult and may take time. If '10018' is selected tin he first place, a backtrack is triggered when we look for repairs to $t5[PhoneNumber]$This happens because the only value that respects FD3 is '2053258100' but it violates FD2 since the ZIPCode value is 35235.

By ordering the attributes the way we propose, we find that $cf_F(FD2) > cf_F(FD1) > cf_F(FD3)$. The order of attributes should be like : PhoneNumber, ZIPcode, State, ProviderNumber, City. The PhoneNumber value is chosen first with no ambiguity as only 2058383122 is accepted. The ProviderNumber value will be then updated to '10011' and the error will be corrected.

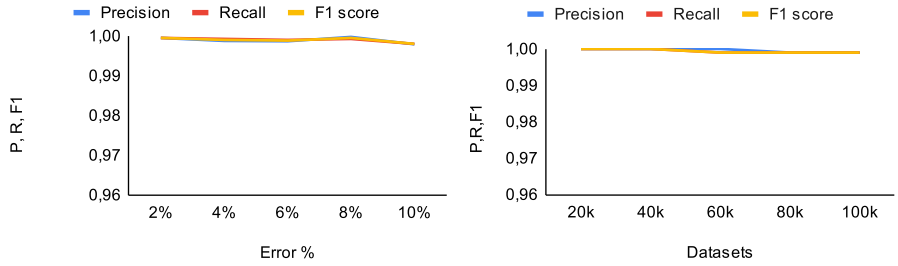
5 EXPERIMENTAL STUDY

In our experimental study, we evaluate the efficiency and the effectiveness of our solution. We use Hospital, a real-world dataset on health-care providers that contains 100k records. Errors from the active domain were generated on all attributes covered by the FDs used in the previous examples (FD1, FD2, and FD3). Our experiments aim to: (1) Evaluate the effectiveness of CSPBasedRepair algorithm when varying the error rate and the dataset size, (2) Study the impact that the error rate and the dataset size have on the efficiency of the algorithm, (3) Study the impact that our optimizations have on the efficiency of the algorithm and the complexity of the problem, (4) Compare our results with some existing solutions.

5.1 Effectiveness

(a) ***Varying the Noise Rate and the Size of the Dataset.***
To evaluate the repair accuracy of CSP-DC, we used 20k records of the Hospital dataset and injected different error rates. Results illustrated in Figure 3a show that the F1 score is stable and higher than 99.8%. This is due to the good automatic repairs and the user intervention in the first repair phase as well as the instantiation of the annotated cells by their correct values using the CSPbasedRepair algorithm.
Note that CSPBasedRepair returns consistent repairs but in some cases, it is not possible to achieve a 100% clean repairs. This happens when dealing with some special cases difficult to handle even by users (see Example 6).

Example 5. Consider the following tuples extracted from the used Hospital dataset: t_i (50008, SAN FRANCISCO, CA, 94115, 4156006000), t_j (50047, SAN FRANCISCO, CA, 94115, 4156006000), and t_k (10018, SAN FRANCISCO, CA, 94115, 4156006000). To repair this violation, two possible repairs are returned for $t_k[ProviderNumber]$ 50008, 50047.)In this case,



(a) Hospital (2% - 10%) (b) Hospital (20k-100k)

Figure 3: Accuracy when varying the noise rate and the dataset size.

both values guarantee the consistency and there is no way to identify the good repair, especially that tuples share the same ZIPCode and State values, therefore, the F1 score may not reach 100%.

To study the impact of the dataset size on the effectiveness of repairs, we fixed the noise rate at 1% and varied the dataset size from 20k to 100k, Figure 3b show that the F1 score is higher than 99% with a 100% precision all the time. It is clear that our repair algorithm is not affected by the increase of the dataset size as it depends on the possible repairs extracted in the previous step.

(b) ***Studying the Degree of User Intervention.*** In this experiments, we used 20k records and varied the error rate from 2% to 10%. The Figure 4 shows that when the error rate is lower than 4%, 100% of the dirty cells are repaired automatically. The user intervention rate increases slightly when increasing the error rate, which was expected because the more noise we have, the more difficult it becomes to find repairs. Despite the increase of human involvement, it didn't go beyond 0.08% even when the error rate was rather high (10%), which means that only 8 cells were manually repaired. On the other hand, about 34% of the dirty cells were repaired automatically via CSPBasedRepair instead of repairing them semi-automatically.

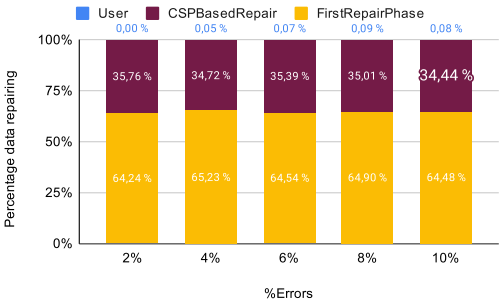


Figure 4: User intervention when varying the dataset size.

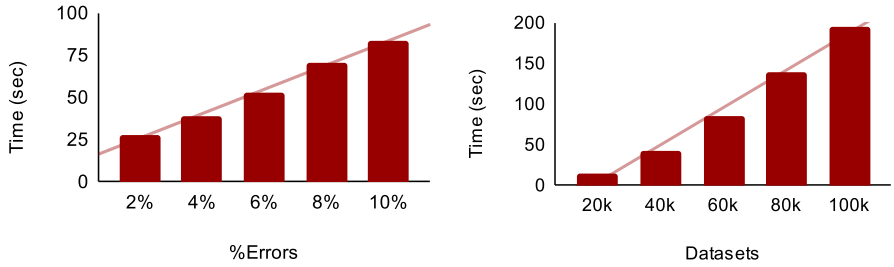
5.2 Efficiency and Run-Time

5.2.1 Run-Time Results

In Figure 5, we report the run-time results when varying: 1) the error rate on 20k records and 2) the dataset size from 20k to 100k with the error rate fixed at 1%. The Figure 5 illustrates the efficiency of CSPBasedRepair. Since the algorithm is executed after a first cleaning phase, the initial error rate does not represent the rate of cells to repair.
Our experiments are run on a 64bit AMD Ryzen 7 5800H and 32GB RAM. As the CSPBasedRepair can be executed in parallel, we used 16 CPU threads for its execution.
Results illustrated in Figure 5a and Figure 5b show that our algorithm is fast. We observe a linear curve when varying the error rate or the dataset size. This is due to our optimizations as well as the fact that the algorithm is designed to be executed in parallel.

5.2.2 Optimization Impact on Run-Time

To study the impact of the domains and neighbors pruning as well as our variables ordering on the efficiency, we varied the dataset size from 5k to 20k and fixed error rate at 1%. We consider three combinations: (1) CSPBasedRepair with the pruning process and our proposed variables ordering technique. (2) CSPBasedRepair and variables ordering technique but without the pruning process. (3) CSPBasedRepair without the pruning process nor the variables ordering technique (i.e., random selection). As illustrated in Figure 6, using our optimizations allowed us to accelerate the repair process. Indeed, when the dataset size reaches 20k our repair time is 300 times faster than the repair without the pruning and 1500 times faster than the repair without the pruning nor the variables ordering. In other words the pruning process and the variables ordering allowed us to reduce the repair time by more than 21% and 78% respectively (i.e, 99% when used together). We also notice that the more data we



(a) Noise (b) Size

Figure 5: Run-time results.

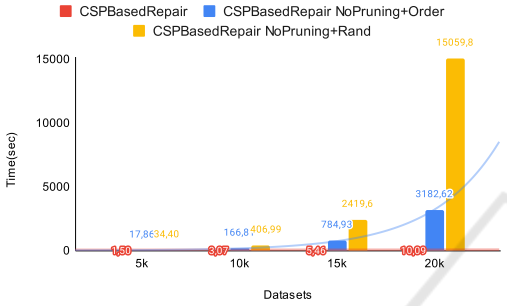


Figure 6: Optimizations impact on the run-time.

have, the larger the gap between the curves becomes, which means that our optimizations helped to reduce the complexity of the problem. We explain this by the fact that domain pruning allowed us to reduce the search space, neighbor pruning reduced the number of comparisons between variables by selecting as few neighbors as possible, and variable ordering helped us to converge quickly to the solution by avoiding dead ends and thus backtracks.

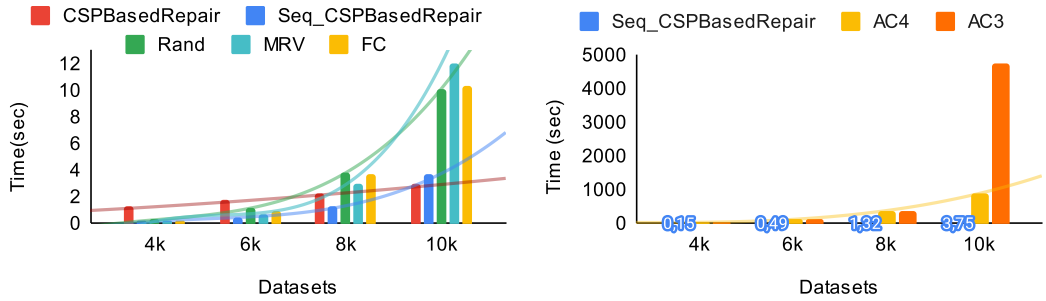
5.3 Comparison to Baselines

We compared the accuracy of the repair process of CSP-DC and Baran (Mahdavi and Abedjan, 2020) a recent data cleaning system using one of AI techniques (transfer learning). We used a sample of 20k records from the Hospital dataset and verified the error rate to assess and compare its impact on both solutions. As Baran may return different results in each run, ten runs were averaged to calculate the metrics. As shown in Table 1, CSP-DC performed better than Baran in both cases. Indeed, we achieved 100% accurate repairs and resolved all violations which is due to: 1) The high quality repairs applied before the CSPBasedRepair. 2) The strength of CSP formulation which allowed us to apply 100% consistent and accurate repairs as it considers all constraints at the same time when looking for a possible update. Baran on the other hand had an unexpected behaviour as it achieved a higher F1 score when the error rate was lower. This also means that Baran was not negatively affected by the error rate because Hospital is a context-rich dataset and Baran makes use of the contextual information available in the dataset which allows it to exploit the remaining trustworthy data. Although Baran returned a rather high F1 score, the precision scores show that about 17% (resp. 7.5%) of the data were not correctly updated when the error rate is 1% (resp. 10%). This could be explained by the high number of possible fixes returned by its different models interacting holistically, which makes it difficult to choose the good repair value. Also, Baran is a general cleaning system, it looks for repairs to multiple types of errors rather than focusing on improving the data consistency, consequently, some violations may not be repaired correctly.

Table 1: Comparing the repair accuracy of CSP-DC and Baran.

E	System	P	R	F1
1%	CSP-DC	1.0	1.0	1.0
	Baran	0.83	0.83	0.83
10%	CSP-DC	1.0	1.0	1.0
	Baran	0.92	0.92	0.92

We study effectiveness and run-time results of the CSPBasedRepair algorithm (CSP-BR) when using our proposed variables ordering technique compared to its performances when using: a random selection or the MRV (Minimum Remaining Value) heuristic as well as filtering strategies : Forward checking (FC) and two optimized versions of Arc-consistency: AC3 and AC4. As MRV performs better in a sequential execution, all algorithms were executed in a sequential manner for a fair comparison. Note that the heuristic and filtering strategies are used with the CSPBasedRepair and after the domains and neighbors pruning. The reported time represents the repair search time without the problems' Initialization (which took at most 5 seconds).



(a) MRV and FC impact on the runtime.

(b) AC3 and AC4 impact on the runtime.

Figure 7: Variables ordering and Filtering strategies impact on the run-time.

Table 2, reports the F1 score, the number of backtracks (BT), and the run-time results. We used 10k records and injected 1% errors from its domain. Results show that CSPBasedRepair not only performs as well as when using MRV and filtering strategies but also is at least 3 times faster. Thanks to our good variable ordering, the purpose of avoiding dead ends was better achieved by our solution (0 backtracks) compared to filtering strategies and MRV which backtracked 217 times.

Table 2: Impact of heuristics and filtering algorithms on the number of backtracks and the run-time.

	CSP-BR	Rand	MRV	FC	AC4	AC3
F1	1.0	1.0	1.0	1.0	1.0	1.0
BT	0	39	217	39	34	39
T	3.7	10.1	12.0	10.3	884	4747

To assess the impact of the dataset size on the run-time and the complexity of the problem, we injected 1% domain errors and varied the size of the hospital dataset from 4k to 10k. We didn't go beyond 10k because MRV, AC3 and AC4 were taking too long and did not terminate repairing 20k records within 24 hours. For a better visualization of the curves, we report the run-time of AC3 and AC4 in a separate figure (Figure 7b).

Figure 7a shows that our sequential version is also fast, the larger the dataset is, the larger the gap between curves becomes. Indeed, when the dataset size is small (4k and 6k), we notice that the repair time when using MRV and FC is almost the same compared to CSPBasedRepair when using our variables ordering strategy. However, once dealing with larger datasets, their run-time increases considerably. The same phenomena is noticed with the random variables selection.

Filtering the domains aims to reduce the research space and converge rapidly toward the solution. Unfortunately, their complexity significantly affected the run-time, especially AC3 and AC4 (Figure 7b). This is due to the fact that for each variable's value, constraints are verified in order to eliminate values that don't satisfy at least one of them. As our problem is composed of a high number of variables with large domains, the complexity of the problem increases when using such a filtering strategy.

Since our solution is parallelizable, we reported the parallel run-time Figure 7a. We notice that when the dataset is small the parallel version of CSP-BasedRepair was slower than most of configurations. However, as the size of the dataset increases, it outperforms all the competing baselines and the run-time difference becomes significant when reaching 10k records. We also notice that the use of parallel execution allowed us to have a linear curve in contrast to the sequential one.

6 RELATED WORK

Existing constraint-based data cleaning approaches have been focusing on finding updates that minimally change original data and satisfy a set of ICs (Xu Chu et al., 2013), (Geerts et al., 2013), (Dallachiesa et al., 2013), (Khayyat et al., 2015), etc. However, the minimal repair doesn't guarantee the accuracy of repairs nor the consistency of data especially, when unverified fixes are applied (Fan et al., 2012). To overcome these problems, we automatically repair only easy repair cases. We also prefer to use one of the existing repair algorithms that exploits external data like master data and knowledge bases to return relevant fixes (Chu et al., 2015), (Yakout et al., 2011), (Geerts et al., 2013). In order to improve the repair quality, users have been involved in the cleaning process to choose a solution among multiple ones (Yakout et al., 2011), to propose repairs for marked cells (Geerts et al., 2013), to give feedbacks on proposed modifications (Yakout et al., 2011) or for crowdsourcing (Chu

et al., 2015). In our work, the user is involved to clean only dirty data, without any automatically generated repairs. The user intervention in this case, guarantees the consistency of data as it resolves all the remaining violations with no proposed possible fixes.
AI techniques have been widely used for data cleaning (Rekatsinas et al., 2017), (Konda et al., 2016), (Yakout et al., 2011), (Krishnan et al., 2017), (Volkovs et al., 2014). Some of them, use machine learning to generate automatic repairs (Yakout et al., 2011), (Mayfield et al., 2010) and/or leverage users feedback using active/reinforcement learning (Yakout et al., 2011), (Berti-Equille, 2019), (Gokhale et al., 2014).Others learn from probabilities extracted from data to predict repairs (Rekatsinas et al., 2017), (Yakout et al., 2011). In our work we exploit AI techniques by formulating our problem as a CSP. We leverage possible repairs returned automatically by a repair algorithm to choose values that guarantee data consistency.

7 CONCLUSION

In this work, we proposed a new data cleaning solution which makes use of the strength of CSP formulation to ensure data consistency and accuracy in a fully automatic way, while also allowing human intervention when necessary. For high quality repairs, we used QDflows as it leverages knowledge bases to perform automatic repairs when possible, or generate possible repairs otherwise. To ensure the consistency in this step, we enable user intervention to manually repair violations with no possible fixes. We also allow multiple cleaning iterations to repair new eventual violations. In order to handle ambiguous repair cases, we annotate the involved cells and collect their possible repairs generated previously, a CSP solving algorithm is then used for a holistic repair. For optimizing the repair search, we propose a new variable selection technique that allows us to reach the solution quickly and avoid dead ends. Our experiments show promising results in improving repair accuracy and data consistency, achieving a F1 score higher than 99% while minimizing human efforts (less than 0.1% for 10% error rate). They also show that our optimizations and the proposed variable ordering technique improve the efficiency of the backtracking search by more than 99.9% and allow it to repair data in a linear time.
Future works may focus on: proposing a new data cleaning approach that provides automatic and consistent repairs before using the CSPBasedRepair, handling larger datasets by using a Big Data processing tool, and automatically discover quality rules from dirty data when they are not available.

REFERENCES

Abdellaoui, S., Nader, F., and Chalal, R. (2017). QDflows: A System Driven by Knowledge Bases for Designing Quality-Aware Data flows. *Journal of Data and Information Quality*, 8(3-4):1–39.

Berti-Equille, L. (2019). Reinforcement Learning for Data Preparation with Active Reward Learning.

Bohannon, P., Fan, W., Rastogi, R., and Flaster, M. (2005). A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification.

Chiang, F. and Miller, R. J. (2011). A Unified Model for Data and Constraint Repair.

Chu, X., Morcos, J., Ilyas, I. F., Ouzzani, M., Papotti, P., Tang, N., and Ye, Y. (2015). 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, Melbourne Victoria Australia. ACM.

Cong, G., Fan, W., Geerts, F., Jia, X., and Ma, S. (2007). Improving Data Quality: Consistency and Accuracy.

Dallachiesa, M., Ebaid, A., Eldawy, A., Elmagarmid, A., Ilyas, I. F., Ouzzani, M., and Tang, N. (2013). NADEEF: a commodity data cleaning system. In *Proceedings of the 2013 international conference on Management of data - SIGMOD '13*, page 541, New York, New York, USA. ACM Press.

Fan, W., Li, J., Ma, S., Tang, N., and Yu, W. (2012). Towards certain fixes with editing rules and master data. *The VLDB Journal*, 21(2):213–238.

Geerts, F., Mecca, G., Papotti, P., and Santoro, D. (2013). The LLUNATIC data-cleaning framework. *Proceedings of the VLDB Endowment*, 6(9):625–636.

Gokhale, C., Das, S., Doan, A., Naughton, J. F., Rampalli, N., Shavlik, J., and Zhu, X. (2014). Corleone: hands-off crowdsourcing for entity matching. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 601–612, New York, NY, USA. Association for Computing Machinery.

Ilyas, I. F. and Chu, X. (2015). Trends in Cleaning Relational Data: Consistency and Deduplication. *Foundations and Trends® in Databases*, 5(4):281–393.

Khayyat, Z., Ilyas, I. F., Jindal, A., Madden, S., Ouzzani, M., Papotti, P., Quiané-Ruiz, J.-A., Tang, N., and Yin, S. (2015). BigDansing: A System for Big Data Cleansing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1215–1230, Melbourne Victoria Australia. ACM.

Konda, P., Das, S., Ardalan, A., Ballard, J. R., Li, H., Panahi, F., Zhang, H., Naughton, J., Prasad, S., Krishnan, G., Deep, R., and Raghavendra, V. (2016). Magellan: Toward Building Entity Matching Management Systems.

Krishnan, S., Franklin, M. J., Goldberg, K., and Wu, E. (2017). BoostClean: Automated Error Detection and Repair for Machine Learning. arXiv:1711.01299 [cs].

Mahdavi, M. and Abedjan, Z. (2020). Baran: effective error correction via a unified context representation and transfer learning. *Proceedings of the VLDB Endowment*, 13(12):1948–1961.

Mayfield, C., Neville, J., and Prabhakar, S. (2010). ERACER: a database approach for statistical inference and data cleaning. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 75–86, New York, NY, USA. Association for Computing Machinery.

Mouelhi, A. E., Jégou, P., Terrioux, C., and Zanuttini, B. (2013). Sur la complexité des algorithmes de backtracking et quelques nouvelles classes polynomiales pour CSP.

Pang, W. and Goodwin, S. D. (2003). A Graph Based Backtracking Algorithm for Solving General CSPs. In Xiang, Y. and Chaib-draa, B., editors, *Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pages 114–128, Berlin, Heidelberg. Springer.

Poole, D. L. and Mackworth, A. K. (2010). *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press. Google-Books-ID: B7khAwAAQBAJ.

Raman, V. and Hellerstein, J. M. (2001). Potter's Wheel: An Interactive Data Cleaning System.

Razgon, I. (2006). Complexity Analysis of Heuristic CSP Search Algorithms. In Hnich, B., Carlsson, M., Fages, F., and Rossi, F., editors, *Recent Advances in Constraints*, Lecture Notes in Computer Science, pages 88–99, Berlin, Heidelberg. Springer.

Rekatsinas, T., Chu, X., Ilyas, I. F., and Ré, C. (2017). HoloClean: holistic data repairs with probabilistic inference. *Proceedings of the VLDB Endowment*, 10(11):1190–1201.

Rezig, E. K., Ouzzani, M., Aref, W. G., Elmagarmid, A. K., Mahmood, A. R., and Stonebraker, M. (2021). Horizon: scalable dependency-driven data cleaning. *Proceedings of the VLDB Endowment*, 14(11):2546–2554.

Volkovs, M., Fei Chiang, Szlichta, J., and Miller, R. J. (2014). Continuous data cleaning. In *2014 IEEE 30th International Conference on Data Engineering*, pages 244–255, Chicago, IL. IEEE.

Wang, J. and Tang, N. (2017). Dependable data repairing with fixing rules. *Journal of Data and Information Quality (JDIQ)*, 8(3-4):1–34.

Xu Chu, Ilyas, I. F., and Papotti, P. (2013). Holistic data cleaning: Putting violations into context. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 458–469, Brisbane, QLD. IEEE.

Yakout, M., Elmagarmid, A. K., Neville, J., Ouzzani, M., and Ilyas, I. F. (2011). Guided data repair. *Proceedings of the VLDB Endowment*, 4(5):279–289.