Navigating the Data Lake with DATAMARAN: Automatically Extracting Structure from Log Datasets

Yihan Gao
University of Illinois at Urbana-Champaign
Urbana, Illinois
ygao34@illinois.edu

Silu Huang
University of Illinois at Urbana-Champaign
Urbana, Illinois
shuang86@illinois.edu

Aditya Parameswaran
University of Illinois at Urbana-Champaign
Urbana, Illinois
adityagp@illinois.edu

ABSTRACT

Organizations routinely accumulate semi-structured log datasets generated as the output of code; these datasets remain unused and uninterpreted, and occupy wasted space—this phenomenon has been colloquially referred to as *"data lake"* problem. One approach to leverage these semi-structured datasets is to convert them into a structured relational format, following which they can be analyzed in conjunction with other datasets. We present DATAMARAN, an tool that extracts structure from semi-structured log datasets with no human supervision. DATAMARAN automatically identifies field and record endpoints, separates the structured parts from the unstructured noise or formatting, and can tease apart multiple structures from within a dataset, in order to efficiently extract structured relational datasets from semi-structured log datasets, at scale with high accuracy. Compared to other unsupervised log dataset extraction tools developed in prior work, DATAMARAN does not require the record boundaries to be known beforehand, making it much more applicable to the noisy log files that are ubiquitous in data lakes. DATAMARAN can successfully extract structured information from all datasets used in prior work, and can achieve 95% extraction accuracy on automatically collected log datasets from GitHub—a substantial 66% increase of accuracy compared to unsupervised schemes from prior work. Our user study further demonstrates that the extraction results of DATAMARAN are closer to the desired structure than competing algorithms.

ACM Reference Format:
Yihan Gao, Silu Huang, and Aditya Parameswaran. 2018. Navigating the Data Lake with DATAMARAN: Automatically Extracting Structure from Log Datasets. In *Proceedings of 2018 International Conference on Management of Data (SIGMOD'18).* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3183713.3183746

1 INTRODUCTION

Enterprises routinely collect semi-structured or partially structured log datasets in shared file systems such as HDFS. These datasets are typically generated automatically as log datasets output by programs, and often number in the billions, e.g., Google has 26B datasets in their shared file system [30]. This phenomenon of accumulation of log datasets within enterprises has recently been referred to as the *"data lake" problem* [29, 54, 55]. Unfortunately, the datasets in a data lake often remain *unused*, *unstructured*, and *uninterpreted*, and as they accumulate, they become *unmanageable*—recent work has characterized this data lake problem as one of the most important challenges facing large enterprises today [46, 54].

The first step to making these log datasets more useful is to convert them into a structured (relational) format. Once we have structured these datasets, we can then infer relationships across datasets, and use them to aid analysis, search, or browsing [12, 13, 22, 38, 48, 58, 62, 64]. The goal of this paper is to *automatically, efficiently, and accurately extract structure from log datasets*, enabling us to tap into the log datasets in large enterprise data lakes.

Why Not Use Prior Work? Given the vast volumes of related work on information extraction [47], one may be tempted to ask: doesn't that solve the problem? Unfortunately, as we will describe in more detail in Section 7, much related work on general HTML wrapper induction, e.g., [21, 31, 32, 41–43], HTML list-based extraction, e.g., [28, 39], and others, e.g., [37, 49], requires training examples or a corpus of entities to be provided. A relatively smaller body of work exists on unsupervised extraction, from general HTML pages [8, 51, 52], and HTML lists [15, 18, 63]. The former crucially relies on the HTML DOM tree, opting to identify recurrent tree patterns; and the latter relies on having each list item corresponding to a record. Log datasets unfortunately do not correspond to a tree structure and records in log datasets are often of multiple types, and span multiple lines, making it hard to identify record boundaries. Moreover, records are interspersed noise or other formatting, making it hard to apply the HTML list techniques. Finally, unsupervised extraction techniques designed for other media, e.g., network protocols, or natural language corpora [7, 14, 17, 53], crucially rely on characteristics of the datasets they are targeting, and are not applicable to log dataset extraction.

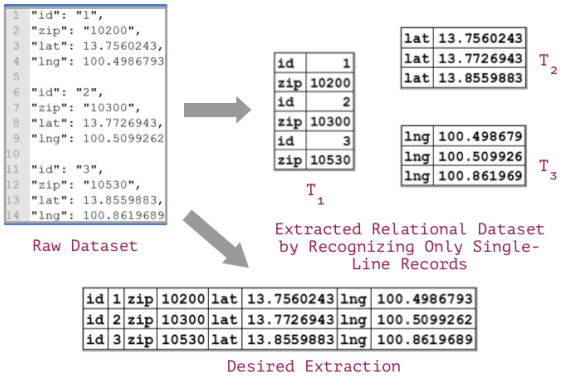


Figure 1: Sample multi-line record dataset, along with the extraction results of line-by-line extraction schemes.

SIGMOD'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3183746

Perhaps the most related body of work is on log dataset extraction itself. Work from program synthesis has developed techniques to perform extraction or transformation from examples [24, 26, 33, 36], while some others [34, 44] require the users to provide the transformation steps; instead, we are opting for a fully unsupervised approach. Fisher et al. [20] take one step towards automation by only requiring that users provide record boundaries: they assume that the data is already *chunked* (i.e., partitioned into small blocks such that each block contains exactly one record) beforehand using external tools. This chunking step is assumed to be a simple form of supervision (e.g., when each record contains exactly k lines), and their work primarily focus on learning structure given the blocks. Recordbreaker [3] is a simple automated implementation of Fisher et al.'s technique that assumes that each record occupies exactly one line. As we will see below, this is far too drastic an assumption to retain applicability in a data lake scenario.

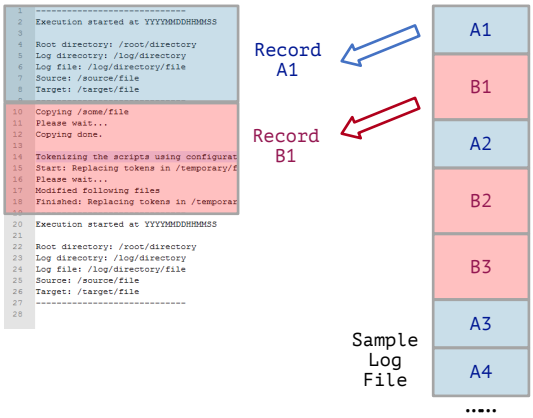


Figure 2: Sample log dataset from GitHub with contents anonymized; only first few lines are shown.

Example 1 (Importance of Recognizing Multi-line Records). *Consider the example log dataset[1] in Figure 1, where each record occupies multiple lines. One promising approach to extract from such a dataset is to use an unsupervised extraction scheme such as RecordBreaker [3] that extracts contents from each line independently, resulting in tables T_1, T_2, T_3 as displayed, such that a multi-line record can be viewed to be a union of multiple single-line records. While such a line-by-line extraction approach indeed can potentially "extract" all relevant content, the associations between records are completely lost in the generated tables. The loss of record association information makes it very hard for users to interpret or utilize such results, say, for example, for keyword search or data integration, both of which require join paths to be preserved. On the other hand, our tool,* Datamaran, *described next, extracts multi-line records correctly as single records.*

Example 2 (Importance of Recognizing Multiple Record Types). *Consider an example log dataset crawled from GitHub in Figure 2, in which there are two types of records (A and B) consisting of 7 and 9 lines respectively, randomly interspersed with each other. Since the sequence of record types can be arbitrary, it is no longer possible to identify the boundaries of records using simple rules, rendering prior unsupervised log structure extraction algorithms non-applicable[2].*

[1]The example dataset in this figure is a simplified version of the Thailand district info dataset, which is one of the 25 datasets used in our evaluation in Section 5 and in our user study in Section 6.

[2]Note, although that while in this example, the boundaries of records are represented as the special "———" lines, an unsupervised chunker cannot utilize this information without human guidance

Datamaran**: Automatic Log Structure Extraction.** In this paper, we present Datamaran[3], an automatic log dataset structure extraction algorithm. At a high-level, the idea behind Datamaran is simple: Datamaran identifies the correct structure of the dataset by looking for repeated patterns: we examine small portions of the dataset and use a hash-table to find repeated patterns covering a significant fraction of the dataset. All such patterns are then evaluated via some scoring function, such as the minimum description length [10] (Note, however, that Datamaran is general, and can adapt to any scoring modality, not just minimum description length). Finally, the best pattern is used to actually extract structured information from the dataset.

However, a naive implementation of this algorithm, as we will demonstrate, can lead to a huge blowup in the number of patterns considered, and therefore the time taken for extraction; as a result, Datamaran requires careful design and engineering to bound the computation at each step. We developed techniques to address the following challenges we encountered when applying the above high-level idea on log datasets:

- *Unknown Record Endpoints.* As described above, identifying the boundaries of records is not straightforward; while the end-of-line character '\n' is often used for separating records, it could also appear within records (i.e., multi-line records).
- *Unknown Field Endpoints.* When trying to detect repeated patterns, it is necessary to separate the formatting characters from the field values. This is not as easy in log datasets, due to the fact that commonly used formatting characters (e.g., the space character ' ') can also appear within field values (e.g., text fields).
- *Complex Structure.* There are often complex structures within records: e.g., if a record contains a list of values, the number of values can vary from record to record, which makes even the underlying formatting vary between records, and therefore the same pattern not applying across the dataset. Indeed, like our example demonstrates, multiple record types may also exist within the dataset. Furthermore, substructures could also exist within the structures via nesting. This makes detecting repetitive patterns substantially more difficult.
- *Redundant Structure.* During the early stages of extraction we often find a number of different repetitive patterns; of which most are completely useless (e.g., the trivial pattern that extracts the entire dataset). The number of such patterns can blow up very quickly as the structure becomes complex: for example, the date component `YYYY-MM-DD` can be identified as either a single field or three different fields, and different combinations of such kind of choices would yield exponentially many patterns. We need an efficient method for filtering out most of the low-quality patterns without evaluating them.
- *Structure Semantics.* Structure extraction is not simply about identifying patterns that can partition the identified records into formatting components (or delimiters), and various pieces of information to be extracted, as the ultimate goal is to transform the log datasets into structured relational datasets. Finding an appropriate structure for this purpose (i.e., making sure that resulting structured datasets are interpretable to users) requires not only a good scoring metric, but also well-designed structure refinement techniques.

[3]Catamaran is a type of boat or raft meant to rapidly navigate large water bodies, such as lakes or oceans.

Overall, DATAMARAN can automatically extract structure from log datasets without any human supervision. Compared to unsupervised adaptations of semi-supervised structure extraction systems [3, 20], DATAMARAN makes fewer assumptions regarding the structure of the dataset, and therefore is much more applicable towards extracting from log datasets: as shown in our experimental evaluation, DATAMARAN can successfully extract structure from all of the datasets used in Fisher et al.'s work [20], and can achieve 95% extraction accuracy on automatically collected log datasets from GitHub, while RecordBreaker [3] can only achieve 29% extraction accuracy on the same dataset collection—*a substantial 66% increase*. DATAMARAN is also efficient and scales well to large datasets: the average running time for small datasets (< 50MB) is less than 20 seconds; even for datasets of size more than 100MB, DATAMARAN can still complete extraction within a few minutes. The main time spent by DATAMARAN for large datasets is in extraction (which is eminently parallelizable), and identifying an appropriate structure can be done much faster. Via a user study, we demonstrate that DATAMARAN is able to generate near-perfect extraction results, compared to the output of RecordBreaker or supervised extraction on the raw dataset, especially when dealing with real log record datasets with noise. Our study indicates that DATAMARAN can be a useful starting point for supervised extraction as well, beyond the applicability to large data lakes.

Paper Outline. The rest of this paper is organized as follows:

- In Section 2, we formally define the problem of unsupervised structure extraction.
- In Section 3, we identify key assumptions that will help us solve the problem in a tractable manner. We also compare the assumptions made in our work with those in prior works to demonstrate why DATAMARAN is better tailored towards structure extraction from log datasets.
- In Section 4, we present DATAMARAN, our structure extraction algorithm, and analyze its time complexity and correctness.
- In Section 5, we experimentally evaluate DATAMARAN on 25 typical datasets and automatically collected log datasets from GitHub, and demonstrate the efficiency, effectiveness, and robustness of DATAMARAN for log dataset structure extraction.
- In Section 6, we conduct a user study to compare the extraction results of DATAMARAN with RecordBreaker. We show that DATAMARAN can handle different types of datasets well, while RecordBreaker requires substantial user supervision for most multi-line record datasets, especially when there is noise present.

2 PROBLEM DEFINITION

We now formally define the problem of (unsupervised) structure extraction from log datasets and introduce related concepts, starting with the concepts of *record templates* and *instantiated records*:

Definition 2.1 (Record Template/Instantiated Record). A *record template* is a string that contains one or more instances of the field placeholder character—a special type of character, denoted as 'F' in this paper—along with other characters. An *instantiated record* is a string with no field placeholder character. We say an instantiated record *can be generated* from a record template iff it can be constructed by replacing field placeholder characters in the record template with strings containing no field placeholder characters.

Given an instantiated record and a record template, we can now define the concept of *field values* as follows:

Definition 2.2 (Field Values). For any pair of instantiated record R and record template RT, if R can be generated from RT, then the replacement strings in R for the field placeholder characters are called the *field values* of R for RT. When the context is clear, we simply call them the field values of R or just the field values.

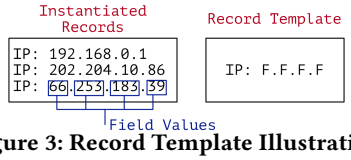


Figure 3: Record Template Illustration

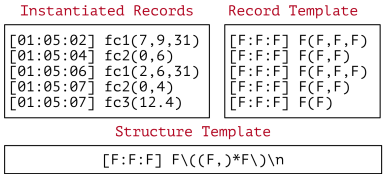


Figure 4: Structural Uncertainty of Record Templates

These definitions are illustrated via an example in Figure 3. As we can see, the instantiated records on the left hand side are generated by replacing the placeholder character `F' in the record template on the right hand side with concrete values. The data items replacing the placeholder character (e.g., 192, 168, 0, 1, . . .) are the field values to be extracted from the dataset.

There are often many record templates that could correspond to a given dataset. Figure 4 illustrates an example wherein the corresponding record templates (i.e., the right hand side) are similar but not exactly the same. To characterize this scenario, we define the concept of a *structure template*:

Definition 2.3 (Structure Template). A *structure template* is a regular expression [50] for record templates. We say the record template *RT can be generated* from the structure template *ST* iff the regular expression of *ST* matches the string form of *RT*.

Intuitively, a structure template captures minor variations in the structure of records within a dataset via a regular expression. The bottom of Figure 4 shows an example structure template corresponding to the records in the top, capturing minor differences in the record templates such as one, two, or three arguments within parentheses. Now, we define the concept of a *log dataset*:

Definition 2.4 (Log Dataset). A log dataset $\mathcal{D} = \{T, S\}$ consists of two components: the textual component T and the structural component S. $S = \{ST_1, ST_2, \ldots, ST_k\}$ is a collection of structure templates, and T is a text dataset with the following structure: T can be partitioned into several blocks separated by the end-of-line character '\n', and each block is either an instantiated record generated from one of the structure templates in S, or corresponds to a noise string with no structure.

Figure 5 illustrates an example log dataset. The parts with a gray background are noise blocks, while the other parts are record blocks. Noise blocks have no structure within, and are not relevant to the structure extraction problem. The requirement that blocks are separated by end-of-line characters in Definition 2.4 is reasonable

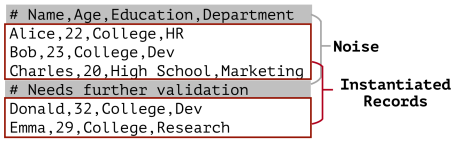


Figure 5: Log dataset illustration

for log datasets: it seems to be a common practice for programmers to write '\n' character at the end of every log line (it holds for every log dataset we have examined). Notice however, that it is not necessary for a record to span just one line, such as the example in Figure 1 or Figure 2; we only require that the structured components and noise are clearly demarcated.

To formalize the structure extraction problem, we start with an intuitive formulation:

Problem 1 (Structure Extraction). *For a log dataset $\mathcal{D} = \{T, S\}$ with only T observed but S unknown, recover S and the field values of the instantiated records in T.*

Note that Problem 1 is not well-posed: for any given text component T, there are infinitely many potential structural components S such that the pair (T, S) obeys Definition 2.4 (for example, the simplest structure template `"F\n"` can pair with any textual component to satisfy Definition 2.4). Most of these structures are unacceptable from an end-user's point of view. In practice, the structure extraction algorithm needs to discover the most plausible one by designing a scoring system that assigns scores to (T, S) pairs. The scoring system is intended to mimic human judgment: a better score implies that the structure is more plausible from an end-user's point of view. We also adopt this approach in Datamaran, and the precise *regularity score function* $F(T, S)$ we use will be discussed later. Thus, an optimization based formulation of the structure extraction problem is as follows:

Problem 2 (Structure Extraction (Optimization)). *For a log dataset $\mathcal{D} = \{T, S\}$ with only T observed but S unknown, find S that optimizes a given regularity score function $F(T, S)$, and extract all the field values of the instantiated records in T.*

3 STRUCTURAL ASSUMPTIONS

In Section 2, we formalized the structure extraction problem as finding the structural component S, i.e., a collection of regular expressions, that best explains or generates the textual component T, i.e., the one that achieves the highest regularity score $F(T, S)$. However, in practice, it is computationally infeasible to search over the entire space of all possible regular expressions. Therefore, it is necessary for structure extraction systems—even semi-supervised ones—to make additional assumptions on the structural component [20, 36]. These assumptions restrict the search space of potential structure templates, thereby serving the following two purposes:

- To enforce human intuition upon the searching procedure. Structure templates following such assumptions are more likely to be the acceptable from an end-user's point of view. In particular, log files have a regular repeating structure, since they were generated by a computer program and meant to contain all relevant information to be extracted via a computer program or script. Our assumptions serve to codify these principles.
- To reduce the complexity of search space of the structural component, making the structure extraction problem more tractable.

In Datamaran, we make three assumptions regarding the structure of the dataset, described next. The validity of these assumptions will be verified in Section 5.3. We will also compare these assumptions with the ones made by RecordBreaker [3] at the end of this section.

3.1 Coverage Threshold Assumption

Here is the first assumption, which is very intuitive:

Assumption 1 (Coverage Threshold). *The coverage of every structure template $ST_i \in S$ should be at least α%. The coverage of structure template ST is defined as the total length (i.e., total number of characters) of the instantiated records of ST.*

Explanation. Assumption 1 states that log datasets don't typically contain a large number of different structure templates within, and thereby each structure template should cover a significant portion of the dataset. Note that a structure template is itself a regular expression that can capture a multitude of record templates, so this is not a severe restriction. The coverage threshold assumption allows us to prune out most unreasonable structure template candidates. We will discuss the impact of varying the parameter α in our experiments.

3.2 Non-Overlapping Assumption

The second assumption we make is the following:

Assumption 2 (Non-Overlapping). *For any structural template ST and any character c, one of the following is true:*

- *for any record template RT generated from ST, $c \notin RT$.*
- *for any instantiated record R generated from ST, no field values of R contains c.*

Explanation. Assumption 2 states that the formatting characters of records cannot be mixed with field values. Intuitively, this makes sense because in practice these records are manually extracted via scripts, and these scripts use delimiters to extract field values.

To formally explain this, we first define some notation: we let *RT-CharSet* denote the set of characters in record templates, while *F-CharSet* denotes the set of characters in field values. Under this notation, Assumption 2 can be simply stated as: For any structure template ST, there exists two disjoint character sets $A(ST)$ and $B(ST)$, such that for any instantiated record R of ST, we have *RT-CharSet*$(R) \subseteq A(ST)$ and *F-CharSet*$(R) \subseteq B(ST)$. In this paper, we further assume that *RT-CharSet* contains only special characters. In other words, we predefine a collection of special characters *RT-CharSet-Candidate*, and assume that *RT-CharSet*$(R) \subseteq$ *RT-CharSet-Candidate* for all records R.

Assumption 2 plays an important role: it allows us to extract the record template directly from an instantiated record given the corresponding character set of the record templates, and efficiently extract matches for a given structure template from the dataset.

Justification of Assumption. Assumption 2 is a relatively strong assumption. To compensate for this, the structural form assumption in Section 3.3 (discussed next) is sufficiently flexible such that even for many datasets that seemingly violate this assumption, we can still get reasonable results.

For example, consider the record template `F,"F",F`. If the field value surrounded by the quotes contains the comma character, then Assumption 2 would be violated. However, Datamaran will still be able to recognize several different record templates in the following, depending on the number of commas in the middle field value:

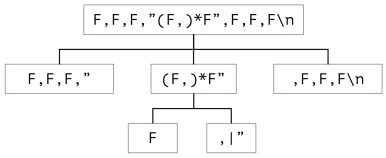


Figure 6: Structural Form Assumption

`F,"F",F` `F,"F,F",F` `F,"F,F,F",F`

Since all the above record templates can be generated from the same structure template `F,"(F,)*F",F`, they will still be recognized as the same record type. We formalize the space of structure templates next.

3.3 Structural Form Assumption

The following assumption restricts the forms of structure templates:

Assumption 3 (Structural Form). *Every structure template is a regular expression that has one of the following forms:*

(1) Array: `({regexA}x)*{regexA}y`
where `{regexA}` *is another regular expression satisfying Assumption 3, and* `x` *and* `y` *are different characters.*

(2) Struct: `{regexA}{regexB}{regexC}....`
where `{regexA}{regexB}{regexC}....` *is a sequence of regular expressions, and each of them is either a simple string or another regular expression satisfying Assumption 3.*

Explanation. Assumption 3 states that records in log datasets are laid out from left to right, with nesting. Formally, the Array-type regular expression is intended to characterize lists of objects. For example, the structure template `[F,F,F,...,F]` can be represented by a prefix `[` and an array-type regular expression `(F,)*F]`. Thus, Assumption 3 essentially states that each structure template must follow a special tree-style structure. An example tree structure for the structure template `F,F,F,"(F,)*F",F,F,F\n` is illustrated in Figure 6. As we can see, the root node in this tree is a Struct node, with three children nodes (level 2 in Figure 6). The second node in level 2 in Figure 6 is an Array regular expression node that has two children nodes (level 3 in Figure 6): the left child is the regular expression part, and right child is the terminating character part.

We can store all of the extracted records in a relational format based on the tree-structure in Assumption 3. Figure 7 demonstrates this procedure: the instantiated records on the left hand side are generated from the structure template in Figure 6, and the right hand side depicts two representations for the relational dataset: one, a normalized relational format, and the other, a denormalized format that uses arrays. As we can see, for the normalized format, each field-placeholder character 'F' in the structure template corresponds to one column in the relational dataset, and the correspondence between non-leaf nodes and their parents are captured using foreign-key references. Datamaran can generate either representation, both of which contain all of the extracted information, and can be utilized by downstream applications.

Our template language in Assumption 3 is basically the same as in LearnPADS [20], except that we do not use a union type-constructor. However, compared to LearnPADS, our definition of a log dataset in Definition 2.4 and the corresponding problem formulation is novel: in Definition 2.4, we defined a log dataset as concatenation of instantiations of multiple types of structure templates plus potentially heavy noise. In contrast, LearnPADS assumes the log dataset to be a well-defined list of chunked records. As described earlier, a key difference is that we no longer assume that record boundaries are known beforehand. This difference leads to a very different algorithmic solution as we shall see next.

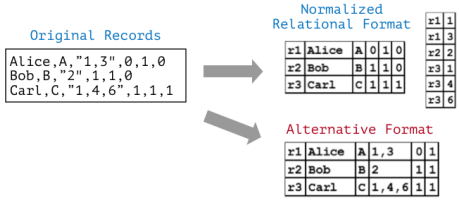


Figure 7: Extracted Relational Dataset

3.4 Assumption Comparison

Here we compare the assumptions made in Datamaran with those in RecordBreaker [3]. The structural form assumption (Assumption 3) has an equivalent counterpart in RecordBreaker. RecordBreaker also makes a stronger version of Assumption 2, together with another additional assumption regarding record boundaries:

Assumption 4 (Boundary). *The boundaries of records can be easily identified beforehand.*

Assumption 5 (Tokenization). *Each record can be tokenized beforehand, such that each token is either part of a field-value, or part of the structure template. In other words, in addition to Assumption 2, it is further assumed that* RT-CharSet *for all records are predetermined in advance*[4]*:* $\forall R$, *RT-CharSet*(R) = *RT-CharSet-Candidate.*

Assumption	RecordBreaker	Datamaran
Coverage Threshold	No	Yes
Non-overlapping	Yes	Yes
Structural Form	Yes	Yes
Boundary	Yes	No
Tokenization	Yes	No

Table 1: The Assumption Comparison Chart

Table 1 compares the assumptions in RecordBreaker and Datamaran. As discussed in the introduction, the two additional assumptions in RecordBreaker are rather restrictive for log datasets. This is further verified in our experiments: about 31% of the log datasets we automatically collected from GitHub (details in Section 5.3) do not satisfy these assumptions. In comparison, the additional assumption made in Datamaran is much milder: due to the coverage threshold assumption, we will only extract from "popular" structure templates rather than all of them. In most practical settings, such a restriction wouldn't cause any problems.

4 THE DATAMARAN ALGORITHM

In Section 2, we defined the structure extraction problem as the problem of finding the structural component S that optimizes a given regularity score function $F(T, S)$ given the observed textual component T. Recall that T has the following form (Definition 2.4):

$$T = B_1 \backslash nB_2 \backslash n \ldots \backslash nB_n$$

where each block B_i is either a noise block or an instantiated record generated from one of the structure templates in S. Due to the extremely large search space of structure templates described in Assumption 3, exhaustive search is not an option, and it is necessary to use the information within T while searching for potential structure templates.

[4]The only difference between Fisher's algorithm [20] and RecordBreaker [3] is the treatment of this assumption: Fisher et al. assume that *RT-CharSet-Candidate* is given by the user for each dataset; RecordBreaker compiled a predetermined character set, making their program unsupervised.

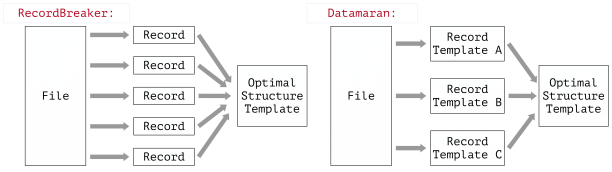


Figure 8: Datamaran **vs. RecordBreaker**

Most prior unsupervised structure extraction algorithms [3, 18, 20] assume that the record boundaries are known beforehand. These algorithms are usually based on the idea of summarization: given all the examples generated from the structure template, the algorithm tries to find the structure template by seeking out the common patterns among records. However, as mentioned previously, the record boundaries within log datasets are usually unknown. which makes these algorithms not directly applicable to log datasets. Furthermore, the task of finding record boundaries is itself also not easy: without knowing the record characteristics first, it is very difficult to pinpoint the exact location of record boundaries, especially with the presence of heavy noise.

Given the difficulty associated with identifying record boundaries, a different approach is used by Datamaran: Datamaran first generates a large collection of structure template candidates directly from the dataset (without actually identifying the record boundaries), and then evaluates the most promising ones to find the optimal structure template. Figure 8 illustrates the conceptual differences between Datamaran and prior approaches such as RecordBreaker [3]. Concretely, Datamaran algorithm consists of the following three steps, as illustrated in Figure 9:

- *Generation.* The first step is to search for candidate structure templates that satisfy the coverage threshold assumption (Assumption 1). To achieve this, we first extract a large collection of structure templates from potential records (i.e., consecutive lines in the dataset), then insert these structure templates into a hash-table to find repeated ones.
- *Pruning.* The second step is to prune out most of the candidates found in the previous step, such that we only need to evaluate the regularity score of a small number of candidates. To achieve this, we designed an *assimilation score function* G, a built-in regularity score function that can be evaluated very efficiently. Intuitively, this assimilation score function tries to filter out all of the redundant structure templates derived by removing some structural details from the true structure templates. We then retain the candidates with highest assimilation score $G(T, S)$ for the final evaluation.
- *Evaluation.* During the final step, we apply two structure refinement techniques to the remaining structure templates after the pruning step, and then evaluate their regularity score to find the one with the highest $F(T, S)$.

The primary algorithmic contributions of Datamaran are the implementations of *generation* and *pruning* step: (a) for the *generation* step, extracting structure templates directly from potential records is highly nontrivial due to the possible variations of field values and record template structures (see Assumption 3), and Assumption 2 plays an important role in this step; (b) for the *pruning* step, the *assimilation score function* requires careful design: it has to be simple enough so that we can evaluate it efficiently, while being effective enough to be able to prune out most of the low-quality redundant candidates. Our final design is based on several iterations, and is not straightforward at first glance.

The details of the Datamaran algorithm will be discussed in the rest of this section: in Section 4.1, we describe the algorithm for efficiently finding structure templates satisfying the coverage threshold assumption (Assumption 1); in Section 4.2, we describe our assimilation score function and discuss the intuition behind its design; in Section 4.3, we describe two structure refinement techniques that are applied during the evaluation step; in Section 4.4, we analyze the time complexity of Datamaran and characterize the conditions under which the correctness of Datamaran can be guaranteed. There are some additional algorithmic details of Datamaran that will not be discussed in this section due to page limitations, and they can be found in the appendix.

The Regularity Scoring Function. In Datamaran, we assume the regularity score function $F(T, S)$ is given, and we can access it through a function call. The design of Datamaran is independent of the choice of this scoring function: we can plug in any reasonable scoring function into Datamaran, and the algorithm would function as before. In this sense, the primary contribution of Datamaran is an efficient and scalable method to optimize any reasonable scoring function.

However, for completeness, we will present the details of the minimum description length [10] regularity score function that we use in our implementation in the appendix, and we demonstrate that it does well empirically in Section 5. That said, through the rest of this section, we assume this function is given and it mimics human judgment regarding the quality of structure templates.

Notation. Table 2 lists the notations used in Datamaran. The first 3 symbols are parameters in Datamaran, while the last 5 symbols represent dataset-dependent values. We will describe each of these parameters later on.

Symbol	Description
M	The number of structure templates retained after pruning
L	The maximum span of records (i.e., the maximum number of lines each record can span)
α	The minimum coverage threshold for records
n	The total number of lines in the dataset
K	The number of structure templates retained after generation
T_{data}	The total size of the dataset
S_{data}	The amount of data sampled during all three steps
c	The number of special characters (i.e., characters in *RT-CharSet-Candidate*) appearing in the dataset

Table 2: Notation Summary

4.1 The Generation Step

In the generation step, we find structure templates satisfying Assumption 1 (i.e., those with at least α% coverage). At a high level, this is achieved by finding repetitive patterns within the dataset. Specifically, Datamaran uses the following five steps to find structure templates with at least α% coverage:

(1) Enumerate possible values of *RT-CharSet* (i.e., the character set in the record templates), and for each such value of *RT-CharSet*, run through steps 2-5.
(2) Enumerate all $O(nL)$ pairs of end-of-line characters `'\n'` that are close to each other (i.e., at most L lines are between them) in the textual component T. For each such pair, treat the content between each pair as an instantiated record, and run steps 3-4.
(3) Extract the record template from the instantiated record using the value of *RT-CharSet*.

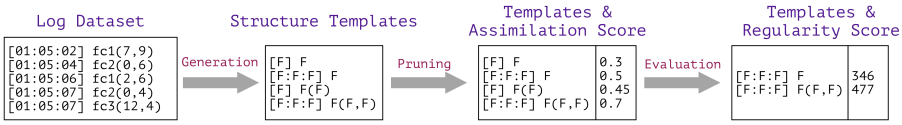


Figure 9: The Workflow of DATAMARAN

(4) Reduce the record template into a structure template (with the form defined in Assumption 3).
(5) Store all of the structure templates generated in step 4 within a hash-table, and then find the ones that satisfy the coverage threshold assumption.

Figure 10 illustrates the workflow of the generation step. The basic idea behind the generation step is very simple: we first enumerate all possible record boundaries (Step 2), then extract structure templates from the contents between them (Step 3, 4), and finally use a hash-table to find the ones with sufficient coverage (Step 5). Assumption 2, which states that $RT\text{-}CharSet \cap F\text{-}CharSet = \emptyset$, is the key assumption that allows us to extract record templates directly from instantiated records (Step 3). Using this assumption, we can separate the field values from formatting characters after enumerating possible values of *RT-CharSet* (Step 1). More details of these steps, together with pseudo-code, can be found in Section 9.1 in the appendix.

The search procedure of generation step is *complete*: since we are enumerating all possible record boundaries, all occurrences of the true structure template will be accounted for. Therefore, the hash-bin associated with the true structure template is guaranteed to have sufficient coverage, which ensures that the true structure template will be among the list of candidate structure templates after the generation step.

Variants of Generation and Sampling Technique: There are two different versions of the first sub-step implemented in DATAMARAN: the exhaustive version enumerates all possible values while the greedy version searches only a subspace of possible values. Intuitively, these two searching procedures represent a trade-off between accuracy and efficiency: the exhaustive search is slower but gives us better extraction results. Additionally, since the running time of generation scales linearly with respect to the dataset size, it may be very expensive for large datasets. We have used a sampling method to ameliorate this. Details of these two techniques can also be found in Section 9.1 in the appendix.

4.2 The Pruning Step

Even with the coverage threshold assumption, there are often far too many structure template candidates remaining after the generation step. As a result, it is impossible to evaluate regularity score $F(T, S)$ for every single one. The purpose of the pruning step is to identify a small promising subset of these candidates to be evaluated in the final evaluation step, and discard the rest.

In the pruning step, we use *assimilation score* $G(T, S)$ to order the structure templates, so that only the best M ones need to be evaluated explicitly in the evaluation step. The assimilation score estimates the amount of data "assimilated" by the structure template (i.e., the amount of data that can be explained by the structure template). Therefore, structure templates with a higher assimilation score are more likely to also have a higher regularity score.

Before we describe the actual design of our assimilation score function, it is helpful to first understand why there are so many structure templates remaining after the generation step. It turns out that most of the redundant structure templates come from two sources as demonstrated in Figure 11: (a) when the structure template consists of multiple lines (line 1-5 in Figure 11 left), any subset of such a structure template would also be captured by the generation step as a legitimate structure template (line 2-4 in Figure 11 right); (b) when the structure template uses multiple types of characters to separate the field values, simpler structure templates can be recognized if some of those characters are treated as field values as illustrated in Figure 11 (bottom).

Therefore, a good assimilation score should be able to distinguish both types of redundancies, and rank the true structure template(s) higher than the redundant ones. At the same time, it should be relatively lightweight to compute. To achieve this, our first component uses the coverage value of structure templates, since we have already computed it during the generation step. However, while the coverage value can effectively distinguish the first source of redundancy, it is not capable of distinguishing the second one.

To address this shortcoming, we introduce another component into the assimilation score: the *Non-Field-Coverage* term, which is defined as the total coverage of the structure template minus the total coverage of all field values of the structure template (i.e., the total length covered by field values in the instantiated records). This term computes the total coverage achieved by "non-field" characters in the template, and can be effectively used to distinguish the second source of redundancy. The final assimilation score function $G(T, S)$ used in DATAMARAN is the following, which filters out all structure templates with either low coverage or low non-field-coverage.

$$G(T, S) = Cov(T, S) \times Non\_Field\_Cov(T, S)$$

4.3 Structure Refinement

To further improve the extraction accuracy, we developed two techniques to refine the structure templates. These techniques are applied to all of the top M structure templates during the evaluation step: we revise these structure templates, and compare the revised structure templates against the original ones, using the regularity score function, replacing them if the score is improved.

4.3.1 Array Unfolding. During the generation step, all of the records are transformed into minimal structure templates, which allowed us to detect repetitive patterns within the dataset. However, there are cases where the minimum structure template is not the optimal structure template.

For instance, in comma-separated values files (*.csv files), all of the records have the form `"F,F,F,....,F,F\n"` (i.e., a fixed number of field values separated by commas). There are two possible structure templates for these records: the plain struct-type `"F,F,F,....,F,F\n"` and the array-type `"(F,)*F\n"`. The plain struct-type template offers a better semantic interpretation in this case (since it implies that the field values are of different types), and also leads to a better regularity score $F(T, S)$.

More generally, because of the structure template reduction procedure (step 4 in the generation step), when the optimal structure template is not a minimal structure template, only its reduced form will be found during the generation step. To address this, we designed the *array unfolding* technique: for each array-type regular

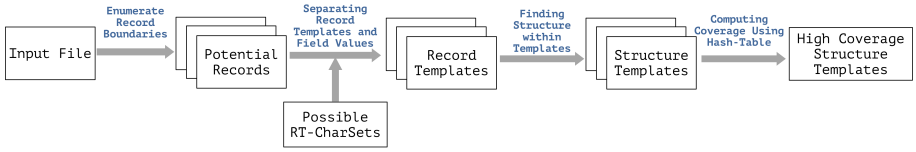


Figure 10: The Generation Step Workflow

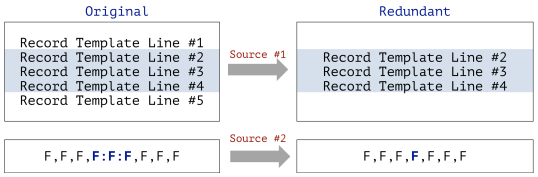


Figure 11: Two sources of redundancies: (1) subsets of multi-line structure templates; (2) structural parts recognized as field values.

expression in the structure template, we attempt to unfold it by expanding it into a struct-type. Figure 12(a) demonstrates this process: the array-type regular expression at the top of the figure will be unfolded into one of the struct-type regular expression at the bottom of the figure. If any of these unfolded structure templates has a better score than the original, the unfolding would be finalized.

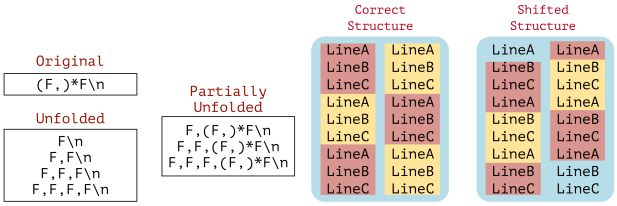


Figure 12: Array Unfolding (left); Structure Shifting (right)

4.3.2 Structure Shifting. Typically, the regularity score function $F(S, T)$ evaluates the quality of structure templates using statistics such as coverage value or minimum description length (see Section 9.1). For most cases, these kinds of score functions can distinguish good structure templates from bad ones. However, there is one ambiguity among structure templates that such a regularity score would fail to detect: the cyclic shifting of structure templates. Figure 12(b) illustrates this: the regularity score $F(T, S)$ of the shifted structure template (right hand side in Figure 12(b)) and the score $F(T, S)$ of the correct structure template (left hand side in Figure 12(b)) are usually approximately equal to each other.

The structure shifting mechanism in Datamaran is designed to distinguish such ambiguities: for each structure template, we consider all possible shifted variants, and then find the position of first occurrence for each one of them. We then pick the one with the earliest first occurrence, which intuitively is most likely the correct structure.

4.4 Theoretical Analysis

4.4.1 Time Complexity. Table 3 lists the time complexity of the three steps in Datamaran respectively[5]. An explanation for the symbols can be found in Table 2. Note that for large datasets, we would utilize sampling for both the generation and evaluation step (details in Section 9.1), and therefore S_{data} is upper-bounded by a large constant. In such cases, the running time of our algorithm is dominated by the actual data extraction procedure.

[5]There are two variants of the search procedure for enumerating RT-CharSet in the generation step, see Section 9.1 for details.

Step	Time Complexity
Generation Step	$O(S_{data}L2^c)$ or $O(S_{data}Lc^2)$
Pruning Step	$O(K \log K)$
Evaluation Step	$O(MS_{data})$
Data Extraction	$O(T_{data})$

Table 3: Time Complexity of the Three Steps in Datamaran

4.4.2 Correctness Guarantee. Datamaran is designed to tolerate noise blocks and variations within record structures and field values. Here we characterize three conditions that are sufficient for guaranteeing the correctness of Datamaran:

Theorem 4.1. *For a log dataset $D = \{T, S\}$ with only T observed, if the following conditions are all met:*

(a) One of the structure templates in S (denote it as ST_0) has the highest coverage and non-field-coverage (defined in Section 4.2) among all structure templates.

(b) For at least α% of the instantiated records, the minimum structure template for them is ST_0.

(c) ST_0 has the best regularity score among all structure templates.

Then Datamaran is guaranteed to return ST_0 as the optimal structure template.

The proof can be found in Section 9.6 in the appendix. For most practical settings, condition (b) is automatically met. Condition (c) requires a carefully designed score function, which is not the focus of this paper. As for condition (a), intuitively it requires the structure templates in S to be sufficiently different from each other, and the field values and noise blocks are sufficiently random. If all of these conditions are satisfied, then Theorem 4.1 would guarantee the correctness of Datamaran.

5 PERFORMANCE EVALUATION

In this section, we experimentally evaluate the performance of Datamaran. The experiments are conducted on two sets of datasets serving different purposes:

- **Manually collected log datasets (Section 5.2).** We collected 25 datasets, including the entire set of 15 datasets used by Fisher et al. [20] and 10 others from various sources (details in Section 5.2). These datasets cover a wide variety of structural formats and possess different characteristics (e.g., file size or structural complexity). We use these datasets to study various properties of Datamaran such as effectiveness, efficiency, parameter sensitivity, and scalability.
- **GitHub log datasets (Section 5.3).** We crawled a collection of 100 log datasets automatically from public GitHub repositories. These datasets reflect the properties of real-world data lakes. We use these datasets to study the properties of data lakes "in the wild", as well as the utility of Datamaran in such settings. This collection of datasets can be viewed as a benchmark for further research.

Datamaran **Settings:** Datamaran is implemented in C++. The default values for the three parameters in Datamaran are: $\alpha = 10\%$ (the coverage threshold parameter); $L = 10$ (the upper bound of

record span); $M = 50$ (the number of remaining structure templates after the pruning step). These default values are used in all of our experiments except for our parameter sensitivity experiments.

RecordBreaker [3] Settings: Despite our best attempts, we were unable to install or run the open-source version of RecordBreaker [3]. Therefore, we decided to faithfully reimplement RecordBreaker in C++ for our comparison. At the first step, RecordBreaker relies on a lexer to break up each record into tokens. We use the open source software Flex [2] as the lexer in our implementation. Accordingly, users need to write a Flex specification file tailored to their dataset in order to obtain a better tokenization scheme. We will compare against RecordBreaker in Section 5.3.

Experiment Settings: All experiments were conducted on a 64-bit Windows machine with 8-core Intel Xeon 3.40GHz CPU and 8GB RAM. All executions are single-threaded.

5.1 Evaluation Criteria

Recall that the structure extraction problem is not well-posed, and the validity of the extracted structure solely depends on the end-user. For many datasets, there are usually multiple structures that can potentially be deemed as valid. For example, the dataset `[01:05:02] 192.168.0.1` has at least the following 4 valid structure templates:

```
[F] F\n                 [F] F.F.F.F\n
[F:F:F] F\n             [F:F:F] F.F.F.F\n
```

Thus, it is not possible to directly compare the extracted structure with a manually labeled structure. In this paper, we define the following evaluation criteria: for each dataset, we first identify several different types of records within the dataset, then identify as many intended extraction targets as possible for each type of record (i.e., observable fields with potentially interesting information). The extraction is considered successful *if both of the following two criteria are met: (a) all of the record boundaries and record types are correctly identified; (b) for each type of intended extraction target, we can select several fields from the structure template, such that all of the intended extraction targets (of this type) can be reconstructed by concatenating the selected fields from the corresponding record.* Figure 13 demonstrates an example successful extraction, in which we have two types of intended extraction targets (i.e., time and IP address), and Datamaran returns the structure template as shown in the middle of the figure. In this example, the extraction is considered successful because both types of intended extraction targets can be reconstructed by concatenating field values at specific positions for all extracted records. If, instead, the targets were extracted together, reconstructing them via concatenation would not be possible.

A more rigorous version of the above evaluation criteria can be found in Section 9.3 in appendix.

5.2 Manually Collected Datasets

The first 15 datasets in this collection come from Fisher et al.'s work [20]. Since Fisher's collection lacks large or complex datasets (i.e., datasets with multiple types of records or multi-line records), we also collected 10 additional datasets from the internet (e.g., the stack exchange data dump [4]) as well as from our genomics collaborators. The sources and characteristics of the 25 manually collected datasets can be found in Section 9.5 in the appendix.

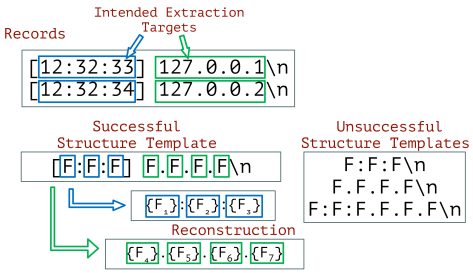


Figure 13: Successful and Unsuccessful Extraction Examples

Evaluation Goal. The goal of the experiments in this section is to study various properties of Datamaran[6]: in Section 5.2.1, we demonstrate the extraction accuracy; in Section 5.2.2, we study the efficiency of Datamaran under various settings. Parameter sensitivity experiments, i.e., the impact of M, L, and α on running time and accuracy can be found in our extended technical report.

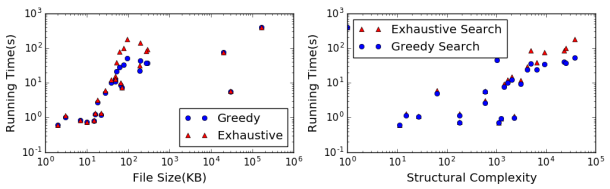
5.2.1 Extraction Accuracy. We used Datamaran to extract structures from the 25 datasets, and the extractions are successful for all 25 datasets based on the evaluation standard in Section 5.1. Datamaran correctly identified the record boundaries for all 25 datasets, without knowing the span of records and the position of noise blocks beforehand. For datasets with multiple types of records, Datamaran can also correctly identify the type of each record. Based on these results, we conclude that Datamaran *is capable of extracting structure from a wide variety of datasets such that end-users could reconstruct any intended target field value using the extracted structures with little extra effort (in most cases no extra effort at all).*

5.2.2 Running Time. We study the efficiency of Datamaran here. We first run Datamaran on the 25 datasets using the default parameters to study the connection between the characteristics of datasets (size/structural complexity) and the running time.

Running Time vs. Dataset Size: Figure 14a depicts the impact of the size of the dataset on the running time of Datamaran (using either exhaustive search or greedy search). The running time on small datasets (less than 50MB) is dominated by the generation and evaluation step. For these datasets, the average running time is 17 seconds for greedy search and 37 seconds for exhaustive search. It takes about 7 minutes for Datamaran to process the largest dataset here (with size 167MB), where the majority of the running time is spent on running the LL(1) parser [23] for the actual data extraction. Note that the running time of the three major steps of Datamaran is not affected by dataset size for large datasets (as discussed in Section 4.4.1). As we can see in Figure 14a, the extraction time is already dominated by the running time of LL(1) parser [23] (which is a necessary step for all structure extraction algorithms) even when the dataset is only moderately large (i.e., about 167MB). Further, this step is easily parallelizable. Therefore, we conclude that Datamaran is efficient enough in practice.

Running Time vs. Structural Complexity: Figure 14b depicts the impact of the structural complexity of the dataset on the running time of Datamaran. The structural complexity of datasets are characterized using the total number of structure templates

[6] We do not compare with RecordBreaker in this section. RecordBreaker employs Fisher's algorithm [20] and all 15 datasets from Fisher's collection are used in this section. Therefore, RecordBreaker will likely perform very well on these datasets, and thus comparison on such datasets would not be objective and meaningful. We will however show that Datamaran can handle all 25 datasets effectively.



(a) (b)

Figure 14: Running Time vs. (a) Dataset Size and (b) Structural Complexity. x axis in (b) is the number of structure templates with at least 10% coverage.

with at least 10% coverage. In general, it takes a longer time for DATAMARAN to extract datasets with higher structural complexity, and the efficiency benefits of greedy search is more significant on these datasets.

5.3 GitHub Datasets

GitHub contains a large quantity of log datasets generated by programmers across the world. We collected 100[7] of such datasets by *uniformly sampling* from the first 1000 search results using the following three criteria: (a) files end with ".log" (b) with length greater than 20000 (c) contains one of the following keywords[8] : "db", "2016", "system", "query", "user". The datasets are sampled using computer-generated random numbers and chosen before any follow-up analysis is conducted, so it represents an unbiased subset of the whole dataset. The characteristics of these datasets are discussed in Section 5.3.1, and the experimental results are discussed in Section 5.3.2. The 100 sampled datasets constitute a new benchmark for structure extraction from log datasets, which will be released to public if this paper is accepted.

Evaluation Goal. The goal of the experiments in this section is to demonstrate the effectiveness of DATAMARAN on common log datasets "in the wild". In Section 5.3.1, we study the characteristics of the log datasets in our sampled collection. In Section 5.3.2, we evaluate the extraction accuracy of DATAMARAN and compare with RecordBreaker [3].

5.3.1 Dataset Characteristics. The sampled datasets are categorized based on three criteria:

- *whether the dataset contains multi-line records*
- *whether the dataset consists of multiple types of records*
- *whether the dataset has any structure at all*

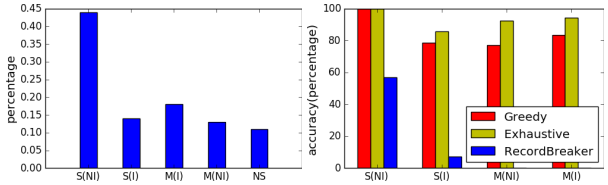
There are five possible labels of datasets based on the above criteria, which are listed in Table 4. The distribution of labels among the 100 sampled log files is shown in Figure 15a.

Label	Description
S (Single-line)	Dataset consists of only single-line records.
M (Multi-line)	Dataset contains records spanning multiple lines
NI (Non-Interleaved)	Dataset consists of only one type of records.
I (Interleaved)	Dataset contains more than one types of records.
NS (No Structure)	Dataset has no structure or its structure does not follow assumptions in Section 3.

Table 4: GitHub Dataset Labels

In the following, we discuss several findings from Figure 15a:

- **Validity of Structural Assumptions:** 89% of datasets follow assumptions in Section 3, and 10% of the datasets has no structure at all (nothing can be extracted from these datasets), only 1% dataset have structure that cannot be described within the framework in Section 3. These statistics suggest that assumptions in Section 3 are well-justified for log datasets.
- **Necessity for Multi-line Record Handling:** 31% of datasets contains at least one type of record spanning multiple lines. The optimal structure in these datasets cannot be successfully extracted if the extraction system cannot handle multi-line records.
- **Necessity for Interleaved Records Handling:** 32% of datasets contains more than one type of records. If the extraction system cannot recognize the existence of multiple types of records, only one type of record can be extracted (the rest will be regarded as noise), resulting information loss.



(a) Characteristics (b) Accuracy

Figure 15: GitHub Datasets: Characteristics and Accuracy

5.3.2 Structure Extraction Accuracy. We applied DATAMARAN to extract structured information from GitHub datasets. Figure 15b shows extraction accuracy for different types of datasets (based on the standard in Section 5.1). Overall, DATAMARAN successfully extracted structure from 85 datasets. The accuracy is 95.5% if we exclude datasets with no structure.

As we can see in Figure 15b, DATAMARAN achieved 100% accuracy on single-line non-interleaved datasets, the simplest type of dataset. The accuracy of DATAMARAN for the other three types of datasets are 85.7%, 92.3% and 94.4% for exhaustive search, and 78.6%, 76.9%, 83.3% for greedy search. Therefore, we conclude that DATAMARAN is effective for most of the log datasets in practice. We also identified major causes for inaccurate extractions, which can be found in Section 9.4 in the appendix.

Figure 15b also shows the extraction accuracy of RecordBreaker [3] with default configurations and parameters for comparison. As we can see, RecordBreaker performs very poorly on log datasets with accuracy 56.8% and 7.1% on S(NI) and S(I) respectively and 0% on M(NI) and M(I), for a total of 29.2% accuracy, which is not very surprising: RecordBreaker is originally designed for well-structured datasets, and cannot handle the noise-heavy log datasets very well. Furthermore, the resulting structure templates depend a lot on the Flex configurations and the tuning of two parameters in RecordBreaker (i.e., MaxMass and MinCoverage). This is because Flex configurations decide the quality of tokenization, while the other two parameters determine the datatype (i.e., *struct*, *array* or *union*) for a given list of records. However, there are no generic configurations or parameter values that work for all datasets, *which makes RecordBreaker less desirable in an unsupervised setting and incomparable to* DATAMARAN.

Figure 15a and Figure 15b also demonstrates why prior work such as RecordBreaker [3] is not well-suited for extracting structure from log datasets: for any dataset containing multi-line records, the task of partitioning such dataset into collection of records is nontrivial (due to the presence of noise & the fact that record span is unknown). From Figure 15a, we see that at least 31%[9] of datasets

[7] The scale is limited to 100 since we have to manually inspect the datasets and the extraction results. DATAMARAN can be automatically applied to thousands of datasets without any problem.

[8] GitHub search function requires at least one search keyword, and we used multiple keywords to improve the diversity of our selection.

[9] This number is an underestimate since Assumption 5 can also be violated in some datasets

cannot be handled by RecordBreaker [3] as demonstrated by M(NI) and M(I) in Figure 15b.

6 USER EVALUATION

To further evaluate the quality of the structure extracted by DATAMARAN, we conduct a user study on five representative log datasets, comparing our results against the raw datasets as well as the extracted results using RecordBreaker.

6.1 Study Design

Our user study simulates the following scenario, where a participant is presented with a log file, and they want to extract some information of interest, prior to analysis. One straightforward way to do so is to import the log file into a spreadsheet tool like Microsoft Excel, and then use Excel functionalities to extract this information. Alternatively, the participant can first use either DATAMARAN or RecordBreaker to extract the structure, and then refine the results using Excel to obtain the desired structure and filter out anything that is not of interest. We will compare these three methods (i.e., from the raw log file, from the result of DATAMARAN/RecordBreaker) in our user study. In order to quantify the manual effort taken to reach the desired extraction result, we create a target extraction result based on our best judgement and use it as our gold standard. For each dataset, we show the raw log file as well as the extraction results of DATAMARAN and RecordBreaker to the participants, and ask them to transform each file into the target structure.

Methodology. The user study consists of three phases, in brief:

(1) Introduction phase: We first show the participant an example of the raw file, extraction results from DATAMARAN and RecordBreaker, along with the target file, denoted as R, A, B and T respectively. Then, we introduce four popular Excel data wrangling functionalities that may be used for transforming those three files into the target file, *Concatenate, Split, FlashFill* and *Offset*. *Concatenate* and *Split* are straightforward; *Flashfill* autocompletes columns from a few user examples [25]; and *Offset* can be used to copy contents every K rows while skipping the $(K-1)$ rows in-between. Overall, *Concatenate*, *Split*, and *FlashFill* are very easy to use, while *Offset* requires more thought and effort and is not very intuitive [1].

(2) Quiz phase: We present five folders to the participant, one for each dataset, where each folder contains the raw file (R), two extraction files (A and B) and the target file (T). One dataset is a single-line record dataset while the other four are multi-line record datasets. For each dataset, the participant is asked to transform R, A, and B into T using the described functionalities in Excel, or any other functionalities they may be aware of. The whole process takes around one and half hours per participant.

(3) Survey phase: We conduct a survey to understand the participant's experience in structure extraction using the raw file R and the two extraction files (A and B).

Participants. We had six participants in our study, including five graduate students from Computer Science and one graduate student from Electrical and Computer Engineering. Four out of the six work with data very often (daily), one often (weekly) and one rarely (yearly or fewer). In addition, every participant has used spreadsheets and scripting language(s), like Python and Matlab, for data analysis, while two participants had also used business analytics tools like Tableau and Power BI.

6.2 Result Analysis

Summary: We find *(a)* Starting from the extracted results using RecordBreaker and DATAMARAN, i.e., A and B, helps the participant "fast-forward" to the desired target structure, compared to the raw file R. *(b)* The extracted result by DATAMARAN is already in a fine-grained clean format, requiring very simple operations, i.e., Concatenate or FlashFill, to concatenate the fine-grained results to get the target format T. *(c)* For multi-line datasets, it is hard to obtain the target information from both the raw file R and the extracted result using RecordBreaker B, as evidenced by the **failures** (black circles) in Figure 16.

For each dataset, we recorded the action sequences performed by each participant during the transformation. In total, there are $6 \times 3 \times 5 = 90$ sequences, since we have six participants, three file types (A, B and R), and five datasets. Each sequence is depicted by a horizontal line in Figure 16, where each colored circle denotes a specific operation[10] performed by the participant, as shown in the legend. The x-axis is the operation's index in the sequence, and y-axis shows the participant id and the file type. For instance, $R.u_1$ refers to the first participant (u_1) and the task is to transform the raw file (R) into the target file.

As shown in Figure 16, participants took more operations to transform the raw file (if no failure occurred) as opposed to extracted files using DATAMARAN and RecordBreaker. This verifies the usefulness of automated extraction tools. Furthermore, participants always took the least number of steps to reach the target file T when using DATAMARAN, with no failure. On the contrary, they were often unable to transform the raw file R and the extracted file using RecordBreaker B, as shown in Figure 16(b,d-e). This occurred mostly when the records span multiple-lines and when the dataset is noisy. Next, we will discuss the findings for each dataset briefly. More details can be found in our technical report [1].

Dataset 1 is a single-line dataset, and the extraction results of both RecordBreaker and DATAMARAN are much better structured than the raw file. Compared to R and B, A took the smallest number of steps in order to be transformed to T, as illustrated in Figure 16(a). When it comes to multi-line record datasets, i.e., datasets 2-5, DATAMARAN exhibits a much more substantial advantage over RecordBreaker and the raw file. First, when there is noise or incomplete records in the dataset (dataset 4 and 5), participants needed to either manually filter the incomplete records one by one, or write some sophisticated code to remove the noise and reconstruct the records. This step is often laborious or hard to implement. Second, RecordBreaker treats each single line as a record unit, and would recognize each line as a different structure, which are then stored into different files. Hence, the participants often found themselves losing context for reconstructing the records when each record spanned multiple files. As a consequence, participants often failed to transform B and R into T after some trials, as shown by the black circles in Figure 16(b,d-e). Due to the context missing in B, participants could only figure out that they failed to reconstruct the rows after a number of operations, as illustrated in Figure 16(e).

6.3 Survey and Interview

Summary: All participants ranked the extracted results by DATAMARAN (A) easiest to use, and the raw file (R) most difficult to use. This is mostly because the structure in the raw file is unclear, while DATAMARAN provides a very clear structure.

[10] We ignore the simple operations like Delete, Copy, Paste.

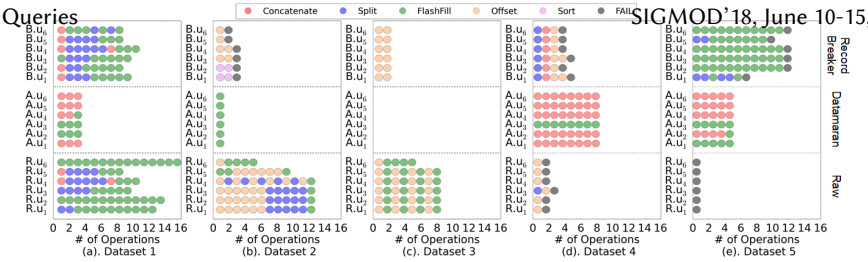


Figure 16: Sequence of Operations for Transformation

Most participants (5/6) reported that *A* (Datamaran) is very easy to use, requiring only merging (i.e., *Concatenate* and *FlashFill*) and deleting operations most of the time. But some participants also complained that *A* still requires a bunch of manual work, like repeating *Concatenate*. This is because the extraction results of Datamaran is of a very fine-grained nature, The large number of repeating operations of *Concatenate* or *FlashFill* is captured in Figure 16(d). On the other hand, all participants (6/6) complained that the raw file is hard to begin with, since it looks messy and is difficult to find the pattern inside. In addition, participants were not satisfied with the extracted results by RecordBreaker, since they were annoyed by the multi-file and multi-line merge operations like *Offset*. On average, participants rated the difficulty of performing transformation from *A*, *B*, and *R* to *T* as 1.8, 7.8, and 9.3 respectively, where 1 indicates the easiest and 10 indicates the hardest.

In particular, one participant (u_4) said the following—"For *A*, it is ready to use, involving mostly merge and delete operations. For *B*, there is lots of extra operations. It's hard to carefully use Offset to merge lines and merging across rows could be painful and error prone. For *R*, it is impossible to do manually. I prefer to write code, but need to make sure the code is bug free." Another participant (u_6) said the following—"No major difficulty for *A*. Each row corresponds to exactly one record. For *B*, there is information lost during processing, hard (impossible?) to join disparate partially processed items together. *R* requires significant manual effort to identify anomalous records before automatic techniques can be applied to put data in structured format." There is also some limitations identified for Datamaran (*A*). One participant (u_1) said the following—"For *A*, it only involves single file operators, easier to track, but still a lot of manual work. For *B*, it requires cross file operations, difficult to track, and sometimes you end up choosing sub-optimal operations. For *R*, it is unstructured, need to create tuple using Offset first, most laborious among the three."

From the user study, we conclude that Datamaran has better extraction results than RecordBreaker, and both tools are a better starting point than the raw file.

Limitations. Since our user study is limited to the comparison between two automated structure extraction tools, i.e., Datamaran and RecordBreaker, and supervised extraction starting from the raw file using techniques like FlashFill, it remains to be seen whether unsupervised tools can perform as well as other more advanced supervised extraction tools. Also, the many concatenate operations (e.g., assembling IP addresses from fragments) can be tedious. For such domain-specific datatypes. Datamaran should be enhanced with type awareness (e.g., for phone numbers, IPs, URLs).

7 RELATED WORK

Our work is related to the vast bodies of work on general information extraction, as well as the more limited work on log dataset extraction, and string transformation. Other related work can be found in Section 9.8.

Unsupervised HTML Wrapper Induction. A few papers attempt to extract from HTML pages directly, without requiring any training examples [8, 51, 52, 57]. All of these papers rely on repetitiveness within a page, or the redundancy across similar pages to separate the content from the template. The rules that are inferred are strongly dependent on the HTML DOM tree structure; in our case, we do not have the luxury of HTML tags to distinguish between records or fields.

Extraction from Web Documents. There has been some work on extraction from other forms of documents, or portions of Web documents, typically leveraging example concepts [49] or a knowledge-base [5, 16, 37, 65] to extract entities and attributes from text files.

List extraction, i.e., extraction from lists on the web is another area that has seen some work [15, 18, 28, 39, 63]. Some of these papers require both the eventual relational schema as well as candidate examples to be provided [28, 39]. Some papers attempt fully-automated list extraction [15, 18, 63]. These papers make the crucial assumption of each record corresponding to a single list item, making it easy to extract the boundaries of the records.

Log Dataset Extraction and Transformation. Wrangler [27, 34] supports the interactive specification of log dataset cleaning operations, drawing from the transformations in Raman et al. [44]. Instead of operator specification, other work relies on user-provided input-output examples [9, 24, 26, 33, 36] to transform one semi-structured dataset to another. The PADS project [20] relies on a user-provided chunker and tokenizer to identify the boundaries of records/field values, while RecordBreaker is a line-by-line unsupervised implementation, with a fixed lexer configuration which makes it inflexible for real log datasets. Recent work by Raza and Gulwani [45] describe an automatic text extraction DSL for single-line extraction, generalizing to both web-pages and text documents.

Other work clusters event logs [40, 56] by treating the lines of the log dataset as data points and assigning them to clusters. Compared to our work, these papers do not attempt to identify the structure within records, and they do not consider the possibility of multi-line records.

8 CONCLUSIONS

We presented Datamaran, a completely unsupervised automatic structure extraction tool specifically tailored towards log datasets. The experimental results demonstrate that Datamaran can efficiently and correctly extract structures from all representative datasets and 95.5% of the GitHub datasets, and is robust with respect to parameter choices, while RecordBreaker can only extract 29.2% from the same dataset collection. Our user study further demonstrates that Datamaran, in addition to automatically extracting from log datasets, provides a valuable starting point for data analysis: all participants (6/6) preferred Datamaran to RecordBreaker and the raw file.

Acknowledgments. We acknowledge support from NSF grants IIS-1513407, IIS-1633755, IIS-1733878 and IIS-1652750 and funds from Adobe, Google, Toyota, and the Siebel Energy Institute.

REFERENCES

[1] Datamaran Technical Report. https://arxiv.org/abs/1708.08905. (????).
[2] Flex: lexical analyzer generator. https://en.wikipedia.org/wiki/Flex_(lexical_analyser_generator). (????).
[3] RecordBreaker: Automatic structure for your text-formatted data. http://cloudera.github.io/RecordBreaker/. (????).
[4] Stack Exchange Data Dump. https://archive.org/details/stackexchange. (????). Accessed: 2017-07-13.
[5] Eugene Agichtein and Venkatesh Ganti. 2004. Mining reference tables for automatic text segmentation. In *KDD*. ACM, 20–29.
[6] Eugene Agichtein and Luis Gravano. 2000. Snowball: Extracting relations from large plain-text collections. In *DL*. ACM, 85–94.
[7] Joao Antunes, Nuno Neves, and Paulo Verissimo. 2011. Reverse Engineering of Protocols from Network Traces. In *Conf. on Reverse Engineering*. IEEE Computer Society, 169–178.
[8] Arvind Arasu and Hector Garcia-Molina. 2003. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 337–348.
[9] Daniel W Barowy, Sumit Gulwani, Ted Hart, and Benjamin Zorn. 2015. FlashRelate: extracting relational data from semi-structured spreadsheets using examples. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 218–228. Issue 6.
[10] Andrew Barron, Jorma Rissanen, and Bin Yu. 1998. The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory* 44, 6 (1998), 2743–2760.
[11] Georges Bossert, Frédéric Guihéry, and Guillaume Hiet. 2014. Towards automated protocol reverse engineering using semantic information. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*. ACM, 51–62.
[12] Michael J. Cafarella, Alon Y. Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. WebTables: exploring the power of tables on the web. *PVLDB* 1, 1 (2008), 538–549.
[13] Kaushik Chakrabarti and others. 2016. Data Services Leveraging Bing's Data Assets. *Data Engineering* (2016), 15.
[14] Shay B. Cohen, Dipanjan Das, and Noah A. Smith. 2011. Unsupervised Structure Prediction with Non-parallel Multilingual Guidance. In *EMNLP*.
[15] William W. Cohen, Matthew Hurst, and Lee S. Jensen. 2002. A flexible learning system for wrapping tables and lists in HTML documents. In *WWW*.
[16] Eli Cortez, Daniel Oliveira, Altigran S da Silva, Edleno S de Moura, and Alberto HF Laender. 2011. Joint unsupervised structure discovery and information extraction. In *SIGMOD*. 541–552.
[17] Weidong Cui, Jayanthkumar Kannan, and Helen J Wang. 2007. Discoverer: Automatic Protocol Reverse Engineering from Network Traces.. In *USENIX Security Symposium*. 1–14.
[18] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. 2009. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment* 2, 1 (2009), 1078–1089.
[19] Oren Etzioni and others. 2004. Web-scale information extraction in knowitall:(preliminary results). In *WWW*. 100–110.
[20] Kathleen Fisher, David Walker, Kenny Q Zhu, and Peter White. 2008. From dirt to shovels: fully automatic tool generation from ad hoc data. In *ACM SIGPLAN Notices*, Vol. 43. ACM, 421–434.
[21] Dayne Freitag and Nicholas Kushmerick. 2000. Boosted wrapper induction. In *AAAI/IAAI*. 577–583.
[22] Hector Gonzalez and others. 2010. Google fusion tables: data management, integration and collaboration in the cloud. In *SoCC*. 175–180.
[23] Dick Grune and Ceriel JH Jacobs. 2007. *Parsing techniques: a practical guide*. Springer Science & Business Media.
[24] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *POPL*. 317–330.
[25] Sumit Gulwani. 2011. Automating string processing in spreadsheets using input-output examples. In *ACM SIGPLAN Notices*, Vol. 46. ACM, 317–330.
[26] Sumit Gulwani, William R. Harris, and Rishabh Singh. 2012. Spreadsheet data manipulation using examples. *Commun. ACM* 55, 8 (2012), 97–105.
[27] Philip J. Guo, Sean Kandel, Joseph M. Hellerstein, and Jeffrey Heer. 2011. Proactive wrangling: mixed-initiative end-user programming of data transformation scripts. In *UIST*. 65–74.
[28] Rahul Gupta and Sunita Sarawagi. 2009. Answering table augmentation queries from unstructured lists on the web. *Proceedings of the VLDB Endowment* 2, 1 (2009), 289–300.
[29] Rihan Hai, Sandra Geisler, and Christoph Quix. 2016. Constance: An intelligent data lake system. In *SIGMOD'16*. ACM, 2097–2100.
[30] Alon Halevy and others. 2016. Goods: Organizing Google's Datasets. In *SIGMOD'16*. ACM, 795–806.
[31] Wei Han, David Buttler, and Calton Pu. 2001. Wrapping Web Data into XML. *SIGMOD Record* 30, 3 (2001), 33–38.
[32] Chun-Nan Hsu and Ming-Tzung Dung. 1998. Generating Finite-State Transducers for Semi-Structured Data Extraction from the Web. *Information Systems* 23, 8 (1998), 521–538.
[33] Zhongjun Jin and Others. 2017. Foofah: Transforming data by example. In *SIGMOD*. ACM, 683–698.
[34] Sean Kandel, Andreas Paepcke, Joseph Hellerstein, and Jeffrey Heer. 2011. Wrangler: Interactive visual specification of data transformation scripts. In *CHI'11*. ACM, 3363–3372.
[35] Nicholas Kushmerick, Daniel S Weld, and Robert Doorenbos. 1997. Wrapper induction for information extraction. (1997).
[36] Vu Le and Sumit Gulwani. 2014. FlashExtract: a framework for data extraction by examples. In *ACM SIGPLAN Notices*, Vol. 49. ACM, 542–553.
[37] Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and HV Jagadish. 2008. Regular expression learning for information extraction. In *EMNLP*. 21–30.
[38] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and Searching Web Tables Using Entities, Types and Relationships. *PVLDB* 3, 1 (2010), 1338–1347.
[39] Ashwin Machanavajjhala, Arun Shankar Iyer, Philip Bohannon, and Srujana Merugu. 2011. Collective extraction from heterogeneous web lists. In *WSDM*. ACM, 445–454.
[40] Adetokunbo AO Makanju, A Nur Zincir-Heywood, and Evangelos E Milios. 2009. Clustering event logs using iterative partitioning. In *KDD*. 1255–1264.
[41] I. Muslea, S. Minton, and C. Knoblock. 1998. STALKER: Learning extraction rules for semistructured, web-based information sources. In *AAAI: Workshop on AI and Information Integration*.
[42] Nilesh N. Dalvi et. al. 2009. Robust web extraction: an approach based on a probabilistic tree-edit model. In *SIGMOD*. 335–348.
[43] P. Gulhane et. al. 2011. Web-scale information extraction with vertex. In *ICDE*. 1209–1220.
[44] Vijayshankar Raman and Joseph M Hellerstein. 2001. Potter's wheel: An interactive data cleaning system. In *VLDB*, Vol. 1. 381–390.
[45] Mohammad Raza and Sumit Gulwani. 2017. Automated Data Extraction Using Predictive Program Synthesis.. In *AAAI*. 882–890.
[46] Janessa Rivera and Rob van der Meulen. 2014. Gartner Says Beware of the Data Lake Fallacy. In *Gartner http://www.gartner.com/newsroom/id/2809117*.
[47] Sunita Sarawagi and others. 2008. Information extraction. *Foundations and Trends in Databases* 1, 3 (2008), 261–377.
[48] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Y. Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding related tables. In *SIGMOD Conference*. 817–828.
[49] Pierre Senellart and others. 2008. Automatic wrapper induction from hidden-web sources with domain knowledge. In *Proceedings of the 10th ACM workshop on Web information and data management*. ACM, 9–16.
[50] Michael Sipser. 2006. *Introduction to the Theory of Computation*. Vol. 2. Thomson Course Technology Boston.
[51] Hassan A Sleiman and Rafael Corchuelo. 2013. Tex: An efficient and effective unsupervised web information extractor. *Knowledge-Based Systems* 39 (2013), 109–123.
[52] Hassan A Sleiman and Rafael Corchuelo. 2014. Trinity: on using trinary trees for unsupervised web data extraction. *TKDE* 26, 6 (2014), 1544–1556.
[53] Valentin I. Spitkovsky, Hiyan Alshawi, Angel X. Chang, and Daniel Jurafsky. 2011. Unsupervised Dependency Parsing Without Gold Part-of-speech Tags. In *EMNLP*.
[54] Brian Stein and Alan Morrison. 2014. The enterprise data lake: Better integration and deeper analytics. *PwC Technology Forecast: Rethinking integration* 1 (2014).
[55] Ignacio Terrizzano, Peter M Schwarz, Mary Roth, and John E Colino. 2015. Data Wrangling: The Challenging Yourney from the Wild to the Lake.. In *CIDR*.
[56] Risto Vaarandi. 2004. A breadth-first algorithm for mining frequent patterns from event logs. *Intelligence in Communication Systems* (2004), 293–308.
[57] Valter Crescenzi et. al. 2001. RoadRunner: Towards Automatic Data Extraction from Large Web Sites. In *VLDB*. 109–118.
[58] Petros Venetis and others. 2011. Recovering Semantics of Tables on the Web. *PVLDB* 4, 9 (2011), 528–538.
[59] Yipeng Wang, Xiaochun Yun, M Zubair Shafiq, Liyan Wang, Alex X Liu, Zhibin Zhang, Danfeng Yao, Yongzheng Zhang, and Li Guo. 2012. A semantics aware approach to automated reverse engineering unknown protocols. In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*. IEEE, 1–10.
[60] Yipeng Wang, Zhibin Zhang, Danfeng Daphne Yao, Buyun Qu, and Li Guo. 2011. Inferring protocol state machine from network traces: a probabilistic approach. In *International Conference on Applied Cryptography and Network Security*. Springer, 1–18.
[61] Sean Whalen, Matt Bishop, and James P Crutchfield. 2010. Hidden Markov Models for Automated Protocol Learning.. In *SecureComm*. Springer, 415–428.
[62] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. InfoGather: Entity Augmentation and Attribute Discovery by Holistic Matching with Web Tables. In *SIGMOD'12*. 97–108.
[63] Yanhong Zhai and Bing Liu. 2005. Web data extraction based on partial tree alignment. In *WWW*. 76–85.
[64] Meihui Zhang and Kaushik Chakrabarti. 2013. InfoGather+: Semantic Matching and Annotation of Numeric and Time-varying Attributes in Web Tables. In *SIGMOD '13*. 145–156.
[65] Chang Zhao, Jalal Mahmud, and IV Ramakrishnan. 2008. Exploiting structured reference data for unsupervised text segmentation with conditional random fields. In *ICDM*. 420–431.

9 APPENDIX

9.1 Other Algorithmic Details

Here we discuss some additional algorithmic and implementation details of DATAMARAN that were not covered in the main body of the paper. Pseudocode can be found in our technical report[1].

Variants of Generation Step. We implemented two searching procedures in DATAMARAN for finding the optimal *RT-CharSet*. Both searching procedures require *RT-CharSet-Candidate*, the set of characters that can potentially be in *RT-CharSet*, as an input.

Suppose there are c different characters in *RT-CharSet-Candidate* that appeared in the dataset. The exhaustive search would enumerate all 2^c subsets. On the other hand, the greedy search procedure would only enumerate $O(c^2)$ of them. The greedy search procedure operates in the following way: initially, *RT-CharSet* is set to be empty; then in each step, one of the characters in *RT-CharSet-Candidate* is added to *RT-CharSet*; the decision for choosing which character to add is made greedily by choosing the character generating the structure template with highest assimilation score (as defined in Section 4.2). An example illustrating the two searching procedures can be found in our technical report [1].

Extracting Record Template From Instantiated Record. The non-overlapping assumption (Assumption 2) states that there exists two disjoint sets of characters A and B, such that for any instantiated record R, *RT-CharSet*(R) (i.e., the record template character set) is a subset of A, and *F-CharSet*(R) (i.e., the field value character set) is a subset of B. By this assumption, the record template can be uniquely extracted from any of its instantiated records given the value of A and B. For example, if A = { ',', '\n' }, then the instantiated record `1,2,3,45,6,78,9,a,bc,d\n` can be transformed into the record template `F,F,F,F,F,F,F,F,F\n` by replacing characters not in A with the field placeholder.

Reducing Record Templates to Structure Templates. We identify the corresponding minimum structure template that can generate each extracted record template. This is achieved by repeatedly reducing repeated patterns into array regular expressions. For example, the record template `F,F,F,F,F ,F,F\n` is reduced into the structure template `(F,)*F\n`. If there are conflicting reduction steps (i.e., reduction steps that cannot be performed simultaneously), we choose one of them arbitrarily. The reduction process only guarantees that we find a minimal structure template (i.e., a structure template that cannot be reduced further), which means that not all instantiated records are reduced back to the same structure template. As a result, the coverage estimate during the generation step is an underestimate. However, in our experiments, the initial coverage estimate is usually still well above the α% threshold, thereby not affecting the correctness of the generation step.

Pruning Using Hash-Table. We store all of the structure templates in a hash-table, and maintain the total coverage of all structure templates associated with each hash-bin. For all hash-bins with less than α% total coverage, the associated structure templates are discarded.

Handling Multiple Structure Templates. In the cases where there are more than one type of record in the dataset, we repeat the entire structure detection process (Generation-Pruning-Evaluation) for multiple times. After each iteration, we retrieve the parts of the dataset that are not explained by the previous structure. These parts are concatenated together, and we run the entire procedure on it again.

Sampling Technique. In the actual implementation of DATAMARAN, sampling is used instead of simply scanning through the entire dataset in both the generation and evaluation step. For large datasets, scanning the whole dataset during these steps is not feasible: the total number of whole dataset scans is equal to the number of *RT-CharSet*s enumerated in the generation step plus M in the evaluation step. Our sampling implementation is cache-aware: we sample several large chunks of data and concatenate them in the memory. Both the generation step and the evaluation steps are performed on the concatenated chunks instead.

9.2 Default Regularity Score Function

We implemented a simple default regularity score function based on the minimum description length principle [10]: we design a record generation procedure from the structure template, and the regularity score is equal to the total amount of information needed for describing all the instantiated records using the structure template, plus the additional information needed to describe the noise blocks. For completeness, we describe the details of this score function in the following. Describing the record using the structure template is straightforward given Assumption 3:

- For arrays, we describe the number of repetitions, then describe each repetition individually.
- For structs, we describe each component individually.
- For fields, the description scheme depends on its value type.

For the field value description, we associate each field in the structure template with one of the following four value-types: enumerated type, integer, real number, or string. The description schemes for field values depend on the data-type—which can be determined by analyzing the field values in the group; the details of these schemes are listed as follows:

- The enumerated type fields are described using $\lceil \log_2 n\_value \rceil$ bits, where n_value is the total number of unique values.
- The integer fields are described using $\lceil \log_2(max\_value - min\_value + 1) \rceil$ bits, where max_value and min_value are the upper bound and lower bound of the field value, which can be determined by scanning through the dataset.
- The real number fields are described using $\lceil \log_2[(max\_value - min\_value) \times 10^{exp} + 1] \rceil$ bits, where max_value and min_value are the same as above, and exp is the maximum number of digits after the decimal point.
- The string fields are described directly using $(len(s) + 1) \times 8$ bits, where $len(s)$ is the length of the field value. The $+1$ term is to include the end-of-string `'\0'` character, and each character needs 8 bits to describe.

Using the description schemes above, the total description length can be computed as $D(dataset) = len(ST) \times 8 + 32 + m + \sum_{i=1}^{m} D(block_i)$. The first $len(ST) \times 8$ bits describe the the structure template, and the next $32 + m$ bits describe the total number of blocks in the dataset and whether each block is a noise block or a record. $D(block_i)$ is the description length of ith block: for noise blocks, it is simply the block length times 8; for record blocks, we compute its description length accordingly.

9.3 Formal Evaluation Standard

In order to formalize our evaluation standard, we consider both the relational dataset extracted from Datamaran[11] (the procedure of converting extracted results into relational format is described in Section 3.3) and a relational dataset containing only the intended extraction targets. *We say the extraction is successful if it is possible to convert the extracted relational dataset into the target relational dataset via a sequence of the following relational operations:*

- **Concat**(R, C_1, C_2)**:** Create a new column in R. For each tuple t in R, the new entry value is equal to the concatenation of the corresponding entries in column C_1 and C_2.
- **GroupConcat**(R_1, R_2, FK, C)**:** Create a new column in R_1. For each tuple t in R_1, the new entry value is equal to the concatenation of entries in column C of tuples in R_2 with foreign-key column FK referencing t (i.e., C and FK are columns of R_2, and FK is a foreign-key column referencing R_1).
- **Trim**(R, C, pre, suf)**:** Remove the first pre characters and the last suf characters of all entries in column C of relation R (i.e., pre and suf are constant numbers).
- **Append**(R, C, pre_str, suf_str)**:** Add pre_str to the beginning and suf_str to the end of all entries in column C of relation R (i.e., pre_str and suf_str are constant strings).
- **DeleteCol**(R, C)**:** Delete column C of relation R.
- **DeleteTable**(R)**:** Delete relation R.

In other words, we consider the extraction successful if the target relational dataset can be reconstructed by merging/removing some columns in the extracted relational dataset. Intuitively, this is only possible if (a) the fields are well-aligned within each column of the extracted relational dataset (i.e., they are of the same data type); and (b) the extracted relational dataset has more fine-grained splitting of fields compared to the intended extraction format. Note that we do not allow splitting columns here, otherwise even the trivial extraction result specifying the whole record as a single field would be considered successful.

9.4 Causes for Inaccurate Extraction

Here we describe the causes for inaccurate extraction for GitHub log datasets (Section 5.3). There are 4 log files where even the exhaustive search version of Datamaran failed to find a valid structure. In the following, we list the two causes for these inaccurate extractions, and discuss the potential ways to address them.

Fail to recognize "long" records: The maximum range of records is set to be 10 lines during the experiments. In some datasets, there are some extremely "long" records that exceeds this limit. If we increase the range limit, the efficiency of Datamaran would suffer. As the records in practice can be arbitrarily long, we are still unaware of methods that can completely solve this problem.

The greedy approach for interleaved datasets: In Datamaran, we handle interleaved datasets by repeatedly applying the algorithm on the dataset. However, this greedy procedure does not always find the correct structure for interleaved dataset. Instead, sometimes we would find structure templates with characteristics of multiple types of records. The following example illustrates this phenomenon. Suppose we have two types of records with templates:

```
F: F F F\n          F: F F F F F F\n
```

[11] For RecordBreaker, it is also possible to convert the extracted result into relational format, and therefore the evaluation standard also applies.

Datamaran could potentially settle on the wrong structure template `F: (F )*F\n`, when this generic structure template has a lower regularity score compared to the two correct record templates.

9.5 Sources and Characteristics of Manually Collected Datasets

Table 5 lists the sources and characteristics of the last 10 datasets that we use for our evaluation, in addition to the 15 datasets from Fisher et al.'s paper [20]—the maximum file size among the Fisher datasets is 0.3MB, with most of them being ~0.02MB, with typically only 1 record type (for all datasets except one), and typically 1 for record span. Details can be found in the technical report.

Data source	File size(MB)	# of rec. types	Max rec. span
Thailand district info	0.19	1	8
stackexchange xml data	20	1	1
vcf genetic format	167.4	1	1
fastq genetic format	29.9	1	4
blog xml data	0.06	1	10
log file (1)	0.03	2	9
log file (2)	0.01	1	3
log file (3)	0.19	2	1
log file (4)	0.07	2	10
log file (5)	0.09	1	4

Table 5: Sources and characteristics of manually collected datasets.

9.6 Proof of Theorem 4.1

Proof. First of all, condition (b) ensures that ST_0 can be found during the generation step. Then, using condition (a), we can ensure that ST_0 to be the top structure template during the pruning step. Finally, condition (c) ensures that ST_0 will be chosen during the evaluation step. Combining all arguments, we can see that Datamaran is guaranteed to return ST_0 as the optimal structure template. □

9.7 Drill Down on User Evaluation

In our user study, we evaluated three different types of datasets: a single-line record dataset (dataset 1), multi-line record dataset with a regular pattern (dataset 2-3), and multi-line record dataset with noise (dataset 4-5). In the following, we only drill down on dataset 5, as a representative of the most complicated case, i.e., a noisy multi-line record log file. Additional drill down analyses can be found in our technical report [1].

Multi-Line Dataset with Noise. Dataset 5 is a real log dataset crawled from GitHub, with each record spanning multiple lines. Noise and incomplete records exist in this dataset. Figure 17 depicts the original file (R), the target file (T), the extraction result using Datamaran (A), and the extraction result using RecordBreaker (B) for dataset 5. As readers may have already noticed, the raw file has no regular patterns. More specifically, Line 3-5 in Figure 17(a) is a block unit forming one record, while Line 8, 10 and 13 are noise data or incomplete records. As a consequence, it is impossible to reconstruct the records via *Offset* in Excel, since there is no regular pattern in the raw file. Similarly, RecordBreaker also fails to handle such noisy datasets, because it cannot filter incomplete records or noise from the desired target. For instance, Line 2 in Figure 17(d) corresponds to the noise data, i.e., Line 8 in Figure 17(a). On the contrary, Datamaran works well with multi-line noisy datasets, and

	A	B	C	D	E	F	G	H	I	J
1										
2										
3	Processing TransactionLogController#index (for 127.0.0.1 at 2011-07-11 12:20:43) [GET]									
4	Rendering transaction_log/index									
5	Completed in 31ms (View: 29, DB: 0) | 200 OK [http://localhost/]									
6										
7										
8	Processing ApplicationController#index (for 127.0.0.1 at 2011-07-11 12:20:43) [GET]									
9										
10	ActionController::RoutingError (No route matches "/javascripts/logtail.js" with {:method=>:get}):									
11										
12										
13	Rendering rescues/layout (not_found)									
14										
15										
16	Processing TransactionLogController#index (for 127.0.0.1 at 2011-07-11 12:23:54) [GET]									
17	Rendering transaction_log/index									
18	Completed in 8ms (View: 6, DB: 0) | 200 OK [http://localhost/]									

(a) Raw File (R)

	A	B	C	D	E	F	G	H	I	J	K
1	127.0.0.1	7/11/2011	12:20:43	GET	transaction_log/index	31ms	29	0	200	OK	http://localhost/
2	127.0.0.1	7/11/2011	12:23:54	GET	transaction_log/index	8ms	6	0	200	OK	http://localhost/
3	127.0.0.1	7/11/2011	12:24:17	GET	transaction_log/index	7ms	5	0	200	OK	http://localhost/
4	127.0.0.1	7/11/2011	12:24:19	GET	transaction_log/index	2ms	1	0	200	OK	http://localhost/
5	127.0.0.1	7/11/2011	12:24:55	GET	transaction_log/index	3ms	1	0	200	OK	http://localhost/
6	127.0.0.1	7/11/2011	12:33:58	GET	transaction_log/index	3ms	1	0	200	OK	http://localhost/
7	127.0.0.1	7/11/2011	12:34:00	GET	transaction_log/index	3ms	1	0	200	OK	http://localhost/
8	127.0.0.1	7/11/2011	12:35:14	GET	transaction_log/index	3ms	1	0	200	OK	http://localhost/
9	127.0.0.1	7/11/2011	12:35:21	GET	transaction_log/index	3ms	1	0	200	OK	http://localhost/
10	127.0.0.1	7/11/2011	12:36:26	GET	transaction_log/index	6ms	2	0	200	OK	http://localhost/
11	127.0.0.1	7/11/2011	12:37:17	GET	transaction_log/index	2ms	1	0	200	OK	http://localhost/
12	127.0.0.1	7/11/2011	12:37:30	GET	transaction_log/index	3ms	1	0	200	OK	http://localhost/

(b) Target File (T)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD
1	Processing	TransactionLogController#index	for	127	0	0	1	at	2011	7	11	12	20	43	GET	Rendering	transaction_log	index	Complete	in	31ms	View	29,	DB	0	|	200	OK	http	localhost
2	Processing	TransactionLogController#index	for	127	0	0	1	at	2011	7	11	12	23	54	GET	Rendering	transaction_log	index	Complete	in	8ms	View	6,	DB	0	|	200	OK	http	localhost
3	Processing	TransactionLogController#index	for	127	0	0	1	at	2011	7	11	12	24	17	GET	Rendering	transaction_log	index	Complete	in	7ms	View	5,	DB	0	|	200	OK	http	localhost
4	Processing	TransactionLogController#index	for	127	0	0	1	at	2011	7	11	12	24	19	GET	Rendering	transaction_log	index	Complete	in	2ms	View	1,	DB	0	|	200	OK	http	localhost
5	Processing	TransactionLogController#index	for	127	0	0	1	at	2011	7	11	12	24	55	GET	Rendering	transaction_log	index	Complete	in	3ms	View	1,	DB	0	|	200	OK	http	localhost
6	Processing	TransactionLogController#index	for	127	0	0	1	at	2011	7	11	12	33	58	GET	Rendering	transaction_log	index	Complete	in	3ms	View	1,	DB	0	|	200	OK	http	localhost
7	Processing	TransactionLogController#index	for	127	0	0	1	at	2011	7	11	12	34	0	GET	Rendering	transaction_log	index	Complete	in	3ms	View	1,	DB	0	|	200	OK	http	localhost
8	Processing	TransactionLogController#index	for	127	0	0	1	at	2011	7	11	12	35	14	GET	Rendering	transaction_log	index	Complete	in	3ms	View	1,	DB	0	|	200	OK	http	localhost
9	Processing	TransactionLogController#index	for	127	0	0	1	at	2011	7	11	12	35	21	GET	Rendering	transaction_log	index	Complete	in	3ms	View	1,	DB	0	|	200	OK	http	localhost
10	Processing	TransactionLogController#index	for	127	0	0	1	at	2011	7	11	12	36	26	GET	Rendering	transaction_log	index	Complete	in	6ms	View	2,	DB	0	|	200	OK	http	localhost
11	Processing	TransactionLogController#index	for	127	0	0	1	at	2011	7	11	12	37	17	GET	Rendering	transaction_log	index	Complete	in	2ms	View	1,	DB	0	|	200	OK	http	localhost
12	Processing	TransactionLogController#index	for	127	0	0	1	at	2011	7	11	12	37	30	GET	Rendering	transaction_log	index	Complete	in	3ms	View	1,	DB	0	|	200	OK	http	localhost

(c) Extraction Result by DATAMARAN (A)

	A	B	C	D
1	Processing	TransactionLogController # index	(for 127.0.0.1 at 2011-07-11 12:20:43)	[GET]
2	Processing	ApplicationController # index	(for 127.0.0.1 at 2011-07-11 12:20:43)	[GET]
3	Processing	TransactionLogController # index	(for 127.0.0.1 at 2011-07-11 12:23:54)	[GET]
4	Processing	ApplicationController # index	(for 127.0.0.1 at 2011-07-11 12:23:54)	[GET]
5	Processing	TransactionLogController # index	(for 127.0.0.1 at 2011-07-11 12:24:17)	[GET]
6	Processing	TransactionLogController # index	(for 127.0.0.1 at 2011-07-11 12:24:19)	[GET]
7	Processing	TransactionLogController # index	(for 127.0.0.1 at 2011-07-11 12:24:55)	[GET]
8	Processing	TransactionLogController # index	(for 127.0.0.1 at 2011-07-11 12:33:58)	[GET]
9	Processing	TransactionLogController # index	(for 127.0.0.1 at 2011-07-11 12:34:00)	[GET]
10	Processing	TransactionLogController # index	(for 127.0.0.1 at 2011-07-11 12:35:14)	[GET]
11	Processing	TransactionLogController # index	(for 127.0.0.1 at 2011-07-11 12:35:21)	[GET]
12	Processing	TransactionLogController # index	(for 127.0.0.1 at 2011-07-11 12:36:26)	[GET]

(d) $B.1$

	A	B	C
1	Completed	,in,31,ms,(View: 29, DB: 0),200,OK,	[http://localhost/]
2	Completed	,in,8,ms,(View: 6, DB: 0),200,OK,	[http://localhost/]
3	Completed	,in,7,ms,(View: 5, DB: 0),200,OK,	[http://localhost/]
4	Completed	,in,2,ms,(View: 1, DB: 0),200,OK,	[http://localhost/]
5	Completed	,in,3,ms,(View: 1, DB: 0),200,OK,	[http://localhost/]
6	Completed	,in,3,ms,(View: 1, DB: 0),200,OK,	[http://localhost/]
7	Completed	,in,3,ms,(View: 1, DB: 0),200,OK,	[http://localhost/]
8	Completed	,in,3,ms,(View: 1, DB: 0),200,OK,	[http://localhost/]
9	Completed	,in,3,ms,(View: 1, DB: 0),200,OK,	[http://localhost/]
10	Completed	,in,6,ms,(View: 2, DB: 0),200,OK,	[http://localhost/]
11	Completed	,in,2,ms,(View: 1, DB: 0),200,OK,	[http://localhost/]
12	Completed	,in,3,ms,(View: 1, DB: 0),200,OK,	[http://localhost/]

(e) $B.2$

	A	B
1	Rendering	transaction_log/index
2	Rendering	transaction_log/index
3	Rendering	transaction_log/index
4	Rendering	transaction_log/index
5	Rendering	transaction_log/index
6	Rendering	transaction_log/index
7	Rendering	transaction_log/index
8	Rendering	transaction_log/index
9	Rendering	transaction_log/index
10	Rendering	transaction_log/index
11	Rendering	transaction_log/index
12	Rendering	transaction_log/index

(f) $B.3$

Figure 17: Multi-Line Dataset with Noise (Dataset 5)

can successfully extract fine-grained attributes from the raw dataset. In the following, we will illustrate how participants transformed R, A and B into T as depicted in Figure 16, respectively.

- *From A to T.* Participants simply used Concatenate or FlashFill to merge columns in Figure 17(c) into column A-C, E and K in Figure 17(b). For instance, by combining column D-G in Figure 17(c), we can obtain column A in Figure 17(b). The total number of operations is 5.
- *From B to T.* Participants first tried FlashFill and Split to extract the target information, but then they found that the partial contents in Figure 17(d), e.g., line 2, belong to the incomplete records, and it is hard to tell the noisy data from the target ones. Thus, participants failed to transform B into T.
- *From R to T.* After looking at the raw file, participants all found it impossible to convert R to T via Excel. This is due to the existence of noise and incomplete records.

9.8 Other Related Work

We now briefly mention other related work that we didn't cover in the main body of the paper.

Example-Driven HTML Wrapper Induction. There has been a long line of work on inducing or learning a "wrapper" to extract content from HTML pages, e.g., [21, 31, 32, 35, 41–43]. The majority of these papers crucially rely on both the web-page structure in the form of the DOM, as well as on text (e.g., extract the piece of text immediately following "Price:"). Examples are provided in the form of entities that belong to the concept class that are to be extracted, or in the form of explicit annotations (e.g., this location contains an item of interest to be extracted). Often, the eventual relational schema is known in advance. Some papers do not rely on the HTML structure, opting instead to use NLP [6, 19]. In our case, we do not require any seed entities or annotations.

Extracting Structure From Other Media. There is work [7, 14, 17, 53] on extracting structure from other types of media (i.e., other than text-formatted log datasets). The extraction strategies adopted by these papers crucially rely on characteristics of the target dataset type. For instance, in security research [7, 11, 17, 59–61], the network traces consist of continuous communication between server and client, best modeled as a deterministic state machine (i.e., messages between server and client represent transitions in the global state), and reliant on indicators that signal the start of a new message, e.g., the presence of an IP address; in either case, the record boundaries are clear. On the other hand, in the field of natural language processing [14, 53], the structure is usually restricted to local context (i.e., within each sentence), and can be captured using probabilistic language models. In particular, Cohen et al. [14] employs language models from other languages to learn the structure of a new language, while Spitkovsky et al. [53] uses clustering based on local context (neighboring words to a given word) to infer dependency structures to inform a sentence parser, where parsing is delimited based on periods. In our case, the fundamental characteristics of log datasets are captured in Definition 2.4, and our whole extraction strategy revolves around this definition.