Navigating the Data Lake: Unsupervised Structure Extraction for Text-formatted Data

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

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

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

ABSTRACT

Many organizations routinely accumulate automatically-generated semi-structured log file datasets; these datasets remain unused and occupy wasted space—this phenomenon has been termed as the *"data lake"* problem. One approach to put these datasets to use is to convert them into a structured relational format, following which they can be analyzed in conjunction with other datasets. To address this, we present CATAMARAN, an automatic structure extraction tool that requires no human supervision. CATAMARAN 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 files, at scale with high accuracy. Compared to unsupervised adaptations of supervised structure extraction tools developed in prior work, CATAMARAN makes fewer assumptions and achieves much higher accuracy. In particular, CATAMARAN can successfully extract structured information from all datasets used in prior work in supervised structure extraction, and can achieve 95% extraction accuracy on automatically collected log files from GitHub.

1. INTRODUCTION

Enterprises routinely collect semi-structured or partially structured datasets in shared file systems such as HDFS. These datasets are typically generated automatically as log files output by programs, and often number in the billions, e.g., Google has 26B datasets in their shared file system [11]. This phenomenon of accumulation of datasets within enterprises has recently been referred to as the *"data lake" problem* [10, 19, 20]. Unfortunately, the datasets in a data lake often remain *unused*, *unstructured*, and *uninterpreted*, and as they accumulate, they become *unmanageable*—recent work has characterized the data lake problem as one of the most important challenges facing large enterprises today [15, 19].

One approach to make these datasets more usable is to convert them into a structured (relational) format. Once we have structured the datasets, we can then infer relationships and correlations across datasets, and put them eventually to use to aid data analysis, search, or browsing [5, 6, 8, 14, 16, 21, 22, 23]. There has been a line of recent papers that explores how to simplify this structure extraction process [7, 12, 13]. These papers construct a parser for a dataset with the help of user-provided examples, or other forms of user guidance. However, this approach does not scale—for example, when dealing with billions of automatically generated datasets, it is infeasible to require human input for extraction for every single dataset. Thus, the question we explore in this paper is: ***can we develop a completely unsupervised structured data extraction tool that can extract from datasets at scale?***

Developing such a tool can have many benefits. First, the tool could help leverage the many unused datasets within enterprises, enabling them to generate structured dataset collections and put them to use. Second, this tool could be used as a "first attempt" for human-driven structure extraction of a dataset. For example, a human expert could simply confirm whether a dataset has been extracted correctly or not—a simple Yes/No question—as opposed to performing the interactions that lead to structure extraction—either providing several examples [3, 17], or selecting steps from a list of suggested structure extraction steps one after the other [12]. Or even better, the human expert could focus their attention on extracting structure from files that the tool failed to extract, i.e., the "hard" cases. Either way, this could considerably reduce the human expert time required for supervision.

Unfortunately, automatically identifying the structure of datasets is not an easy task; there are many challenges:

- *Record Endpoints.* The boundaries between records are unknown and hard to identify: although in most datasets, records are separated by the end-of-line character '\n', this character could also appear within records (i.e., multi-line records).
- *Field Endpoints.* It is not easy to distinguish between the formatting characters and the field values in records: while special characters such as the space character ' ' are often used to separate field values, they could also be part of the field values, especially in text fields.
- *Complex Structure.* There are often complex structures within records: for example, if a record contains a list of items, the number of items can vary from record to record, which makes even the underlying formatting vary between records. Furthermore, substructures could also exist within the structures via nesting.
- *Noise.* Many datasets contain noise, i.e., the portions of the datasets that obey no structure at all. These could form part of the preamble within the dataset before the records are listed, or could be in between the records as well—overall, this noise could constitute a significant fraction of the dataset.
- *Multiple Structures.* There could be multiple types of records interleaved with each other in a single dataset, which makes it even more difficult to tackle the aforementioned challenges.

These challenges make it hard for us to find a simple "golden rule" for structure extraction, and more sophisticated methods must be developed for this problem.

Related Work. Prior work on structure extraction can be broadly divided into two categories: unsupervised and semi-supervised. Semi-supervised structure extraction algorithms require the user to provide some guidance, typically via an interactive interface. The most notable semi-supervised structure extraction program is FlashExtract [13], which allows the user to gradually create and

revise the existing structure by providing examples within the interactive interface. Wrangler [12] is another well-known tool for interactive structure extraction: it allows the user to select the steps to gradually transform a dataset into structured form.

To the best of our knowledge, the only general-purpose unsupervised structure extraction algorithm is RecordBreaker [1]. RecordBreaker originates from Fisher et al.'s semi-supervised structure extraction program [7], which requires the user to provide rules for chunking and tokenizing (i.e., provide the set of characters separating field values). RecordBreaker adopts Fisher's algorithm and uses a simple default rule: they assume each record consists of exactly one line, and compile a common set of characters that are usually used in record templates and use that for tokenization. Thus, RecordBreaker is a simple unsupervised adaptation of a semi supervised structure extraction algorithm. While their approach works for simple structured datasets, it is not sufficient for more complicated structures, which is especially common in computer-generated log files (see Section 5.1 for details).

Our tool. We present CATAMARAN[1], an automatic structure extraction program that requires no human supervision. CATAMARAN 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 the minimum description length principle [4], and the best pattern is used to actually extract structured information from the dataset. (Note, however, that CATAMARAN is general, and can adapt to any scoring modality, not just minimum description length.) CATAMARAN requires careful design and engineering since a naive implementation can lead to a huge blowup in the number of structures considered and therefore the time taken for extraction—many of these design choices are described and evaluated carefully in this paper.

Thus, CATAMARAN can automatically extract structured datasets from text-formatted files without any human supervision. Compared to unsupervised adaptations of semi-supervised structure extraction systems [1, 7], CATAMARAN makes fewer assumptions regarding the structure of the dataset: *record endpoints* and *field endpoints* are no longer assumed to be known beforehand. However, even with fewer structural assumptions, CATAMARAN can still achieve the same level or even higher extraction accuracy than prior tools [1, 7]. Our theoretical analysis shows that the structure found by CATAMARAN is provably correct if the dataset satisfies certain conditions. We also experimentally evaluate the performance of CATAMARAN and demonstrate that CATAMARAN can successfully extract structure from all datasets used in Fisher et al.'s work [7], and can achieve 95% extraction accuracy on automatically collected log files from GitHub. CATAMARAN 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, CATAMARAN can still complete extraction within a few minutes. The main time spent by CATAMARAN for large datasets is in extraction (which is eminently parallelizable), and identifying an appropriate structure can be done much faster.

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 several important assumptions that will help us solve the problem in a tractable manner.
- In Section 4, we present CATAMARAN, our structure extraction algorithm.
- In Section 5, we discuss CATAMARAN in more detail:
  - We compare the high-level intuition of CATAMARAN and prior unsupervised structure extraction systems, and discuss why CATAMARAN's approach is superior.
  - We analyze the time complexity of CATAMARAN and provide theoretical correctness guarantee under ideal circumstances.
- In Section 6, we experimentally study the performance of our algorithm on 25 typical datasets and automatically collected log files from GitHub.

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

2. PROBLEM DEFINITION

In this section, we formally define the problem of structure extraction. We begin by defining the concepts of *record templates* and *instantiated records*—these concepts mirror those described in prior work on semi-supervised structure extraction. We will describe an example that clarifies these concepts subsequently.

DEFINITION 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. 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 formed 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 (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.*

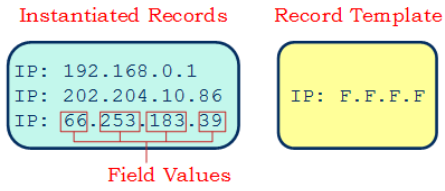

Figure 1: Record Template Illustration

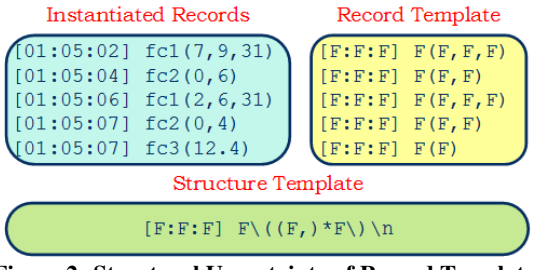

Figure 2: Structural Uncertainty of Record Templates

These definitions are illustrated via an example in Figure 1. 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., $192, 168, 0, 1, \ldots$) are the field values to be extracted from the dataset.

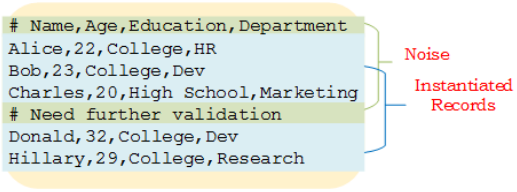

Figure 3: Text-formatted dataset illustration

In many datasets, there is some ambiguity in the structure of the record templates within. Figure 2 illustrates this phenomenon. As we can see, for different records, the corresponding record templates on the right hand side are similar but not exactly the same. To characterize this scenario, we define the concept of a *structure template*:

DEFINITION 3 (STRUCTURE TEMPLATE). *A* structure template *is a regular expression [2, 18] 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.*

Regular expressions [2,18] are a classical notion in theoretical computer science. In brief, a regular expression is a sequence of characters that define a search pattern, often used by string searching algorithms for "find-and-replace" operations on text files; a formal reference can be found in Goyvaerts and Levithan's book [9]. The bottom half of Figure 2 shows an example structure template, which corresponds to the records in the top half of the figure. The assumption that structure templates are regular expressions will be validated in experiments, together with some stronger restrictions on structure templates (which will be discussed in Section 3).

Now, we define the concept of a *text-formatted dataset*:

DEFINITION 4 (TEXT-FORMATTED DATASET). *A text-formatted 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 file with the following structure: T can be partitioned into several blocks separated by 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 3 illustrates an example text-formatted dataset. The light blue parts of the dataset are record blocks, and the green parts are noise blocks. These noise blocks have no structure within, and therefore are not relevant to the structure extraction problem.

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

PROBLEM 1 (STRUCTURE EXTRACTION). *For a text-formatted 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 structure components S such that the pair (T, S) obeys Definition 4. Most of these structures are unacceptable from an end-user's point of view. In practice, the structure extraction program 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 CATAMARAN, and the precise 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 text-formatted dataset $\mathcal{D} = \{T, S\}$ with only T observed but S unknown, find S that optimizes a given 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 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 [7, 13]. 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.
- To reduce the complexity of search space of the structural component, making the structure extraction problem more tractable.

In CATAMARAN, we make three assumptions regarding the structure of the dataset. These three assumptions will be introduced in the rest of this section. The validity of these assumptions will be experimentally verified in Section 6.2 through examination of a large collection of log files in GitHub. At each step, we will also compare the assumptions made with the ones made by RecordBreaker [1], the only known unsupervised adaptation of semi-supervised structure extraction algorithms.

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 $\alpha\%$. 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. For most datasets, there typically aren't that many different structure templates within, and thereby each structure template should cover a significant portion of the dataset. Assumption 1 makes this intuition explicit. The coverage threshold assumption allows us to prune out most unreasonable structure template candidates. We describe how we set α later on, and validate this in our experiments.

3.2 Non-Overlapping Assumption

The second assumption we make is the following:

ASSUMPTION 2 (NON-OVERLAPPING). *For any record template RT and any character c, if $c \in RT$, then no field values of any instantiated record of RT can contain c. In other words, for any record template, its character set does not overlap with the character set of the field values of its instantiated records.*

Explanation. To explain this, we define some notation first:

- *RT-CharSet* denotes the set of characters that are used in record templates.
- *F-CharSet* denotes the set of characters that are used in the field values.

Under this notation, Assumption 2 can be simply stated as: *RT-CharSet* $\cap$ *F-CharSet* $= \emptyset$. 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* $\subseteq$ *RT-CharSet-Candidate*.

Assumption 2 plays an important role in CATAMARAN: 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 all the matches of any structure template from the dataset. Assumption 2 has also been made implicitly in RecordBreaker [1].

Remark. Although the validity of Assumption 2 is not obvious, for many datasets that seemingly violate this assumption, we can still get reasonable results even if we assume it to be true. To see this, consider the record template `F,F,F,"F",F,F,F`: it violates the assumption since the field value surrounded by the quotes could potentially contain the comma character. However, if we assume Assumption 2 to be true in this example, CATAMARAN will recognize the following record templates, depending on the number of commas in the middle field value:

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

All of these record templates can be generated from the same structure template `F,F,F,"(F,)*F",F,F,F`. Therefore, all these records will be recognized as the same type if CATAMARAN successfully finds the correct structure template.

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 such regular expression either satisfies Assumption 3 or is a simple string.*

Explanation. The Array-type regular expression is intended to characterize lists of objects. For example, the record template `(F,F,F,...,F)` can be represented by a prefix "`(`" and an array-type regular expression `(F,)*F\)` corresponding to "`F,F,F,...,F)`". Assumption 3 essentially states that each structure template must follow a special tree-style structure, as illustrated in Figure 4 using the example structure template "`F,F,F,"(F,)*",F,F,F\n`". In this tree, each Struct regular expression node (root in Figure 4) has their subcomponent as children nodes (level 2 in Figure 4), and each Array regular expression node has two children (level 3 in Figure 4): the left child is the regular expression part, and right child is the terminating character part. Combining with the definition of text-formatted datasets (Definition 4), we can extract the structure from a dataset *if it can be represented as a union of tree-style structure templates* in Assumption 3. In our experimental study, we found this structural form to be powerful enough to capture almost all structures appearing in practice.

Note that the tree-structure in Assumption 3 naturally divide the field values of the dataset into different groups: the field values are in the same group if they correspond to the same field-placeholder character 'F' in the leaf nodes of the tree. For example, the field values in the tree-structure in Figure 4 are partitioned into 7 groups: 6 of them are associated with leaf nodes in level 2, and all the field values inside the array are in the same group. This partitioning allows us to separate field values with different semantic meaning

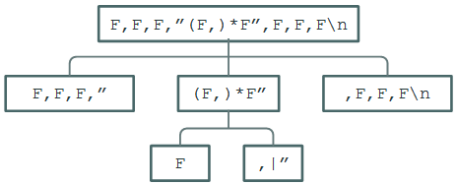

Figure 4: Structural Form Assumption

(e.g., age, height, year, time, etc.), which could be beneficial for downstream data processing and visualization.

Assumption 3 is essential to the efficiency of CATAMARAN. As we will see in Section 4.3, this assumption allows us to efficiently extract all occurrences of the structure template within a single scan of the dataset.

Remark. Our definition is slightly different from the one Fisher et al.'s work [7] and therefore in RecordBreaker [1]: in our version, the Array structure actually produces something like $SxSxSx \dots Sy$ instead of simple repetitions like $SSS \dots S$. Our definition allows us to capture all field values inside a list using one single structure: note that for the record template `(F,F,...,F)`, it is not possible to include all field values within the range of array using the simple repetition. If we use simple repetition array here (with $S =$ "`F,`"), then the last field in this list would not match the array pattern since the character after the field value is "`)`" instead of "`,`".

3.4 Assumption Comparison

We remark that among the three assumptions described above, two were also made implicitly in RecordBreaker [1]. There are two additional assumptions made in RecordBreaker, that we do not adopt, listed below:

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,* RT-CharSet *is fixed for all datasets and predetermined in advance.*

Assumption	RecordBreaker	CATAMARAN
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 CATAMARAN. Note that the two additional assumptions in RecordBreaker are very restrictive: about 31% of the log files we automatically collected from GitHub (details in Section 6.2) do not satisfy these assumptions. We will continue the discussion of these two assumptions when we compare the high-level intuition behind CATAMARAN and RecordBreaker in Section 5.1.

4. THE CATAMARAN ALGORITHM

As described in Section 2, our goal is to first find the structural component S based on the observed textual component T that optimizes a given score function $F(T, S)$, and then use S to extract structured information from the dataset. CATAMARAN uses a three-step approach to solve this problem, listed below:

- *Pruning.* In this step, we find structure templates that satisfy the coverage threshold assumption (Assumption 1), and order these structure templates using a score that is an approximation of F, which we term the *approximation score*.

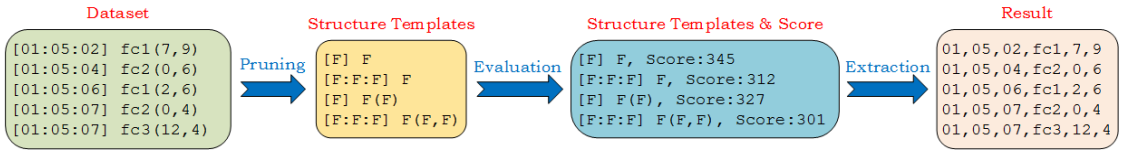

Figure 5: CATAMARAN **Workflow**

- *Evaluation.* In this step, we evaluate the score function for the top structure templates (i.e., with the highest approximation scores) from the pruning step. We also try to modify the structure templates using some structure refinement techniques, and retain the changes if the modified version has a better score $F(T, S)$ than the original one.
- *Extraction.* In this step, we use the best structure template from the evaluation step to extract information from the dataset.

The three steps are illustrated in Figure 5 for a simple example. The details of these steps will be discussed in the rest of this section: in Section 4.1, we describe the algorithm for efficiently finding structure templates that satisfy the coverage threshold assumption, and discuss the intuition behind the approximation score choice and present a theoretical justification for this score; in Section 4.2, we describe the details of the scoring function and techniques for structure refinement; in Section 4.3, we describe the algorithm for efficiently extracting all of the instantiated records from the dataset given the structure templates; in Section 4.4, we discuss some additional details of CATAMARAN that were not covered in the previous sections.

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

However, for completeness, we will present the details of the scoring function that we use in our implementation in Section 4.2.1. That said, through the rest of this paper, we assume the scoring function is given and it accurately reflects human judgment regarding the quality of structure templates.

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

Symbol	Description
M	The number of structure templates evaluated in the evaluation step
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
T_{data}	The total size of the dataset
S_{data}	The amount of data sampled during the pruning and evaluation step
c	The number of special characters (i.e., characters in *RT-CharSet-Candidate*) appearing in the dataset

Table 2: Notation Summary

4.1 The Pruning Step

In the pruning step, we first find structure templates satisfying Assumption 1 (i.e., those with at least $\alpha\%$ coverage). The algorithm for efficiently finding such structure templates is described in Section 4.1.1.

However, for most datasets, there are still thousands of structure templates even after the initial pruning based on Assumption 1. Evaluating these thousands of structure templates in the evaluation step would be extremely time-consuming. Therefore, we use an approximation score function (which can be evaluated very efficiently) to order these structure templates. Only the top M structure templates are retained and evaluated explicitly in the next step. We will discuss the intuition behind our approximation score function and theoretically analyze it in Section 4.1.2.

4.1.1 Finding Structures with Enough Coverage

CATAMARAN uses the following five steps to find structure templates with at least $\alpha\%$ coverage, described later in detail:

1. search for possible values of *RT-CharSet* (i.e., the character set in the record template), 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*.
4. reduce the record template into a structure template.
5. store all structure templates generated in step 4 within a hash-table, and then find the ones that satisfies the coverage threshold assumption.

These steps are discussed in more detail in the following:

First step: We implemented two searching procedures in CATAMARAN for finding the optimal *RT-CharSet*. Both searching procedures require $char\_candidates$, the set of characters that can potentially be in *RT-CharSet*, as an input, which contains all special characters (i.e., characters in *RT-CharSet-Candidate*) appearing in the dataset. The exhaustive search enumerates all 2^c subsets of $char\_candidates$, while the greedy search only enumerates $O(c^2)$ of them.

For the greedy search procedure, we initially set the *RT-CharSet* to be empty. In each step, we consider all of the characters in *RT-CharSet-Candidate* and attempt to add one of them to *RT-CharSet*. Among all of the possible choices, we decide on the one that can generate the best structure template (i.e., the one with best approximation score).

The following example helps illustrate the two searching procedures: consider the dataset in the leftmost block in Figure 5, with 7 special characters: `[]:(),` (space character). The exhaustive search would enumerate all 128 possible subsets, while the greedy search starts from the empty set and gradually adds new characters into it. In the first step of the greedy search, it enumerates all the one-character subsets containing only one character, and computes the structure templates using these values of *RT-CharSet* (i.e., invoking steps 2-5). It then decides which subset to proceed based on which one has the structure template with the best approximation score. For this example dataset, the structure template with the best approximation score is "`F:F:F`" in the first step (details of the approximation score will be presented in Section 4.1.2). Therefore, in the second step of the greedy search, it would enumerate all 6 subsets consisting of the character ':' and one additional character. This procedure repeats until either the subset is full or we can no longer find any structure template with at least $\alpha\%$ coverage. In this

example, the maximum number of subsets that the greedy search would have enumerated is 1+7+6+5+4+3+2+1=29 (also counting the empty subset here).

These two searching procedures represent a trade-off between accuracy and efficiency: exhaustive search is slower but gives us better extraction results. We will experimentally study their performance in Section 6. Note that if the field values do not contain any special characters in *RT-CharSet-Candidate*, then *RT-CharSet* is equal to $char\_candidates$ (i.e., *RT-CharSet* contains every special character). In this case, the greedy search procedure is guaranteed to find the correct value of *RT-CharSet* since it will always consider the full subset at the end of the searching procedure.

Second step: We only enumerate pairs of end-of-line characters `'\n'` that are close to each other: the number of lines between them is at most L. For example, when $L = 1$, we will only enumerate contents between the ith and $(i+1)$th end-of-line character. In this way, we can ensure that the number of pairs that need to be enumerated is linear in the total number of lines in the dataset. The reader may have noticed that this step may be very expensive for large datasets, we discuss how we can use sampling to ameliorate this in Section 4.4.

Third step: This step relies on the non-overlapping assumption (Assumption 2). Recall that Assumption 2 states that the character set of record templates (*RT-CharSet*) does not overlap with the character set of field values (*F-CharSet*). By this assumption, the record template can be uniquely extracted from any of its instantiated records if we know the corresponding *RT-CharSet* beforehand.

For example, if we know that the *RT-CharSet* consists only `','` and `'\n'` characters, then the following instantiated record

```
1,2,3,45,6,78,9,a,bc,d\n
```

can be transformed into the following record template by replacing every other character with the field placeholder:

```
F,F,F,F,F,F,F,F,F,F\n
```

Fourth step: We find out 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`. Note that if there are conflicting reduction steps (i.e., reduction steps that cannot be performed simultaneously), we choose one of them arbitrarily, and therefore this process only guarantees to find a minimal structure template (i.e., structure template that cannot be reduced further). This means that not all instantiated records are correctly reduced back to the corresponding structure template. As a result, the coverage estimate during the initial pruning is an underestimate. However, in our experiments, the initial coverage estimate is usually not far away from the correct value, and therefore is still well above the $\alpha\%$ threshold.

Fifth step: 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 $\alpha\%$ total coverage, the associated structure templates are pruned out.

The pseudocode of the pruning step can be found in Algorithm 1. Two searching procedures correspond to function *GreedySearch* and *ExhaustiveSearch* respectively. The function *GenST* finds structure templates with at least $\alpha\%$ coverage given the value of *RT-CharSet*.

Algorithm 1 The Pruning Step

```
    function GenST(char_set)
        n ← Total Number of Lines
        for i ← 1 to n do
            for j ← i + 1 to i + L do
5:              left_boundary ← i
                right_boundary ← j
                r ← ExtractRecord(left_boundary, right_boundary)
                rt ← ExtractRecordTemplate(r, char_set)
                st ← GenerateStructureTemplate(rt)
10:             k ← ComputeHashKey(st)
                cov(k) ← cov(k) + length(r)
                st_set(k) ← st_set(k) ∪ {r}
            end for
        end for
15:     Find all hash keys with more than α% coverage.
        return the associated structure templates.
    end function
    function ExhaustiveSearch(char_candidates)
        ST_set ← ∅
20:     for char_set ⊆ char_candidates do
            ST_set ← ST_set ∪ GenST(char_set)
        end for
        return M structure templates in ST_set with highest approximation score.
    end function
25: function GreedySearch(char_candidates)
        char_set ← ∅
        ST_set ← ∅
        repeat
            new_best_char_set ← ∅
30:         best_f ← 0
            for c ∈ char_candidates \ char_set do
                new_char_set ← char_set + c
                new_ST_set ← GenST(new_char_set)
                ST_set ← ST_set ∪ new_ST_set
35:             for st ∈ new_ST_set do
                    if ApproximationScore(st) > best_score then
                        best_score ← ApproximationScore(st)
                        new_best_char_set ← new_char_set
                    end if
40:             end for
            end for
            char_set ← new_best_char_set
        until no structure template has at least α% coverage
        return M structure templates in ST_set with highest approximation score.
45: end function
```

4.1.2 The Approximation Score

The approximation score in the pruning step is used to order the structure templates, so that only the best M ones need to be evaluated explicitly in the evaluation step. Therefore, a good approximation score should have the following two characteristics:

- Structure templates with a higher score $F(T, S)$ should also be ranked higher when using the approximation score.
- The approximation score can be efficiently computed.

We initially considered using the coverage value directly as the approximation score: structure templates with low coverage should not have a good score. However, the coverage score is not ideal here since it heavily favors simple structure templates. For example, the simplest record template "`F\n`" has 100% coverage.

To address this shortcoming, we introduce another term into the approximation 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 used to penalize extremely simple structure templates. The final approximation score in CATAMARAN is the following:

$$ApproximationScore(S) = Cov(S) \times Non\_Field\_Cov(S)$$

Theoretical Properties of the Approximation Score. As it turns out, the above approximation score has some desirable theoretical

properties, as is formalized by the following theorem:

THEOREM 1. *For a text-formatted dataset $D = \{T, S\}$ with only T observed, if all the following conditions are met:*

1. *T is noise-free.*
2. *S consists of only one single structure template ST_0 (i.e., the cardinality of S is 1).*
3. *the field values of instantiated records of ST_0 contain no special characters in RT-CharSet-Candidate.*

Then, ST_0 has the highest approximation score among all possible structure templates. Furthermore, there are at most $\frac{1}{2}L(L+1)$ structure templates with the same approximation score among all structure templates found by Algorithm 1, where L is the maximum record span (i.e., the number of lines each record can contain, see Algorithm 1).

In brief, this means that the correct structure template is guaranteed to be ranked the highest when using the approximation score during the pruning step, in an ideal situation.

PROOF. Since the dataset is noise-free and ST consists of only one structure template, the structure template ST_0 must have 100% coverage. ST_0 also has the highest non-field-coverage since the dataset is noise-free and field values contain no special characters. It immediately follows that ST_0 has the highest approximation score among all possible structure templates.

For the second statement, note that any structure template with the same approximation score as ST_0 must have 100% coverage and have the same *RT-CharSet* as ST_0. The total coverage of all structure templates with the same RT-CharSet as ST_0 is $\frac{1}{2}L(L+1)\times 100\%$. Therefore, the number of structure templates satisfying both conditions is at most $\frac{1}{2}L(L+1)$. □

4.2 The Evaluation Step

In the evaluation step, we evaluate the score for each structure template obtained from the previous pruning step, and try to refine them if possible. We first introduce the scoring function implemented in CATAMARAN in Section 4.2.1, and then propose two structure refining techniques in Section 4.2.2.

4.2.1 Scoring Function

The score implemented in CATAMARAN is based on the minimum description length principle [4]: we design a record generation procedure from the structure template, and compute the total amount of additional information needed to describe the instantiated records using the structure template. The final score is the total amount of information needed for describing all the instantiated records, plus the additional information needed to describe the noise blocks.

Describing the record using the structure template is straightforward given the structural form assumption (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 utilize the group partitioning of field values in Section 3.3, and associate each group 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 in this field.

Algorithm 2 The Evaluation Step

```
function EVALST(ST)
    (RecordBlocks, NoiseBlocks) ← ParseData(ST)
    Determine the data types of field values
    Learn the distributional parameters
    TotalDL ← len(ST) × 8 + 32 + NumBlocks
    for record ∈ RecordBlocks do
        RT ← GetRecordTemplate(record)
        TotalDL ← TotalDL + D(RT|ST)
        TotalDL ← TotalDL + D(record|RT)
    end for
    for block ∈ NoiseBlocks do
        TotalDL ← TotalDL + len(block) × 8
    end for
    return TotalDL
end function
function REFINEST(ST)
    repeat
        ST' ← UnfoldArray(ST)
        if EvalST(ST) > EvalST(ST') then
            ST ← ST'
        end if
    until ST cannot be further unfolded
    ST ← ShiftStructure(ST)
    return ST
end function
```

- 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 follows:

$$D(dataset) = len(ST)\times 8 + 32 + m + \sum_{i=1}^{m} D(block_i) \text{ bits}$$

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.

The pseudocode for computing the description score can be found in Algorithm 2, with the following 3 steps:

1. extract all the instantiated records from the dataset.
2. estimate the data-type parameters from the extracted records.
3. compute the description length using the formulae above.

4.2.2 Structure Refinement

In order to further improve the extraction accuracy of CATAMARAN, we developed two techniques to refine the structures during the evaluation step. These techniques are applied to all the top M structure templates: we revise these structure templates, and compare the revised structure templates against the original ones (using

the scoring function). We replace the original structure template with the revised ones when the score is improved.

The Array Unfolding Technique. During the pruning step, we transform record templates into structure templates by finding the minimum structure template that can generate the record template. However, sometimes the minimum structure template may not be the optimal one. For instance, in comma-separated values (CSV) files, the record template (which looks like `"F,F,F,...,F,F\n"`) should be treated as a struct instead of array: since these field values could have different value-types, separating these field values instead of grouping them into one single array both improves the score and makes it easier for downstream processing as we discussed in Section 3.3.

The array unfolding technique is designed to handle this case: for each array-type regular expression in the structure template, we attempt to unfold it by replacing it with a struct-type regular expression. For instance, we can unfold the array-type regular expression `({regex},)*{regex}\n` into struct-type regular expression `{regex},{regex},{regex},{regex}\n`. If the unfolded structure template has better score than the original one, then we can finalize the unfolding.

Partial unfolding is another type of array unfolding mechanism implemented in CATAMARAN. In partial unfolding, we attempt to replace the "first part" of an array-type regular expression with a struct-type regular expression. For instance, the array-type template `({regex},)*{regex}\n` can be unfolded into the following struct-type template:

```
{regex},{regex},({regex},)*{regex}\n
```

Partial unfolding is primarily used to handle the cases where regular field values are "mixed up" with text field values, as in the following example:

```
Apr 24 04:02:24 srv7 snort shutdown succeeded
Apr 24 04:02:24 srv7 snort startup succeeded
Apr 24 14:44:28 srv7 Disabling nightly yum
```

The first four fields are regular fields, but the last one is a free-text field. Note that the ideal structure template here is `F F F F (F )*F\n`, but the minimum structure template here is a single array `(F )*F\n`. Since the regular array unfolding technique does not apply here, we have to use the partial unfolding technique to separate these fields apart.

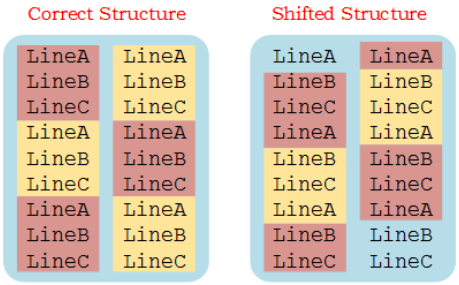

Figure 6: Structure Shifting

The Structure Shifting Technique. There is one drawback with the application of the simple minimum description length scoring function in Section 4.2.1: it cannot properly distinguish structure templates obtained by cyclic shifting.

Figure 6 illustrates this phenomenon: the left hand side depicts the correct structure template `LineA\nLineB\nLineC\n`, while the right hand side shows the shifted structure template `LineB\nLine C\nLineA\n`. Note that the correct structure template has essentially the same description length as the shifted structure template.

Ideally, the scoring function should be able to distinguish structure shifting. However, since our simple scoring function fails in this aspect, another metric is used here to compare these shifted structure templates: the position of first occurrence in the dataset. Intuitively, among all the shifted structure templates, the one with the earliest first occurrence is most likely the correct structure.

4.3 The Extraction Step

Once we found the best structure template, the actual extraction step can be completed within a single scan over the dataset. Recall that in Section 3.3 we explained that structure templates following the structural form assumption (Assumption 3) have tree-style structures (see Figure 4), where each node in the tree structure is either:

- an array node that has exactly two children.
- a struct node that has several children nodes.
- a leaf node representing a simple string.

It is straightforward to match any given string with the structure template via this tree structure:

1. Replace consecutive characters in *F-CharSet* with field placeholder character 'F'.
2. Scan the string from left and right while traversing through the tree structure (depth-first, left to right).
3. Whenever we encounter a leaf node, check if the string content matches the current input.
4. When we reach the second child of any array node, we decide whether we return to the beginning of the array, or proceed forward. This decision can be made by comparing the current character with two possible branches.

Using the above procedure, we can parse the entire dataset by initiating a matching procedure at the beginning of each line. Whenever any such matching procedure succeeds, we output the record, and the next matching procedure will be started from the end of current record. This process continues until we reach the end of file.

4.4 Other Algorithmic Details

There are two additional algorithmic details that do not fit into previous sections, and we discuss them here.

Handling Multiple Structure Templates. In case that there are more than one type of record in the dataset, we repeat the entire process described above multiple times. Each time we retrieve the parts of the dataset not explained by the previous structure. These parts are concatenated together, and we run the entire procedure on it again.

Sampling Implementation. In the actual implementation of CATAMARAN, sampling is used instead of simply scanning through the entire dataset in both the pruning and evaluation step. For large datasets, scanning the whole dataset in these steps could be extremely slow since it is repeated for every possible value of *RT-CharSet* in the pruning step and every top-M structure template in the evaluation step. Our sampling implementation is cache-aware: we sample several large chunks of data and concatenate them in the memory. The scanning in both steps will be performed on the concatenated chunks instead.

5. PERFORMANCE DISCUSSION

Now that we have described the algorithmic details of CATAMARAN, we are now in a better position to compare the capabilities of CATAMARAN with other tools. In Section 5.1, we compare the high-level intuition of CATAMARAN and RecordBreaker [1] and discuss why CATAMARAN's approach is superior. In Section 5.2,

we analyze the time complexity of CATAMARAN and provide a theoretical correctness guarantee.

5.1 Comparison with RecordBreaker

In CATAMARAN, we adopted a fundamentally different approach to RecordBreaker [1], the unsupervised adaption of prior semi-supervised structure extraction systems. In this section, we compare the high-level intuition of two systems, and discuss why CATAMARAN's approach is superior.

RecordBreaker's Approach. The core task of the structure extraction problem is to identify the correct structure template, which is challenging due to the structural uncertainty of record templates (see Figure 2 in Section 2). RecordBreaker's approach is based on summarization: the program first extracts all record templates, then tries to find the best structure template that summarizes all record templates. The left hand side of Figure 8 illustrates RecordBreaker's approach.

The Problem with RecordBreaker's Approach. Assumption 4 (i.e., the boundaries of records can be identified beforehand) is the key assumption in RecordBreaker's approach. In order to summarize the common structure from record templates, it is necessary that most of the record templates are correctly identified, which requires that we know the boundaries of records beforehand.

However, as we will see in our experimental study in Section 6, Assumption 4 doesn't hold for many log files. In fact, there are three common characteristics of log files that make it nearly impossible to determine the boundaries of records in advance:

- *Multi-line*: each record can occupy more than one line.
- *Noise*: a significant fraction of the file are noise.
- *Interleaved*: there can be multiple types of records.

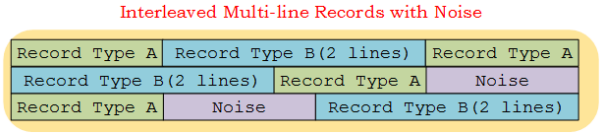

Figure 7: Log File Characteristics

Figure 7 illustrates an example (fictitious) dataset with all three characteristics, for which it is clear that Assumption 4 is false. Since RecordBreaker relies on Assumption 4, it is impossible for their system to handle such kind of datasets.

CATAMARAN's Approach. In CATAMARAN, we adopted a different approach for this task: we infer the most plausible structure template directly from each record template. We first transform all record templates into structure templates, then identify those with sufficient coverage across the dataset. The right hand side of Figure 8 illustrates CATAMARAN's approach.

Note that CATAMARAN's approach no longer requires Assumption 4: by simply enumerating every pair of nearby `'\n'` characters, we can guarantee that all the correct record boundary pairs are being accounted for. Now, this collection of "potential records" contains all the actual records, but it also contains a lot of noise. Therefore, we no longer try to find structure templates that explain all the record templates, but only find structure templates covering a sufficient portion of these "potential record templates".

5.2 Theoretical Analysis

Time Complexity Analysis. Table 3 lists the time complexity of the three steps in CATAMARAN respectively. An explanation for the symbols can be found in Table 2. Note that for large datasets, S_{data} is constant (due to sampling). In such cases, the running of our algorithm is dominated by the extraction step, which does not depend on the value of L, c and M.

Step	Time Complexity
Pruning	$O(S_{data}L2^c)$ (exhaustive) / $O(S_{data}Lc^2)$ (greedy)
Evaluation	$O(MS_{data})$
Extraction	$O(T_{data})$

Table 3: Time Complexity of the Three Steps in CATAMARAN

Correctness Guarantee. The following theorem demonstrates that CATAMARAN is guaranteed to find the correct structure template under an ideal setting:

THEOREM 2. *For a text-formatted dataset $D = \{T, S\}$ with only T observed, if the following additional conditions are all met:*

1. *T is noise-free.*
2. *S consists of only one single structure template ST_0 (i.e., the cardinality of S is 1).*
3. *the field values of instantiated records of ST_0 contain no special characters in RT-CharSet-Candidate.*
4. *For at least $\alpha\%$ of the instantiated records, the minimum structure template for them is ST_0.*
5. *$M > \frac{1}{2}L(L+1)$.*
6. *ST_0 has the best score among all structure templates.*

Then, CATAMARAN *is guaranteed to return ST_0 as the optimal structure template.*

PROOF. Based on Theorem 1, ST_0 is guaranteed to be among the top $\frac{1}{2}L(L+1)$ structure templates, provided that it can be found during the initial pruning stage. Together with the second additional condition, we can ensure that ST_0 will be among the top M provided that it is found during the initial pruning stage. The first additional condition ensures that we can find ST_0 during the initial pruning stage, and the third additional condition ensures that ST_0 will be selected during the evaluation step. Combining all arguments, we can see that CATAMARAN is guaranteed to return ST_0 as the optimal structure template. □

6. EXPERIMENTAL EVALUATION

In this section, we experimentally study the performance of CATAMARAN. The experiments are conducted on two sets of datasets with different purposes:

- **Manually collected datasets.** We collected a prototypical set of 25 datasets from various sources. These datasets cover a wide variety of structural formats and possess different characteristics (e.g., file size, structural complexity, etc.). We use these datasets to study various properties of CATAMARAN such as efficiency, parameter sensitivity, and scalability.
- **GitHub datasets.** We crawled a collection of 100 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 CATAMARAN in such settings.

The data sources of manually collected datasets and the experimental results can be found in Section 6.1. The details of GitHub datasets and corresponding experimental results can be found in Section 6.2.

Default Values of Parameters: The default values for the three parameters in CATAMARAN are: $\alpha = 10\%$ (the coverage threshold parameter); $L = 10$ (the upper bound for record span); $M = 50$ (the number of remaining structure templates after the pruning step). These default values are used in all experiments except in Section 6.1.2 and 6.1.3, where we study the sensitivity of these parameters.

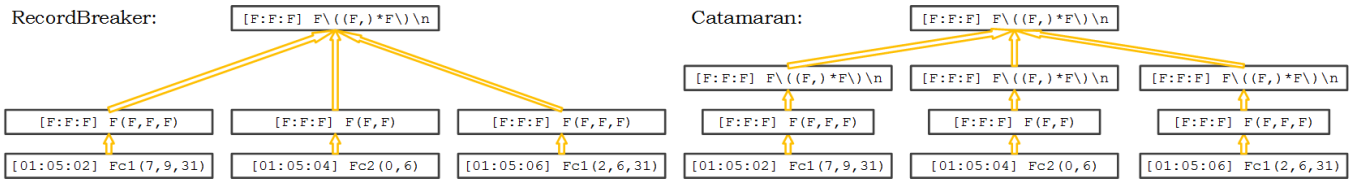

Figure 8: The different approaches of RecordBreaker and CATAMARAN

6.1 Manually Collected Datasets

In this section, we use the 25 manually collected datasets to study the properties of CATAMARAN. Table 4[2] lists the sources and characteristics of these 25 datasets. We used all 15 datasets from Fisher et al.'s paper [7] (marked with "*" in Table 4). Since Fisher's collection lacks large datasets or complex datasets (i.e., datasets containing multiple types of records or multi-line records), we also collected 10 additional datasets from the internet as well as from our genomics collaborators. The download links of these datasets can be found at https://publish.illinois.edu/structextract/.

Data source	File size(MB)	# of rec. types	Max rec. span
*transaction records	0.07	1	1
*comma-sep records	0.02	1	1
*web server log	0.29	1	1
*log file of Mac ASL	0.28	1	1
*Mac OS boot log	0.02	1	1
*crash log	0.05	1	1(3)
*crash log (modified in [7])	0.05	1	1(3)
*ls -l output	0.01	1	1
*netstat output	0.01	2	1
*printer logs	0.02	1	1
*personal income records	0.01	1	1
*US railroad info	0.01	1	1
*application log	0.06	1	1
*LoginWindow server log	0.05	1	1
*pkg install log	0.02	1	1
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 from GitHub (1)	0.03	2	9
log file from GitHub (2)	0.01	1	3
log file from GitHub (3)	0.19	2	1
log file from GitHub (4)	0.07	2	10
log file from GitHub (5)	0.09	1	4

Table 4: Dataset sources and characteristics. Datasets marked with "*" were also used in Fisher et al.'s work [7].

6.1.1 Extraction Accuracy

Evaluation Standard: Recall that the structure extraction problem is not well-posed, and the validity of the extracted structure solely depends on the end-user. However, for many datasets, there are usually multiple potentially valid structures from the end-user's point of view. 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
```

Depending on the application, users may not share the same opinion regarding the validity of extracted structures. This fact makes the evaluation of extraction result particularly difficult. Therefore, we use the following standard to objectively evaluate the extraction accuracy of CATAMARAN:

- We first manually create an "ideal" structure for each dataset.
- For each field value in the "ideal" structure, we consider it successfully extracted if either: (a) it is contained in the extracted structure; or (b) it can be reconstructed by combining several fields in the extracted structure.
- The structure extraction for each dataset is considered *successful* if **all of the field values** in the "ideal" structure are successfully extracted.

Note that we cannot use the description length score here to evaluate the extraction quality since CATAMARAN is directly optimizing this metric. If we use the description length score for utility evaluation, the result would be biased and we would overestimate the utility of CATAMARAN.

Experimental Findings: We used CATAMARAN to extract structures from the 25 datasets, and the extractions are successful for all 25 datasets based on the above standard. For single-line record datasets, CATAMARAN correctly identified the fact that each record consists of exactly one line, even though the possibility of multi-line records are also considered. For multi-line record datasets, CATAMARAN identified the boundaries of records perfectly even for noisy datasets. For datasets containing multiple types of records, CATAMARAN can also correctly identify the type of each record.

End-product of CATAMARAN: the end-product of CATAMARAN is a tab-separated values(tsv) format file. This format can be readily recognized by spreadsheet software like Microsoft Excel, making it convenient for follow-up data processing. Figure 9 shows an example end-product of CATAMARAN.

	A	B	C	D	E	F	G	H
1	207	136	97	-	-	15	Oct	1997
2			49					
3	207	136	97	-	-	15	Oct	1997
4			49					
5	207	136	97	-	-	15	Oct	1997
6			49					
7	207	136	97	-	-	15	Oct	1997
8			49					
9	208	196	124	-	-	15	Oct	1997
10			26					
11	208	196	124	-	-	15	Oct	1997
12			26					

Figure 9: End-Product of CATAMARAN

Comparison with Recordbreaker. We were unable to build RecordBreaker [1] following their installation instructions[3]. But since RecordBreaker can only handle single-line records, we know for certain that it cannot extract correctly for 7 out of 25 datasets. Therefore, their extraction accuracy would 18/25 at best—if they get all the remaining 18 correct. Also note that the 15 datasets demonstrated in Fisher et al.'s work [7] actually have very simple structure.

6.1.2 Running Time

In this section we study the efficiency of CATAMARAN. We first run CATAMARAN on the 25 datasets using the default parameters to study the connection between the characteristics of datasets (size/structural complexity) and the running time of CATAMARAN. Then, we vary the parameters to study their impact on the efficiency of CATAMARAN.

[2]for crash log datasets, there are two valid structures with max record span 1 and 3 respectively.

[3]We encountered many errors following the exact steps provided at `http://cloudera.github.io/RecordBreaker/`.

Running Time vs. Dataset Size: Figure 10 depicts the impact of the size of the dataset on the running time of CATAMARAN (using either exhaustive search or greedy search). The running time on small datasets (less than 50MB) is dominated by the pruning 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 CATAMARAN to process the largest dataset here (with size 167MB). The running time in this case is dominated by the extraction step, which scales linearly with the dataset size.

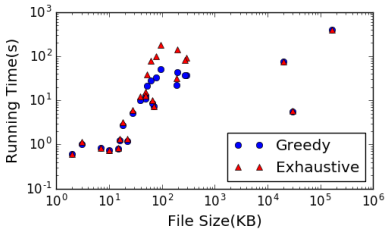

Figure 10: Running Time vs. Dataset Size

Running Time vs. Structural Complexity: Figure 11 depicts the impact of the structural complexity of the dataset on the running time of CATAMARAN. The structural complexity of datasets are characterized using the total number of structure templates with at least 10% coverage. In general, it takes a longer time for CATAMARAN to extract datasets with higher structural complexity, and the efficiency benefits of greedy search is more significant on these datasets.

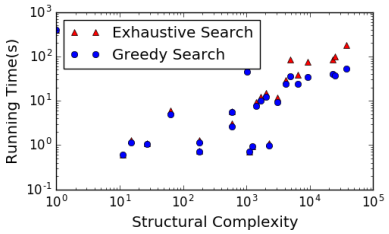

Figure 11: Running Time vs. Structural Complexity. x axis: number of structure templates with at least 10% coverage.

Running Time vs. Parameters: Figure 12 shows the impact of parameters on the running time of CATAMARAN (exhaustive search). As we can see in the top figure, the value of M directly affects the overall extraction time, and this effect is more significant for larger datasets. In the bottom figure, we can see that changing parameters α or L also affect the efficiency of CATAMARAN.

Note that if we evaluate all structure templates with at least $\alpha\%$ coverage ($M = \infty$), the average running time would be longer than 6 minutes even for small datasets. Therefore, it is necessary to use approximation score to prune out structure templates.

6.1.3 *Parameter Sensitivity*

Since there are usually multiple valid structures for these datasets, the extraction accuracy is not well suited for characterizing parameter impact. Therefore, we use another metric to evaluate the impact of parameters: whether CATAMARAN can find the optimal structure template (i.e., the structure template with best score, this is found by evaluating the score of every structure template with at least $\alpha\%$ coverage). Figure 13 shows the percentage of datasets in which CATAMARAN can find the optimal structure template on different parameter combinations. As we can see, CATAMARAN is very robust with respect to the parameter settings: even for parameter M, changing its value from 50 to 1000 only increased the likelihood of finding the optimal structure by about 10%. Figure 13

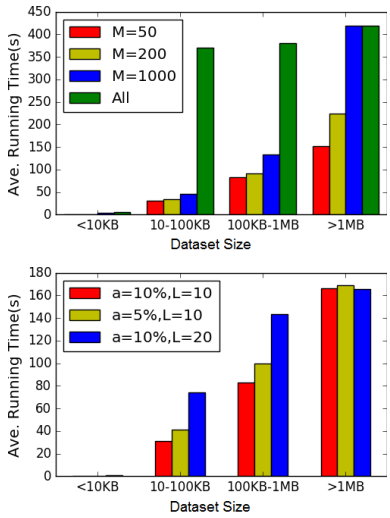

Figure 12: The impact of parameters on the running time.

also verifies the effectiveness of the approximation score in practice: for 40% of the datasets, the optimal structure also has the best approximation score.

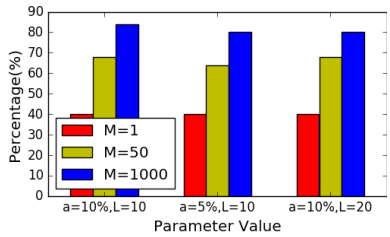

Figure 13: The percentage of datasets in which CATAMARAN can find the optimal structure on different parameter settings

Note that it is not necessary for CATAMARAN to find the optimal structure, any structure that respects the criterion in Section 6.1.1 is sufficient. The metric used in this section is solely for comparison purpose.

Combining with the results in Section 6.1.2, we suggest using the following default parameter configuration in practice: $\alpha = 10\%$, $L = 10$, $M = 1000$.

6.2 GitHub Datasets

GitHub contains a large quantity of log files generated by programmers across the world. We collected a set of such log files by searching for files end with ".log", with length greater than $20,000$ on github.com. The first 1000 search results for each of the following keywords[4] are collected: "db", "2016", "system", "query", "user". After that, 100[5] random datasets are uniformly sampled from the 5000 datasets. The characteristics of these datasets are discussed in Section 6.2.1, and the experimental results are discussed in Section 6.2.2.

6.2.1 *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*

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

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

There are five possible labels of datasets based on the above criteria, which are listed in Table 5. The distribution of labels among the 100 sampled log files is shown in Figure 14.

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 5: GitHub dataset labels

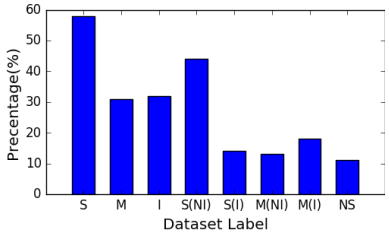

Figure 14: The Characteristics of GitHub Log Files

In the following, we discuss several findings from Figure 14:

- **Validity of Structural Assumptions:** The structure of 89% of these datasets follow assumptions in Section 3, and 10% of the datasets has no structure at all (nothing can be extracted from these files), only 1% dataset have some structure but cannot be described within the framework in Section 3. These statistics suggest that assumptions in Section 3 are well-justified.
- **Necessity for Multi-line Record Handling:** In 31% of these datasets, there are 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:** There are more than one type of records in 32% of these datasets. 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.

Limitations of RecordBreaker. Table 6 shows the applicability of RecordBreaker and CATAMARAN on the different types of datasets. Combining with Figure 14, we see at least 31%[6] of datasets cannot be handled by CATAMARAN. This fact suggests that RecordBreaker is not well-suited for extracting structure from log files.

Label	RecordBreaker	CATAMARAN
S & NI	Need Assumption 5	Yes
S & I	Need Assumption 5	Yes
M & NI	No	Yes
M & I	No	Yes

Table 6: Applicability of RecordBreaker and CATAMARAN

6.2.2 *Structure Extraction Accuracy*

We applied CATAMARAN to extract structured information from these datasets. Figure 15 shows extraction accuracy for different types of datasets (based on the standard in Section 6.1.1). Overall, CATAMARAN successfully extracted structure from 85 datasets. The accuracy is 95.5% if we exclude datasets with no structure.

As we can see in Figure 15, CATAMARAN achieved 100% accuracy on single-line non-interleaved datasets, the simplest type of dataset. The accuracy of CATAMARAN 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. Next, we describe the causes for inaccurate extraction.

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

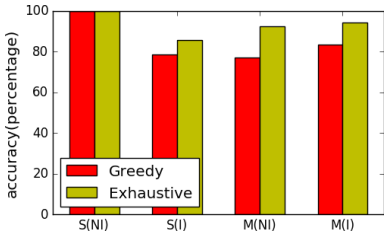

Figure 15: Extraction Accuracy on GitHub Datasets

There are 4 log files where even the exhaustive search version of CATAMARAN 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 CATAMARAN would suffer. Furthermore, since records can be arbitrarily long, we are still unaware of methods that can completely solve this problem.

The greedy approach for interleaved datasets: Our approach for handling interleaved datasets is to repeatedly apply 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 template 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
```

CATAMARAN could potentially lock on the wrong structure template `F: (F )*F\n`. This happens if this structure template has a lower description length compared to the two correct record templates. To address this problem, we need to simultaneously search for multiple structure templates. We currently don't know how to perform such search procedure efficiently, and plan to investigate this direction in future work.

7. CONCLUSIONS AND FUTURE WORK

We presented CATAMARAN, a completely unsupervised automatic structure extraction tool. We formally defined the structure extraction problem as an optimization problem, where we are given a scoring function that reflects human judgment, and we search for the best structure template that optimizes this scoring function. The algorithm consists of three major steps: the pruning step searches for structure templates satisfying the minimum coverage threshold assumption, and orders them using an approximation score function; the evaluation step evaluates the structure templates found in the pruning step, and further refines them to achieve even better score; the extraction step extracts all the field values using the given structure template. We experimentally evaluated CATAMARAN on a collection of representative datasets and a large collection of log files crawled from GitHub. The experimental results demonstrate that CATAMARAN 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. We have identified major causes of extraction failures, and plan on address the these limitations of CATAMARAN in future work.

8. REFERENCES

[1] Recordbreaker: Automatic structure for your text-formatted data. `http://cloudera.github.io/RecordBreaker/`.

[2] V. Alfred. Algorithms for finding patterns in strings. *Algorithms and Complexity*, 1:255, 2014.

[3] D. W. Barowy, S. Gulwani, T. Hart, and B. G. Zorn. Flashrelate: extracting relational data from semi-structured spreadsheets using examples. In *PLDI'15*, pages 218–228, 2015.

[4] A. Barron, J. Rissanen, and B. Yu. The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2760, 1998.

[5] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. WebTables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549, 2008.

[6] K. Chakrabarti, S. Chaudhuri, Z. Chen, K. Ganjam, Y. He, and W. Redmond. Data services leveraging bings data assets. *Data Engineering*, page 15, 2016.

[7] K. Fisher, D. Walker, K. Q. Zhu, and P. White. From dirt to shovels: fully automatic tool generation from ad hoc data. In *ACM SIGPLAN Notices*, volume 43, pages 421–434. ACM, 2008.

[8] H. Gonzalez et al. Google fusion tables: data management, integration and collaboration in the cloud. In *SoCC*, pages 175–180, 2010.

[9] J. Goyvaerts and S. Levithan. *Regular expressions cookbook*. Oreilly, 2009.

[10] R. Hai, S. Geisler, and C. Quix. Constance: An intelligent data lake system. In *SIGMOD'16*, pages 2097–2100. ACM, 2016.

[11] A. Halevy et al. Goods: Organizing google's datasets. In *SIGMOD'16*, pages 795–806. ACM, 2016.

[12] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *CHI'11*, pages 3363–3372. ACM, 2011.

[13] V. Le and S. Gulwani. Flashextract: a framework for data extraction by examples. In *ACM SIGPLAN Notices*, volume 49, pages 542–553. ACM, 2014.

[14] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and Searching Web Tables Using Entities, Types and Relationships. *PVLDB*, 3(1):1338–1347, 2010.

[15] J. Rivera and R. van der Meulen. Gartner says beware of the data lake fallacy. In *Gartner http://www.gartner.com/newsroom/id/2809117*, 2014.

[16] A. D. Sarma, L. Fang, N. Gupta, A. Y. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *SIGMOD Conference*, pages 817–828, 2012.

[17] R. Singh and S. Gulwani. Transforming spreadsheet data types using examples. In *POPL'16*, pages 343–356, 2016.

[18] M. Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

[19] B. Stein and A. Morrison. The enterprise data lake: Better integration and deeper analytics. *PwC Technology Forecast: Rethinking integration*, (1), 2014.

[20] I. Terrizzano, P. M. Schwarz, M. Roth, and J. E. Colino. Data wrangling: The challenging yourney from the wild to the lake. In *CIDR*, 2015.

[21] P. Venetis et al. Recovering Semantics of Tables on the Web. *PVLDB*, 4(9):528–538, 2011.

[22] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: Entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD'12*, pages 97–108, 2012.

[23] M. Zhang and K. Chakrabarti. Infogather+: Semantic matching and annotation of numeric and time-varying attributes in web tables. In *SIGMOD '13*, pages 145–156, 2013.